BagStack Classification for Data Imbalance Problems with Application to Defect Detection and Labeling in Semiconductor Units

by

Bashar Muneer Haddad

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved January 2019 by the
Graduate Supervisory Committee:

Lina Karam, Chair
Baoxin Li
Jingrui He
Pavan Turaga

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

Despite the fact that machine learning supports the development of computer vision applications by shortening the development cycle, finding a general learning algorithm that solves a wide range of applications is still bounded by the ”no free lunch theorem”. The search for the right algorithm to solve a specific problem is driven by the problem itself, the data availability and many other requirements. Automated visual inspection (AVI) systems represent a major part of these challenging computer vision applications. They are gaining growing interest in the manufacturing industry to detect defective products and keep these from reaching customers. The process of defect detection and classification in semiconductor units is challenging due to different acceptable variations that the manufacturing process introduces. Other variations are also typically introduced when using optical inspection systems due to changes in lighting conditions and misalignment of the imaged units, which makes the defect detection process more challenging.

In this thesis, a BagStack classification framework is proposed, which makes use of stacking and bagging concepts to handle both variance and bias errors. The classifier is designed to handle the data imbalance and overfitting problems by adaptively transforming the multi-class classification problem into multiple binary classification problems, applying a bagging approach to train a set of base learners for each specific problem, adaptively specifying the number of base learners assigned to each problem, adaptively specifying the number of samples to use from each class, applying a novel data-imbalance aware cross-validation technique to generate the meta-data while taking into account the data imbalance problem at the meta-data level and, finally, using a multi-response random forest regression classifier as a meta-classifier. The BagStack classifier makes use of multiple features to solve the defect classification problem. In order to detect defects, a locally adaptive statistical background modeling is proposed.

The proposed BagStack classifier outperforms state-of-the-art image classification techniques on our dataset in terms of overall classification accuracy and average per-class classification accuracy. The proposed detection method achieves high performance on the considered dataset in terms of recall and precision.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to all those people who made this dissertation possible and because of whom my graduate experience has been one that I will cherish forever.

I would like to thank my advisor Dr. Lina Karam for her support, patience, suggestions, and encouragement throughout my graduate studies. Her guidance helped me in all the time of research and writing of this thesis.

I would like to thank Nital Patel at Intel Corporation. He provided me with necessary assistance and has been always there to listen and give advice. I am deeply grateful to him for giving me the opportunity to join his team at Intel and worked on real industrial research problem.

Most importantly, none of this would have been possible without the love, care, and patience of my family here in the USA and back in Jordan. My wife supported me during all the hard times. My family back in Jordan has been a constant source of love, concern, support and strength all these years.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

The mutually beneficial relationship between machine learning and computer vision has significantly increased during the last two decades driven by the improvements of training algorithms, data availability and hardware performance. Machine learning is the science of giving computers the ability to learn without being explicitly programmed. Machine learning uses algorithms to parse, process, learn from data, and finally make a prediction about unseen data in the real world. Computer vision is the science of giving machines the capability to visually sense and recognize the world around them. It is mainly concerned with automatic extracting, analyzing and understanding useful information from a single image or a sequence of images. It involves the development of theoretical and algorithmic methods to achieve automatic visual understanding.

The applications of computer vision are numerous, including face recognition, defect detection, object recognition, autonomous vehicles, biomedical applications, text recognition, image restoration, remote sensing and surveillance among others [1]. In fact, the range of computer vision applications is very large, and methods carried out can be very different, inspired from physics, biology, statistics theory, functional analysis, and other disciplines. Different applications have different characteristics, which makes a specific machine learning algorithm or a subset of algorithms only useful to learn a specific application or specific problem. The *No Free Lunch Theorem* [2], implies that it is not possible to have a universal machine learning algorithm that can learn and consistently perform better than all other machine learning algorithms for all different tasks (problems). It is important to notice that the *no free lunch theorem* considers the whole problem space, that is, all the possible learning tasks; while in real practice, we are usually interested in a specific task (problem) or a very limited portion of the whole space. Thus, finding a machine learning algorithm that can perform better than all other algorithms on this task is possible and the search is valid.

Object detection and classification are among the important applications of computer vision. Object detection is the process of detecting objects of interest in an image or video. Classification is the task of assigning to an input image or a detected object one label from a fixed set of categories. Image classification has a large variety of practical applications. Among these applications, Automated Visual Inspection (AVI) systems are gaining growing interest in the manufacturing industry.

An AVI system is a mechanized one that is used to perform quality control by using cameras connected to an intelligent system. Inspection is an important process to detect defective products and keep these from reaching customers. Humans can be engaged in the inspection process but, due to issues, such as tiredness, lack of consistency, and boredom, their performance is often unreliable. In some cases, such as when components to be inspected are very small and the production rate is very high, the use of manual inspection is not possible. Therefore, AVI systems started playing important roles in advanced manufacturing to guarantee the quality of products in a timely manner[3].

In the last few years, computer vision and more precisely object detection and classification have progressed immensely. This significant improvement has been achieved by the power of data, using thousands and millions of examples, images and instances for training to fit gigantic models. However, learning from small datasets with just few hundreds or in some cases tens of images is still a challenge. Learning from small datasets becomes a more difficult problem when it is augmented with other challenging problems, e.g., data imbalance, variations within the class, and similarities between classes among others. Dealing with small datasets is not a problem if we can define a set of features that can be used efficiently to separate between classes and if suitable training algorithms to perform the training are also available. Selecting such a feature set and finding such algorithms can be challenging.

In this work, a BagStack classifier is proposed for data imbalance problems, unequal number of samples belonging to different classes, in which the proposed classifier uses the concepts of bagging and stacking to tackle both variance and bias errors. The classifier is designed to learn from small datasets and to effectively deal with data imbalance.

1.1 The Classification Problem

In machine learning, it is well known that applying different learning algorithms, e.g., support vector machine [1], decision tree [4] and Naïve Bayes classifier [1], can result in performance variations. Similarly, applying the same learning algorithm to different datasets can lead to different performance results. It is typically difficult to tell in advance which algorithm can achieve the best performance for a specific problem. The main challenge in dealing with this dilemma is that the true data generating distribution P_0 of the problem is unknown. Designing one learning algorithm

that can deal with different variations of true data generating distributions is a difficult task, if not impossible. In this section, the characteristics of computer vision problems (image classification) are discussed from the machine and statistical learning perspectives, including some of the main characteristics that affect the simplicity/complexity of the proposed machine learning solutions. Such characteristics include the number of available samples, data imbalance, multi-class, within-class variations and inter-class similarities, variance and bias errors.

Number of available samples: It is well known that having more samples, in general, improves the training accuracy and reduces the generalization error. Let us define L to be the total number of samples available for both learning and testing. $L = r + v + t$, where t, r and v are the numbers of samples used for testing, training and validation, respectively. Let $P_{n,Tr}$, $P_{n,V}$ and $P_{n,Tst}$ be training, validation and testing distributions, respectively.

The convergence between the true data generating distribution and these distributions depends on the total number of samples. Having a small number of samples increases the divergence between P_0 (the true data generating distribution) and these distributions. Then, the $KLD(P_{n,Tr}, P_0)$, $KLD(P_{n,V}, P_0)$ and $KLD(P_{n,Tst}, P_0) \rightarrow \infty$ as $L \rightarrow 0$, where KLD is the Kullback–Leibler divergence [5]. If L is very small, the distributions are not good representatives of P_0. But, how large the KLD will be? The answer to this question is a function of the variation in the true data generating distribution P_0. In practice, we do not have access to this distribution (P_0), thus we do not have access to the true variation in the distribution P_0. The simplest way to estimate this variation is by using the available number of samples L. Here one can argue the following. **First**, the estimated variation based on the given number of samples tells only if more samples are needed. It does not tell how many samples are required to solve the problem and achieve acceptable accuracy over the true data generating distribution (P_0). **Second**, the given samples should be randomly sampled from the true data distribution to give accurate estimation of the variation. **Third**, the distribution P_0 should not be changing as a function of other variables, e.g., time.

As $L \rightarrow \infty$, all $r,\ v\ and\ t\ \rightarrow \infty$. If these samples are randomly sampled from the true data generating distribution P_0, the empirical distributions $P_{n,Tr},\ P_{n,V},\ and\ P_{n,Test}\ \rightarrow P_0$. This convergence between empirical distributions and true data generating distribution is beneficial for both training and testing. In addition to this convergence, having more samples gives more freedom to increase the system complexity without worrying about data overfitting. Increasing the complexity

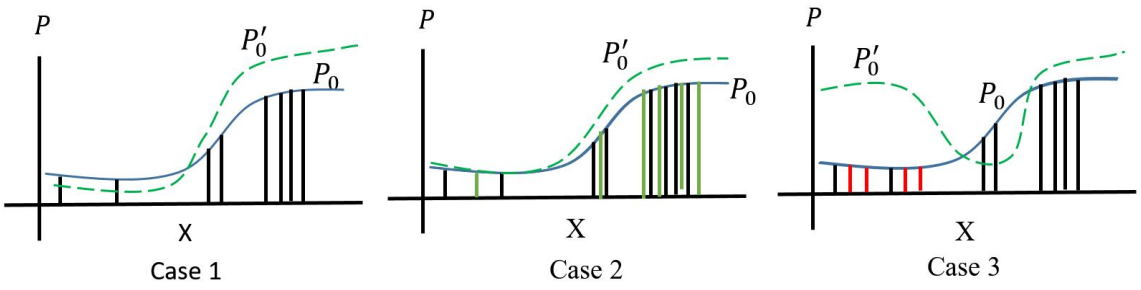

Figure 1: Examples of adding more samples to the observations' set.

of the system and the number of training samples improves the ability of the system to capture all complex and real variations (in contrast to variations due to noise) within the data.

The assumption that adding more samples improves the training accuracy can be argued. In fact, there is no guarantee that adding more samples will improve the generalization accuracy over the true data generating distribution. Figure 1 shows 2 different approaches (Case 2 and Case 3) for adding more samples to the set of observations. In Figure 1.1, Case 1 corresponds to the initial number of samples. P_0 is the true data generating distribution and $P_0^{'}$ is the estimated empirical distribution based on the given observations. Each vertical line corresponds to one added observation. In Case 2, samples are added in a random way. This addition improves the estimation. In Case 3, most of the samples are sampled from a small range of X. It is a biased sampling process. This leads to a less accurate estimation.

To absorb this argument, the following considerations should be taken into account: (1) P_0 is an unknown distribution and can only be estimated closely when $L \rightarrow \infty$ (very large number of samples); (2) the variation within the true data generating distribution P_0 is reasonable, most of the tasks that we consider are structured tasks (not random tasks); (3) samples are added to the dataset by randomly sampling from the distribution P_0; (4) the number of training samples should be sufficient to deal with the complexity of the task and should be directly proportional to the variation within the true data generating distribution P_0.

Data Imbalance: Data imbalance is a terminology used to describe the situation where the classes are not represented equally. It is the case when different classes have a significantly different number of samples. If the data samples are generated by randomly sampling from the true data generating distribution, then the true data generating distribution is a non-uniform distribution.

Due to data imbalance, the training algorithm gives more attention to the class(es) with the

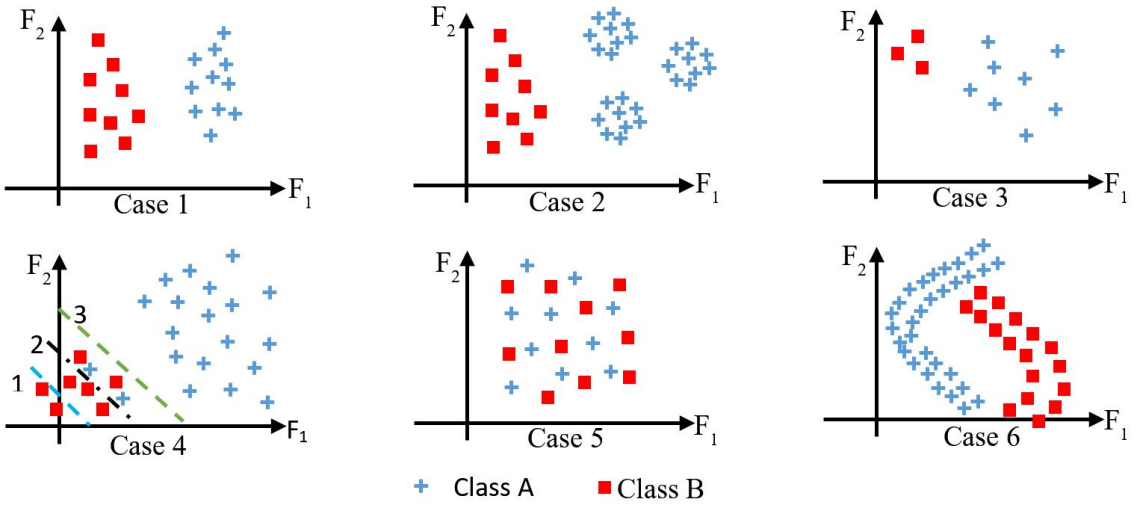

Figure 2: Examples of different classification problems.

majority of samples, which results in biased classifiers, i.e., classifiers that give more attention to samples belonging to the majority classes. If the empirical training and testing distributions are identical to the true data generating distribution, then the training process is doing the right thing by minimizing the error on the training set. Minimizing the error on the training data results in better accuracy on the testing data set in this case. Data imbalance is not a problem if: (1) classes are separable, (2) the number of available samples is large enough to cover all aspects of the true data generating distribution, in which case undersampling can be used to balance the data, and (3) the variation within the majority class is very low, which means that most of the given samples (majority class) are just redundant and do not add significant information content.

Data imbalance is a problem if all classes are important and even more so if the minority class is more important (this is very common in defect, anomaly, and fraud detection). In practice, data imbalance can be solved by either undersampling [6] or oversampling (changing the given distribution) [7]. Another approach consists of using different costs for different classes [8].

Figure 2 depicts six different classification problems. All these problems are binary classification problems. In Case 1, there is no data imbalance; the problem is linearly separable. In Case 2, there is data imbalance, but it is not a problem. The variation within the majority class (class A) is small, i.e., most of the samples are redundant samples and the two classes are linearly separable. In Case 3, there is data imbalance, but again it is not a problem, since the two classes are linearly separable. In Case 4, there is data imbalance and the two classes are not linearly separable. Both lines 2 and 3 result in the same classification error rate (two misclassifications). Selecting line 2 guarantees that

all samples belonging to class A are correctly classified, while selecting line 3 guarantees that all samples belonging to class B are correctly classified. In Case 5, there is a data imbalance problem. It cannot be solved by over- under- sampling. The two classes are not separable. The variations within the two classes are very significant. In Case 6, there is no data imbalance problem, but the two classes are not linearly separable.

The data imbalance is considered as a problem based on the task at hand and the used performance metric. In general, we are interested in two performance metrics: overall accuracy and average per-class accuracy, also denoted as average accuracy for short. Let us β be a learned function (classifier or predictor) that is obtained as a result of a training process. The overall accuracy (OA) is the total number of samples that are classified correctly divided by the total number of samples (n) and it can be expressed as:

$$OA = \frac{\sum_{i=1}^{n} I(\beta(X_i) = Y_i)}{n} \tag{1.1}$$

where $I(.)$ is equal 1 if $\beta(X_i) = Y_i$ and 0 otherwise, and n is the total number of samples. The average per-class accuracy (AA) is given by:

$$AA = \frac{\sum_{c=1}^{C} \frac{\sum_{i=1,Y_i=C}^{n} I(\beta(X_i)=Y_i)}{n_c}}{C} \tag{1.2}$$

where n_c is the total number of samples belonging to Class c and C is the number of classes.

Usually, data imbalance is not a problem if the main performance metric that we are interested in is the overall accuracy. In this case, as long as the true data generating distribution, the training distribution and the testing distribution are identical, a training process that minimizes the loss value on the training data minimizes the generalization error as well. If the success criteria that we are interested in is the average accuracy, data imbalance is a problem and needs to be solved before starting the training process to avoid biased classifiers.

Multi-class classification problem: Multi-Class (MC) classification is the problem of assigning a label $y \in \{y_1, y_2, y_3, ..., y_C\}$ to an input data vector $\bar{X}$, where C is the number of classes. The MC classification problem cannot be solved by using ONE simple binary classifier, e.g., SVM. Having only two classes requires only one flag to distinguish between them; this flag can be either 0 or 1, positive or negative. The MC classification problem needs multiple flags to distinguish between classes. Converting the MC classification problem into multiple binary classification problems can be implemented by using either one-vs-one or one-vs-all binary classifications. These two methods are very common in machine learning, even though they are not the only ones.

If one-vs-one binary classifiers are used to implement the MC problem, then the total number of required classifiers is $\sum_{i=2}^{C}(i-1) = (C-1)+(C-2)+...+1$, where C is the total number of classes. Consequently, the number of binary classifiers to be used to implement the MC problem can be quite large $\sim C^2$ and the collinearity between these classifiers is very high. During testing, a sample labeled as Y is tested by a classifier that is trained to solve the problem ($Label\ J\ vs.\ Label\ Q$). This classifier has not been trained on any samples of class Y. in fact, this is the first time that this classifier is seeing samples from this category and it is completely unaware of it. The classification (regression) is completely random, given that the $Class_J$ and the $Class_Q$ are completely independent of $Class_Y$.

If one-vs-all binarization is used to implement the MC problem, then the total number of required classifiers is $\sum_{C_i=1}^{C} 1$, where C is the number of classes. The binary classifier must deal with a complex binary classification problem (one-vs-all), where the variation within the class (all) is very high. This is highly expected since the class (all) is a combination of samples from different classes as defined by the true data generating distribution. Merging different samples that belong to different classes into one class converts a balanced problem into an imbalanced problem. Solving the problem of imbalanced data by using oversampling or undersampling can lead to using redundant information or information loss, respectively.

Variations and similarities: Variations within the same class and similarities between classes give an indication about the complexity of the problem. These two properties result from the distribution P_0. Variation within the class increases the demand for having more samples of the same class in the training process. More samples are required to cover the variation. Variation within the same class makes the ability of the model to generalize for new samples belonging to the same class more difficult. Similarity between classes adds more complexity to the system that is required to solve the problem. Increasing the complexity necessitates having more training samples to avoid overfitting. The main challenge/concern about either variation within class or similarity between classes is that the true data generating distribution is unknown and, as a result, the level of variation and/or similarity cannot be determined accurately. Variations and similarities can only be estimated by using the training data (the given samples).

None-zero generalization error: The generalization error consists of three main components, variance error, bias error and another term that is related to the randomness in the true data gener-

ating distribution. The generalization error is given by [9]:

$$GeneralizationError = Variance + (Bias)^2 + irreducible\ error \tag{1.3}$$

Achieving zero bias and zero variance depends on many factors including the complexity of the classification problem, the quality of the data, the quality of the data representation, the number of samples, similarities between classes, variation within classes, the distribution of different classes and the available training algorithms. The variance error can be reduced by using ensemble classifiers (Bagging) [9], bias error can be solved by changing the training algorithm, the data representation or by using ensemble classifiers [9]. If all available models (models that have been trained by using different available training algorithms and different data representations) have non-zero variance error and non-zero bias error, achieving high classification accuracy becomes more challenging. The proposed BagStack classifier has been designed to minimize both variance and bias errors when:

- All models that are generated by using different training algorithms and different data representations have non-zero bias and non-zero variance error.
- A small dataset, and/or data imbalance, and/or variations within classes, and/or similarities between classes exist.

1.2 Industrial Application: Defect Detection and Classification in Semiconductor Units

Automated inspection systems play an important role in manufacturing to guarantee higher quality and reduce production costs. In the semiconductor manufacturing industry, assembly and testing processes are getting more complex, resulting in a greater tendency of defects to impact the production process. These defects can cause field failures and can result in customer dissatisfactions and returns, which makes it important to have reliable inspection systems to control and guarantee the product quality [10].

Currently available defect detection and classification systems are customized and hard-wired to the detection of particular classes of defects and cannot deal with new unknown classes of defects. The inability to adapt to new types of defects results in additional handling and additional inspection time depending on the number and types of defects being inspected, and in the failure of detecting

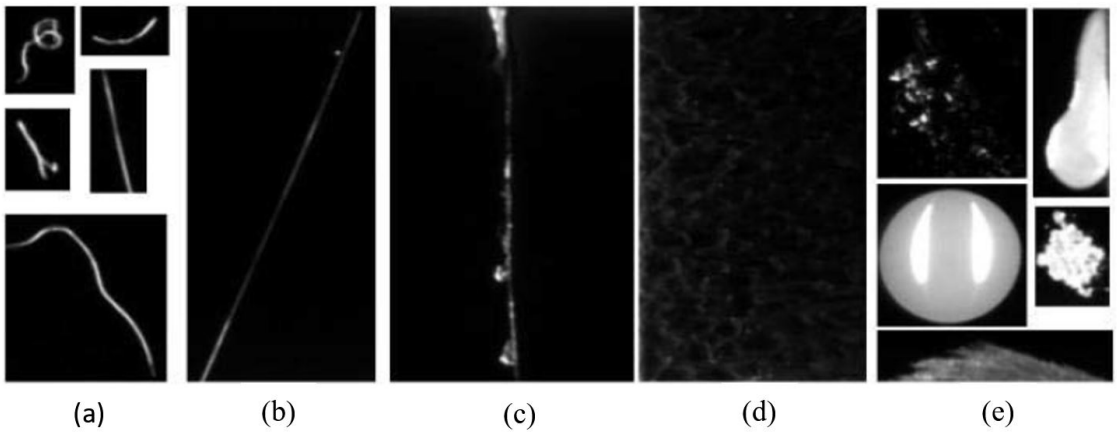

Figure 3: Defects in the die area. (a) Foreign material on die. (b) Scratch on die. (c) Crack on die. (d) Fingerprint on die. (e) Epoxy on die.

new or unknown defects [11]. Dealing with new unknown defects necessitates redesigning the existing automated algorithms to detect and classify the new defects, which causes long development cycles, and delays, and which requires human effort to support constantly the development of the system.

Detection and classification tools can provide important information about defects' types, sizes, locations, distributions, and repetitions. Determining the stage of production responsible for the fault and classifying these defects are important when corrective actions need to be taken. Data mining methods have been used successfully in the semiconductor manufacturing process; however, no data mining method was proposed for defect detection and classification for semiconductor units in the manufacturing industry. Some prior works applied data mining methods to assessing the quality of semiconductor wafers and equipment but not semiconductor units [12, 13].

Defects can occur due to a variety of causes, including unintended human interaction, failures of machines that are used in the manufacturing process, low quality materials, or any unexpected events, such as power failure. Defects can affect one or more parts of the semiconductor unit, such as the die, epoxy, and substrate regions of the unit. The types of defects may differ depending on the manufacturing stage. Some of these defect types include cracks, fingerprints, epoxy on die, and defects due to foreign material, such as hair, threads, fibers, dust, or fluids. Figure 3 depicts some sample defects. Some challenges include the shortage of defective units, imbalanced data, wide variation within the same defect category, and similarities between different defect categories.

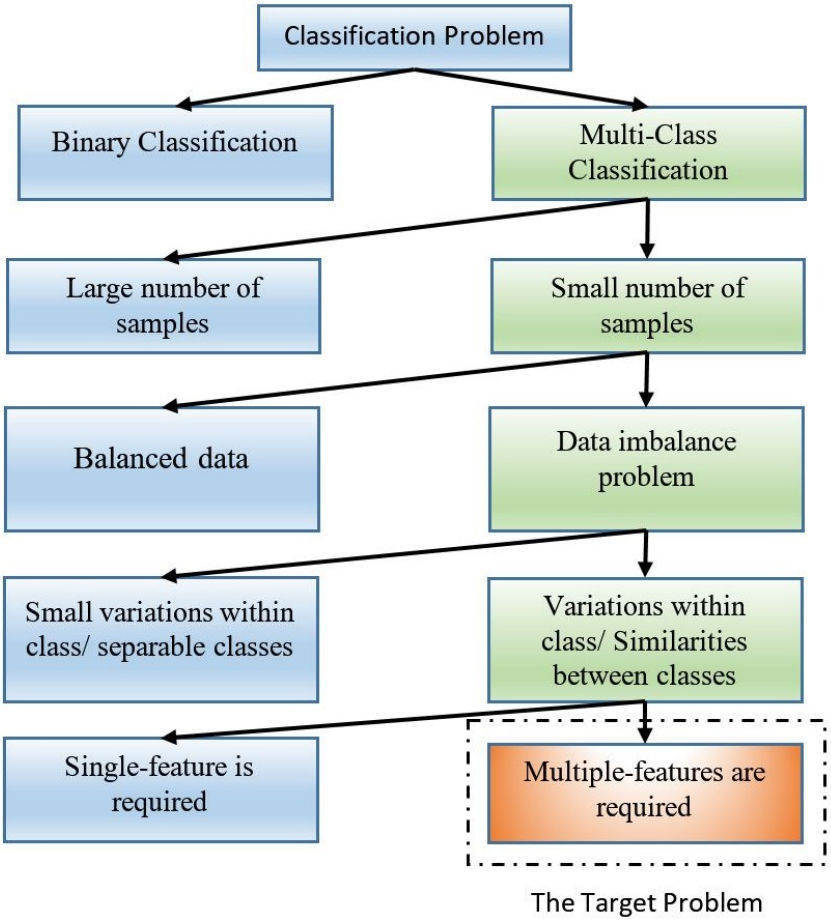

Figure 4: The characteristics of the problem.

Defect classification in semiconductor units is a challenging problem. The main objective to be achieved is to reduce underrejection and overrejection rates. The performance metric is defined as the average accuracy. Figure 4 depicts the different classification problems and shows to which category the target application belongs.

1.3 Contributions

The first contribution of this research is to design a classifier that can handle data imbalance when the number of available samples is very small. The proposed BagStack classifier uses the concepts of bagging and stacking to reduce both variance and bias errors.

In the process of pursuing these goals, several techniques are proposed to handle the data imbalance problem. The multi-class classification problem is transformed into either one-vs-one or

one-vs-all multiple binary classification problems. Many base learners are assigned to solve each problem. Bagging is used to train the subset of base learners assigned to the same problem. The number of base learners is adaptively specified to solve each sub-problem (the binary classification) and the number of samples to use to train each base learner. Experimental results show that the proposed BagStack classifier can outperform existing state-of-the-art classifiers for different tasks, especially in the case of the ones when a single training algorithm is not sufficient to achieve good accuracy and the set of available base learners results in a non-zero bias error, i.e., when all available training algorithms are linear, and the classes are non-linearly separable. Stacking uses many training algorithms that have different bias errors (non-zero and independent) to transform the problem from the input space to the meta-data space where the meta-classifier may have better opportunity to achieve a higher accuracy.

If the training data is imbalanced and if the standard K-fold cross-validation is used to generate the meta-data, such that we re-train/re-test the base learners K times and each time $K-1$ folds are used to train the base learners and the last fold is used to generate the meta-data, then the meta-data will have the same distribution as the training data. Using imbalanced meta-data to train the meta-classifier results in a biased meta-classifier. In this research, a simple, yet very efficient, technique is proposed to handle the data imbalance problem at the meta-data level. A novel imbalance-aware $K-fold-M-split$ cross validation technique is proposed to generate the meta-data that adapts to the imbalance characteristics of the data for imbalance data problem. The proposed approach adaptively increases the number of folds (K) to handle the data imbalance problem at the meta-level. Imbalance-aware cross validation is different from the well-known standard K-fold cross validation in two aspects. First, the training set is divided into C (the number of classes) subsets, where each subset contains all samples belonging to the same class, each subset is divided into M splits, and the resulting folds are combined back such that: C splits (one-split from each class is randomly selected) are used to construct the validation set, and all other $C \times (M-1)$ remaining splits are used to construct the training set. Second, the random process of sampling of splits, constructing training and validation sets, and performing training and testing is repeated for K times, which depends on the maximum imbalance ratio between classes (majority/minority). This number (K) may exceed the number of splits (M). It is worth mentioning that the standard $K-fold$ cross validation is just a special case of the imbalance aware cross validation where $M=K$. Experimental results show that imbalance-aware cross-validation increases the average accuracy of

many classification problems with either an increase or a slight reduction in overall accuracy. The proposed approach has been tested on many datasets to validate its efficiency. Imbalance-aware cross validation is a replacement for cross-validation in the context of stack-based classifiers.

Specifying the number of base learners adaptively may lead to a large number of base learners resulting in a high computational complexity. The complexity of the proposed approach can be constrained by setting two main constraints: the coverage percentage, which represents the number of samples (belonging to the majority class) that are used in the training process divided by the total number of samples belonging to the majority class, and the maximum number of base learners. These constraints may lead to requiring, for the training of each base learner, a number of training samples that is larger than the number of samples in the minority class and that is lower than the number of samples in the majority class. Instead of using the standard oversampling / undersampling, a new sampling technique is proposed, Variation-Based and Adaptive-Synthetic Minority Oversampling Technique (VA-SMOTE), to handle the issue when synthetic data from a minority class invade the majority class too deeply (creating isolated islands) by considering the variation within the minority class' samples. VA-SMOTE is a variance-based, adaptive synthetic minority technique. It is a variation of the SMOTE algorithm [14]. The main contribution is designing a synthetic-based oversampling technique and adaptive-undersampling technique that can be used to achieve data balance to deal with severe data imbalance problem and the presence of large variations within the minority class.

A novel approach for defect detection and classification in semiconductor units is proposed. The proposed system consists of three stages: proposal generation stage (change detection), defect detection stage and refinement stage. In the proposal generation stage, changes on the target unit are detected using a novel change detection approach. In the second stage, a deep neural network is used to classify detected regions into either defective or non-defective regions. Non-defective regions are regions exhibiting allowable changes due to factors such as lighting conditions and subtle differences in manufacturing. The defect detection stage achieves up to 94.3% accuracy. In practice, defects that are smaller than a specified tolerance size are ignored by manufactures. The tolerance size depends on the defect types and is determined based on risk factors. In order to ignore such defects, the proposed approach includes a final refinement stage wherein the detected defects are categorized by a stacking-based ensemble classifier into different classes. Defects

smaller than their corresponding tolerance size are ignored. The refined system achieves up to 97.88% overall detection accuracy.

To the best of our knowledge, this is the first work that uses machine learning to address the problem of defect detection and classification at the level of semiconductor units, wherein defect detection is performed in an automatic way and all regions of the unit are included. The performance of the proposed approach is tested on images of real semiconductors units from Intel. The proposed approach was shown to result in a high detection and classification accuracy. The proposed system was able to achieve a 98.2% precision and a 99.44% recall values. The proposed method can handle different types of acceptable variations (e.g., light variations, epoxy variations). The proposed method can ignore these differences while still maintaining the ability to detect the actual defects. Although the proposed change detection system is designed specifically for semiconductor units, it can also be applied to other problems (e.g., surface defect detections) that have similar types of nuisance variations (e.g., lighting conditions and material physical properties).

1.4 Organization

This thesis is organized as follows. Chapter 2 presents a detailed literature review for different ensemble-based classifiers, including bagging, boosting and stacking. In the literature review, the data imbalance problem and the rational behind bagging and stacking are also presented.

Chapter 3 presents the proposed BagStack classifier. This chapter covers the main contributions of the BagStack classifier and explains the rational behind the design. In the last section of Chapter 3, a detailed analysis and evaluation of the proposed classifier are presented. The evaluation consists of different experiments that cover different parameters to show the effect of changing each one of them. The results verify the ability of the BagStack classifier to outperform many other state-of-the-art ensemble classifiers.

Chapter 4 presents a multi-feature sparse based approach for defect classification in semiconductor units. In this chapter, the initial version of the BagStack classifier is covered. In this initial version, cross-validation are not used to generate the meta-data, most of the parameters that are used to control the behavior of the classifier are selected manually through a heuristic search / optimization method. The initial version was only tested using defects on the die region. Defects are cropped manually. In Chapter 5, the full framework for defect detection and classification is pre-

sented. A locally adaptive statistical background modeling for change detection is proposed. The detected changes are processed by using a DNN-based solution to classify the detected changes into either defective or non-defective images. In the last stage, the BagStack classifier is used to classify detected defective images into one of the defects' classes. In Chapter 6, another application is introduced, namely defect detection and classification on steel surfaces, to prove the capability of the BagStack classifier. Finally, Chapter 7 summarizes the results, contributions and future research directions.

Chapter 2

BACKGROUND AND LITERATURE REVIEW

This chapter covers the background and literature review of ensemble classifiers, bagging, boosting and stacking-based classifiers. A brief description of the most common used boosting and bagging techniques is presented. A detailed description of stacking-based classifiers is presented. This chapter covers also the data imbalance problem and the main contributions in the literature to deal with it. In this chapter, the literature review of the following topics are covered: (1) Ensemble classifiers, bagging and boosting; (2) Stacking-based Classifier; (3) Data imbalance problem, sampling techniques and ensemble classifiers for data imbalance problem; (4) The rational behind bagging and stacking ensembles.

2.1 Ensemble Methods: Bagging and Boosting

In machine learning, ensemble classifiers are learning algorithms that are constructed by combining weak classifiers. In ensemble learning, a classification task is learned by dividing it to smaller subtasks. The idea of using ensembles to tackle different problems has gained significant interest in the last two decades. Usually, training these weak classifiers is less expensive and easier than training strong classifiers. The performance of ensemble classifiers depends on the quality and diversity of their weak (base) learners. Diverse classifiers generate independent and uncorrelated errors. There is a trade-off between these two properties, two accurate classifiers are less probable to be diverse and two diverse classifiers cannot at the same time both be very accurate.

2.1.1 Bagging

Bootstrap Aggregating (Bagging) [9] is an ensemble classifier that uses bootstrap sampling to generate subsets of the training data and train multiple weak classifiers to reduce the variance error component in the bias-variance equation. Bootstrapping is a general sampling technique in which the data is divided into several non-disjoint training sets by drawing randomly, with replacement from the data. Bagging produces the final prediction by combining (averaging or majority voting)

the predictions of its base learners (weak learners). Bagging ensemble is the right candidate to improve classification accuracy, when available training algorithms are unstable for a given training set. The reduction in variance error is proportional to the number of used base learners [15].

Many works in the literature, [4, 16, 17, 18, 19, 20] mathematically explain why and how bagging works. It is well known that bagging misclassification error converges to an asymptotic value as the number of base learners increases. Many works in the literature focused on understanding the relation between the ensemble size and this asymptotic value as well as how to specify the optimal number of base learners to construct bagging-based ensemble classifier.

Bagging experienced different variations [21]; these variations can be categorized into one of the following: using different bootstrap sampling schemes, using subset of features, using different voting schemes, adding noise to the inputs, using heterogeneous base learners instead of homogeneous ones and methods that use bagging in online environments. For more details, the reader is referred to [21, 22, 23, 24, 25, 26, 27].

2.1.2 Boosting

The first Boosting procedure was proposed in 1990 by Schapire [28]. The main contribution of this work is to show that strong classifiers can be built by combining weak classifiers. The boosting procedure described in [28] is very simple, using only three base learners, the data set is divided into three partitions as follows. For a given instance, if the first two classifiers agree on the class label, this is the final decision for that instance. The set of instances on which they disagree defines the third partition which is used to train the third classifier. Based on this description an important difference between bagging and boosting is emerged. In bagging, different base learners are independent, they can be trained in parallel. In boosting, the probability of a specific instance to be part of a particular classifier's training set depends on the performance of previous classifiers on that instance. In fact, the probability distribution used to sample form the training set is updated such that instances that miss-classified by previous classifiers will have higher probability to be selected for training the current classifier. Thus, boosting tries to generate new classifiers that are able to better classify the 'hard' instances for the previous ensemble members.

The most significant variant of boosting algorithm is the one proposed by Schapire and Freund, Adaptive Boosting, known as AdaBoost [29]. The proposed algorithm improves the classification

accuracy of algorithms by combining several (large number of) base learners, each is trained using the same learning algorithm (same bias) but different training examples. AdaBoost uses weighted versions of the same training data instead of randomly subsamples. The same training set is repeatedly used. The algorithm learns a set of classifiers that are obtained sequentially, using re-weighted versions of the training data, with the weights depending on the accuracy of the previous classifiers.

Increasing the weights of misclassified samples allows the weak learner at each iteration to focus on instances that were not correctly classified by previous classifiers. It is important to choose weak learners (e.g. linear classifiers) to obtain the base classifiers, allowing them to learn without decreasing significantly the weight of previously correctly classified instances. If the base learner is too strong, it may achieve high accuracy, leaving only outliers and noisy instances with significant weight to be learned in the following rounds [23]. The AdaBoost algorithm is now a well-known and deeply studied method to build ensembles of classifiers with very good performance. One of the main strength of using boosting is the ability of inducing diversity even for stable classifiers by changing the focus on different samples of the training set. This main strength can become a drawback if the data is noisy. Many works addressed the question of which is better bagging or boosting? And what are the circumstances that any of them overcome the other? [30, 31]. Boosting performance is explained based on margin-based discriminative learning [32, 33].

One of the main drawbacks of boosting is that it is quite susceptible to noise. Many studies [34, 35, 36, 37] have shown that. One of these studies is the one carried by Dietterich [34]. In his experiment, he evaluates both of bagging and boosting using some standard datasets. He investigates the accuracy of these methods in relation to the addition of classification noise to the training data. AdaBoost experiences a significant reduction in performance comparing to bagging as we increase the level of the noise. As it is expected, AdaBoost will focus more on misclassified instances, these few abnormal instances, which leads to overfitting problem.

Even though the most significant variation of boosting is the AdaBoost, many other variants have been published in literature to address some of the main problems we mentioned above. In general, these variations can be categorized into one of the followings. Methods that use different sampling techniques, methods that use subsets of the features, methods that use different voting rules, methods that can handle noisy data, methods that apply boosting based on the characteristics of test instances and methods that apply boosting in online environments.

Real AdaBoost [38, 39], Logit Boost [40], Gentle AdaBoost [41], Modest AdaBoost [42], Float

Boost [43], Emphasize Boost [44] and Reweight Boost [45] are some of these variations that have significant contributions. Boosting algorithm, represented by AdaBoost and other variants, has been successfully used in many real applications. Optical character recognition, pedestrian detection, text categorization, speech and face detection are among the others. For more details, the reader is referred to [21, 22, 23, 24, 25, 26, 27].

2.2 Stacking Based Classifier

Bagging and boosting ensembles are used to combine **homogeneous** base learners. On the other hand, Stacking is used to generate ensembles of **heterogeneous** classifiers. Recently, it has been proven that using stacking to mix different models that have different biases can outperform other ensemble techniques. Stacking has been used in many competitions. Netflix competition is one among the others [46, 47, 48].

Known in the literature by either stacking, meta-learning, blending classifier or mixing, stacking was initially presented by Wolbert in 1992 under the name of Staked Generalization [49]. In his paper, Wolbert described stacked generalization algorithm. He used the word black art to describe different components, aspects, parts and decisions in his work. Many questions have been emerged. In general, research related to stacking based classifier can be divided into two major parts. The first era 1992 to 2007 is best described by the word **"experimentally"**. Many researchers worked to answer some of the "Black Art" questions carried out by Wolbert. In 2007, a paper "Super Learner" [50] published by Mark J. van der Laan, Eric C. Polley et al proved mathematically the rational behind stacking "the relation between general cross validation and stacking". The second era 2007 to 2017 is best described by **"heuristic search"**. Many researchers in this era used different optimization and heuristic search algorithms to handle the challenge of finding the best configuration of a stack-based classifier given a set of classifiers (training algorithms of different inherited biased). In this section we will present a detailed literature review for stack based classifier. In literature, there are many surveys [51, 52] that cover the development of stacking ensemble and its variations. However, up to our knowledge we could not find any comprehensive survey that covers these two eras. In this section we will present a comprehensive survey to cover the most important advancements in stacking-based ensembles.

The stacking based classifier method is concerned with combining multiple classifiers generated

by using different learning algorithms $L_1,\ L_2,\ ....\ L_N$ on a single data set S which consists of examples $S_i = (X_i, Y_i)$, i.e., pairs of feature vector (X_i) and their classification (Y_i). In the first phase a set of base-level classifiers $C_1,\ C_2,\ ...\ C_N$ is generated, $C_i = L_i(S)$. In the second phase, a meta-level classifier is trained that combines the outputs of the base level classifiers.

To generate a training set for learning the meta-classifier, a leave-one-out or a cross validation procedure is applied. In leave-one-out, the training set $S = (X_i, Y_i)_{i=1}^{i=m}$, where m is the number of samples, is used to generate m training-validation sets, each training set contains $m-1$ samples and each testing set contains one sample. $\forall_i = 1, 2, 3,\ ...\ m,\ \forall_k = 1, 2, 3,\ ...\ N:\ C_k^i = L_k(S - s_i)$ We then use the learned classifier to generate the prediction for $s_i : \widehat{y}_k^i = C_k^i(X_i)$. The meta-level data set consists of examples of the form $((\widehat{y}_1^i, \widehat{y}_2^i, ..., \widehat{y}_k^i), Y_i)$ for all $i = 1, 2, ..., m$, where the features are the predictions of the base-level classifiers and the label is the corresponding label of the sample at hand.

By using leave-one-out sampling technique, we need to train the base classifiers $(m+1)$ times. The first m times is to generate the meta-data and the last time is to generate the final base learners that are used for the final testing. Using leave-one-out is expensive, especially when the number of samples is large. On the other hand, if we define the function $Dist(C_1, C_2)$ that measures the difference between base classifiers trained by using samples (enough number) from the same data, the difference between base learners C_i^S that uses the training set T_{train} of $S = m-1$ samples and the C_i^{all} that uses the T_{train} of S samples will be very small. Thus, the quality of the meta-data will be very high, and the meta-classifier is trained using samples very similar to the expected samples in testing.

When having large number of samples, training using leave-one-out is expensive, using $Q-fold$ cross validation is a good alternative. When $Q = m$, the $Q - fold$ cross validation is equivalent to leave-one-out. In $Q - fold$ cross validation, instead of leaving out one example at a time, subset of size $(\frac{1}{Q})$ of the original data set are left out and the predictions of the learned base-level classifiers obtained on these.

Stacking-based classifier is considered as ensemble classifier, where multiple base learners are combined to build a strong classifier (a classifier with better accuracy). The other two ensemble classifiers are Bagging (Bootstrap Aggregating) and Boosting. In bagging and Boosting, the diversity of classifiers is achieved by using different data sets to train each base learner. These techniques use voting (either average or weighted) to find the final predictions. Two major differences between

(Bagging, Boosting) and (Stacking): Bagging and Boosting use homogeneous base learners, base learners that are trained by using the same training algorithm and the difference between base learners as discussed above is in the training set used to train each classifier. Stacking uses heterogeneous classifiers, classifiers that use different learning algorithms (bias). The other difference is related to the way they construct the final prediction. Bagging and Boosting use voting schemes to find the final prediction. On the other hand, stacking uses a meta-classifier to predict the final prediction.

In stacked generalization [49], the concept of stacking has been introduced. The author described the whole process as a black art. Wolbert kept two questions opened: what is the best meta-data to be used? And what is the best meta-classifier to be used?

Most of the research from 1992 to 2007 related to stacking-based classifiers tried to answer these two questions. In 1992, Breiman suggested the stacking regression [53]. Stacking regression is a method for forming linear combination of different predictors (classifiers) to give improved prediction accuracy. The idea is to use cross validation data and least squares under non-negativity constraints to determine the coefficients in the combinations. The final prediction is given by:

$$Y_{final} = \alpha_1 \times X_1 + \alpha_2 \times X_2 + ... + \alpha_k \times X_k \tag{2.1}$$

where k is the number of base learners. In his paper, Breiman used two different regression techniques, linear subset and ridge regression.

Skalak [54] proposed the use of instance-based learning classifiers that store a few prototypes per class as level-0 classifiers. They also proposed to use a decision tree as a meta-classifier or level-1 classifier. Fan [55] used conflict-based accuracy estimates to measure the accuracy of the ensemble generated by Stacking. The authors use two tree-based classifiers and one rule-based classifier as base-level classifiers. In contrast, for the meta-level, they use a rote table that behaves as a decision tree without pruning in this case. This Stacking configuration is evaluated using four datasets (including two artificial datasets). Although the authors claim that the proposed measure is superior to all existing measures, their results do not clearly demonstrate that this estimate can be generalized to more datasets or other meta-classifiers [52].

Ting and Witten [56, 57] addressed the two important questions that were asked by Wolbert in 1992: the type of generalizers that is suitable to derive the higher-level model, and the kind of attributes that should be used as its inputs. They suggested stack-base level classifiers whose predictions are probability distributions (PDs) over the set of class values, rather than single class

values. The meta-level attributes are thus the probabilities of each of the class values returned by each of the base-level classifiers. The authors argue that this allows them to use not only the predictions, but also the confidence of the base-level classifiers.

Each base level classifier predicts a PD over the possible class values, thus, the output of the base classifier is a vector of C classes. This vector is given by:

$$P^i(X) = (P^i(C_1 \setminus X), P^i(C_2 \setminus X), \ .... \ , P^i(C_c \setminus X)) \tag{2.2}$$

where $P^i(C_1 \setminus X)$ is the probability of being class 1 given the example X as predicted by the classifier i. Thus, each meta-data example $(\overline{X}_i, y_i)$ is of dimension $\overline{X}_i = (X_1, X_2, \ ... \ X_{kc})$. The meta-level attributes, which used and suggested by [57], are the probabilities predicted for each possible class by each of the base-level classifiers.

The second important question, which meta-level classifier to be used? has been also answered by [57]. They suggested using Multi-Response Linear Regression MLR. MLR is a modification "adaptation" of linear regression for classification problem. With C classes values $[C_1, C_2, \ ... \ C_c]$, C regression problems are defined, one for each class. For each class, a linear regression equation LR_j is constructed to predict a binary variable, which has value 1 if the class value is C_j and zero otherwise.

Given a new sample X to be classified. $LR_j(X)$ is calculated for each class and the class K predicted with the max value $LR_k(X)$. For each one of these linear regression, the input is a vector of size$[1 \times KC]$, where K is the number of base learners and C is the number of classes. In 2002, MLR was used by Seewald [58] with a different set of meta-data.

In [59], Bagging and Dagging, Ting and Witten investigated the method of stacked generalization in combining models derived from different subsets of a training data sets by using a single learning algorithm, as well as different learning algorithms. Bagging uses Bootstrap, which is a method of sampling with replacement, while Dagging (disjoint aggregating) uses disjoint sampling to generate different training sets to train each base learner. The main contribution of their work is to replace voting scheme, they suggest using higher level learning algorithm to combine the outputs (predictions) of level 1 base learners. They did not apply any cross validation when generating the meta-data, instead they use the whole training set for each of the k-model, despite the fact that subsets of L where used to train those models. In order to achieve high accuracy using stacking based classifier, four areas (issues) should be taken care of: quality of base classifiers, diversity between base classifiers, high quality meta-data and finally the quality of the meta-classifier. In their

work, using the same data to generate the meta-data will affect the quality of the meta-data. In fact, if the sampling percentage is high, this may lead to overfitting problem. On the other hand, using a small portion of the data to generate the base learners in order to avoid the overfitting problem may also lead to low quality base classifiers, since small portion of the available data is used. This tradeoff between the quality of base learners and the quality of the meta-data can be solved by using cross-validation.

Merz suggested SCANN [60] for combining the predictions of level-0 base learners. SCANN uses the strategies of stacking and corresponding analysis to model the relationship between the learning example and their classification by a collection of learned models. Corresponding analysis is a method for geometrically modeling the relationship between the rows and cols of a matrix. A nearest neighbor method is then applied within the resulting representation to classify unseen examples. When the errors are uncorrelated, the optimal approach is to take the majority vote. However, when patterns exist in the errors of the learned models, a more elaborate combining scheme is necessary. SCANN uses stacking and corresponding analysis to model the relationship between learning examples vs. models. More specifically, the way these models classify these samples. The new representation is captured in a space of uncorrelated dimensions. SCANN learning algorithm can be broken into four parts: representation, classification, search and evaluation. Stacking and corresponding analysis are used to create a new representation where SCANN applies KNN.

Dzeroski [61] suggested three different contributions: combining multiple models with meta-decision tree [62], stacking with multi-response model tree and an extended set of meta-level features [63]. In 2000, Dzeroski introduced a new method for meta-learning for combining classifiers with stacking: meta-decision tree has been introduced. The meta-decision tree, which is used as a meta-classifier, has base level classifiers in the leaves instead of class value predictions. It predicts which base learners to trust for prediction instead of predicting the actual label of the example. In [62], a full description of the meta-decision tree training algorithm is presented. Properties (maximum probability, entropy and weight) of the probability distributions predicted by the base-level classifiers are used as meta-level attributes instead of the probability distributions themselves. $MAXProb(X, C)$ is the highest-class probability (the probability of the predicted class) predicted by the base level classifier C for example (X). $Entropy(X, C)$ is the entropy of the class probability distribution predicted by the classifier C for example X. $Weight(S, C)$ is a function of the training examples used by the classifier C to estimate the class distribution for example X, for decision

trees, it is the weight of the examples in the leaf node used to classify the example. For rules, it is the weight of the examples covered by the rules which has been used to classify the example. The entropy and the maximum probability reflect the certainty of the classifier in the predicted class value. If the returned predicted probability is highly spread, the maximum probability will be low, and the entropy will be high, indicating that the classifier is not certain in its prediction. The weight quantities reflect how reliable is the predicted class probability distribution. The higher the weight, the more reliable the estimate. In [63], Dzeroski reported that stacking with $MDTs$ clearly out performs voting and stacking with decision trees as well as boosting and bagging of decisions trees. On the other hand, the proposed method only performs slightly better than SCANN and select the best classifier using cross validation. In [63], the authors did not report and comparison with the best-known method for stacking at that time [57], stacking using probability distributions as meta-data and MLR as meta-classifier. In later research [64], the author reported that $MDTs$ perform slightly worse than stacking with MLR. In general, SCANN, $MDTs$, stacking with MLR and select the best classifier seem to perform at about the same level.

An extended set of meta-level features is suggested by Dzeroski [63] to improve the stacking based classifier accuracy. The same method suggested by Wolbert [49] is used except that cross validation is used to generate the meta-data. The extended set of meta-level features is constructed as follows. First the probability distribution is generated by each classifier (base learner), then a new vector is constructed based on the first one. The new vector results from multiplying the probability distribution vector by the max probability of this distribution, finally the entropy of the probabilities is calculated and used as a meta-data. Overall the new extended meta-data contains $2 \times m + 1$ for each classifier and $N \times (2 \times m + 1)$ for all classifiers. MLR is used to combine these attributes and find the final prediction.

Dzeroski [61] suggested another extension to stacked generalization, stacking with multi-response model trees. In this method, Dzeroski adopted a similar framework to what was proposed by Ting and Witten [57]. Instead of using MLR, he used model tree and kept everything else the same. Instead of using m linear regression equation LR_j,he induced m-model trees MT_j.

Seewald is known for two major contributions: Grading classifiers [65] and Stacking confidence "StackingC" [66]. Grading classifier is a meta-classification scheme. While stacking uses the prediction of level-0 classifiers as meta-data (meta-level attributes), grading classifier uses graded predictions, predictions that have been marked as correct or incorrect as meta-level classes. For

each base classifier, a meta-classifier is trained, whose task is to predict when the base classifier will make error. While stacking can be seen as generalization of cross validation, grading can be seen as generalization of voting. The major differences between stacking and grading are: stacking uses the base-level learners' predictions as meta-data, grading uses the original data as meta-data. Stacking uses one meta-level classifiers to combine the predictions of all classifiers, grading uses K (where k is the number of base-level classifiers) classifiers, each one is corresponding to one of the classifiers in level-0. Stacking uses the original examples labels to train the meta classifier, grading uses graded labels, 1 if the prediction of the corresponding base classifier is correct and 0, if not.

Selection by cross validation simply picks the classifier corresponding to the data set with the fewest wrong examples. Grading tries to make this decision for each example separately by focusing on those base classifiers that are predicted to be correct on this example. During the testing time, if multiple base level classifiers are predicted to be true and have different labels, either voting or selecting the base classifiers of the highest confidence is used to find the final label. On the other hand, if no base level classifier is predicted to be true, all base classifiers are used with using (1-confidence) as the new confidence. Thus, prefer the classifier that are most unsure about the predictions.

StackingC classifier [66] is a stacking-based classifier. It represents an extension to the method suggested by Ting and Witten [57]. Motivated by the fact, as reported by [57], that stacking based classifier performs worse on multi-class data sets. In StackingC, instead of using all probability distributions that are generated by all base learners to train each linear regression, StackingC uses only corresponding predictions of class i to fit the linear regression equation corresponding to class i.

In Ting and Witten [57], given a stack-based classifier of K base learners and m classes. The prediction of each base learner k is a probability distribution in the form of $(P_k(C_1 \setminus X_i), P_k(C_2 \setminus X_i), ..., P_k(C_m \setminus X_i))$. The meta-data is constructed by stacking the predictions of all these base learners. The meta-data is used to fit each one of the linear regression equations. On the other hand, in StackingC, only probabilities corresponding to the same class i, which are generated by different base learners, are used to fit the corresponding linear regression equation LR_i.

In general, the research during the period 1992 to 2007 can be summarized as follows. The main idea of stacking has been introduced by Wolbert in 1992 [49]. Stack-regression has been introduced in 1996 [53]. In 1999, Ting and Witten answered the two important questions related

to Stack-based classifier: probabilities are the best attributes to be used as meta-data and multi-response linear regression is the best classifier to be used as meta-classifier [56, 57]. Most of the other works have no enough contribution, mainly, experimentally-based contributions, provide very slightly improvement, do not address specific problems, except StackingC, which tries to solve the problem related to using stacking-based classifier with multi-class problems.

One of the main conclusion of the research during the period 1992 to 2007, is that there are many contradictory results and there is no consensus on which combination of classifiers (base classifiers and meta-classifier) is the best one. New research direction related to stacking-based classifier emerged based on the last point, given a set of classifiers (different learning algorithms), parameters set and a training data, what is the best configuration to be used?

In [67],the authors addressed the problem of overfitting when using stacking-based classifier, even when using linear models to combine the outputs of the base learners. This problem emphasizes the need for regularization to reduce the overfitting problem and increase the prediction accuracy. They tried three different regularization methods: ridge regularization, lasso regularization and elastic net regularization. Regularization attempts to increase (improve) the predictive accuracy by reducing variance error at the cost of slightly increased the bias error, this is known as the bias-variance tradeoff. Lasso regression and some settings of elastic net regression generate sparse models, selecting many of the weights to be zero. This means that each class prediction may be produced by a different set of base classifiers. When linear regression problem is under determined, there are many possible solutions. This can occur when the dimensionality of the meta-feature space K is larger than the effective rank of the input matrix m, where k is the number of base learners and m is the number of training samples. In this case, it is possible to use a basic solution, which has at most m- nonzero components, where m is the effective rank of the input matrix. Ridge regression augment the linear least squares problem with a $L_2 - norm$ constraint, $P_R = \sum \beta_j^2 \leq S$. Lasso regression augments the linear least squares problem with a $L_1 - norm$ constraint, $P_L = \sum |\beta| \leq t$. The $L_1 - norm$ constraint makes the optimization problem nonlinear in y_i and quadratic programming is typically used to solve the problem. Unlike ridge regression, lasso regression tends to force some model parameters to be identically zero if the constraint t is tight enough, thus resulting in sparse solutions. Elastic net regression is a convex combination of the ridge and lasso penalties. $P_{EN}(\beta, \alpha) = (1-\alpha) \times fraction12 \times \|\beta\|_L^2 2 + \alpha \times \|\beta\|_{L1}$, where $0 \leq \alpha \leq 1$ controls the amount of sparsity. Elastic net regression has the ability to perform groupwise selec-

tion when there are many correlated features, unlike lasso, which instead tends to select a single feature under the same circumstances. When there are many excellent classifiers to combine, their outputs will be highly correlated, and the elastic net will be able to perform groupwise selection.

Stacking-based solutions have been used widely and successfully in recommendations engines. **Recommendation engines** are special systems used to give recommendations for the user, e.g. what to watch next? What to buy next? What to do next? … etc. A very important example of designing recommendation engines is the Netflix competition [46, 47, 48], where the target is to design a recommendation system to recommend what to watch next. The challenges of designing recommendation engines emerge from the fact that the input of the recommendation system is a description of the user itself. The users may have different descriptions, the quality of these descriptions may vary from one user to another. For example, some users can be very active, e.g. users who watch a lot of movies vs. users who watch very few number of movies. Some users may rate the movies they see, other users they do not. Some users are interested in one type of movies, others are interested in different types. These kind of variations makes designing a single system to perform recommendation a challenging process. Many papers address the problem of recommendation engines and how variation between users can affect the solution [46, 47, 48, 67]. STREAM, Stacking Recommendation Engine with Additional Meta-features [68], Feature-weighted linear stacking **(FWLS)** [47] are among the others.

Many algorithms address the issue that there is no guarantee that stack-based classifiers outperform all base classifiers. In [69], the author proposes a new stacking algorithm to address this issue, where the predictive scores of each possible class labels are firstly collected by the meta learner and then all possible class labels are re-ranked according to these scores. They propose a new stacking algorithm that builds a meta-learner to find a linear optimum combination of base classifier on the training samples during training process. The meta-learner firstly collects the predictive sores returned by the base classifiers for each possible class label, and then re-ranks all possible class labels according to these scores.

Menahem [70] proposed a new variant of Stacking called Troika to deal with the problem of performance when using stacking with multi-class classification problem, whose main feature is that the meta-level is composed of three layers. In [70], the authors address the problem of converting the multi-class classification problem into one-against-one or one-against-all. Converting the problem to one-against-all is believed to be a problem, especially when we have data-imbalance

problem. When using one-against-one binarization, many classifiers will be testing samples that have not seen during the training process. If we have k classes, the total number of classifiers is $\frac{k\times(k-1)}{2}$. Only $(k-1)$ of the base learners have been trained on samples include the class C and $\frac{k\times(k-1)}{2} - (K-1)$ of the base learners have not seen any sample belong to class C during the training process. This means, that for a very large number of classes, the probability that a value in the meta-data is generated randomly (by using a blind classifier, a classifier that has been trained on samples do not belong to the same class of the tested sample) is high. In fact, this probability goes to one as the value k goes to infinite.

Troika [70] introduces a stack-based classifier that consists of three different layers. In the first layer, the outputs of the base classifiers are combined using an OAO ensemble, whose members are called specialist classifiers. The goal of each specialist is to predict the probability that an instance belongs to one of the two classes that it distinguishes. In the second stage, the outputs of the specialists are combined again using an OAA schema. The task of the level-2 classifiers is to learn the behavior patterns of the specialist classifiers and to predict whether the output given by a specialist is correct. The third layer contains a classifier and produces the ensemble final decision. Moreover, the authors analyze three arrangements to train the base classifiers (OAO, OAA, and all-against-all) and determine that Troika is more accurate than Stacking and StackingC in all cases. Regarding the runtime, the authors conclude that Troika outperforms Stacking and StackingC only when the base classifiers are trained using the OAO architecture.

Even though, in [70], the authors proclaim that using three layers of combining classifiers is beneficial for the performance, it is not clear how these layers help the classification process. There is no mathematical justification nor a simple justification why it works. It is not clear how they deal with the curse of dimensionality problem. The main problem that leads to lower accuracy (due to overfitting) when using stacking with multi-class problems. In fact, adding new layer between the base learners and the meta-learner does not reduce the dimensionality. Regarding the problem of one-against-all, even though the specialist classifiers deal with one-against-one problem, the next layer of classifiers deal with a one-against-all problem. In fact, they just move the problem from one layer to another.

One problem associated with stacked generalization is identifying which learning algorithm should be used to obtain the meta-classifier, and which ones should be the base classifiers. Between 2010 and 2016 many articles have been published to address this problem, most of them

are categorized under the category of optimizing stacking configuration based on meta-heuristic search techniques. Genetic Algorithm GA, Ant-Colony optimization algorithm ACO, Bee-Colony based optimization algorithm BCO, Data Envelopment analysis and PSO algorithms have been used in optimizing the configuration of the stack classifier.

Ledezma et al [71] proposed an approach that poses this problem as an optimization task. They use optimization technique based on heuristic search to solve it. In particular, they apply genetic algorithms to automatically obtain the ideal combination of learning methods for the stacking system. Their approach, called GA-Stacking, not only determines which meta-level and which (and how many) base classifiers must be present but also their learning parameters. Moreover, GA-Stacking provides flexibility and extensibility compared to previous Stacking variants because it can easily incorporate new learning algorithms and is not restricted by 'a priori' assumptions. In general, one strength of these heuristic search-based approaches is the ability to adapt the Stacking configuration to the domain biases and characteristics so that the Stacking configurations determined by GA-Stacking are domain dependent.

GA-Stacking has two major drawbacks: first, GA-stacking requires longer training time, several iterations of training/evaluation is required to find the optimal configuration. Second, using genetic algorithms to search for good Stacking configurations can lead to overfitting problem. The main reason for having the overfitting problem when using GA algorithm is that the fitness value is obtained using the same instances employed to generate the ensemble of classifier by means of Stacking.

GA-Ensemble was proposed by Ordonez et al [72] to address the first problem: how to improve the efficiency without losing accuracy? GA-Ensemble tries to determine, in a reasonable time, which classifiers and which method to combine them is the best option for a specific domain. To solve this issue, in this work the genetic algorithm makes use of a pool of trained classifiers; that is, all the algorithms needed by the genetic algorithm are trained a priori to avoid training them in successive generations of the genetic algorithm. GA-Ensemble applies a genetic algorithm in searching the configurations according to different datasets without a priori assumptions. At the beginning, a set of candidate base-level classifiers is trained to generate a pool of base-level classifiers thus to improve the efficiency without losing accuracy. The candidate set must be encoded in a chromosome, which represents a potential configuration. Binary encoding is used to accompany the canonical GA, where a 0 in the gene means that the classifier of this gene will not be used in the configuration and a 1 means the classifier will be used. The last gene in a chromosome represents two different stacking

combining schemes: multi-response model tree or majority voting. This GA search process will iterate for several generations. For each generation, the classification accuracies on validation sets are used as the fitness values to evaluate the chromosomes. Some elite chromosomes will be kept for the next generation and some poor ones will be eliminated. Mutation and crossover operations will be applied to some chromosomes to generate new chromosomes. After all generations are finished, the best chromosome will be chosen as the final configuration.

Ledezma et al [73] proposed a new approach to apply GA. To avoid overfitting, they divide the training data into two sets, training set and validation set. The training set used in the training and the validation set used in obtaining the fitness value. In [74], the authors use concepts from both ensemble learning and distributed data mining to create a modified and improved version of the standard stacking ensemble learning technique by using a genetic algorithm (GA) for creating the meta-classifier. The authors study different ways of distributing the data and use the stacking ensemble learning to used different learning algorithms on each sub-set and create a meta-classifier using a genetic algorithm. Recently, an advanced genetic approach for stacking classifier has been proposed [75]. The authors propose Advanced GA-Ensemble (AGA-E) which selects the configuration by several independent GA processes on subspaces and uses a tabu strategy to deal with the unnecessary reproduction.

Using Data Envelopment Analysis DEA to construct ensemble classifiers was proposed by [76]. Zhu proposed the DEA-Stacking approach which applies data envelopment analysis (DEA) to find optimal stacking [77]. DEA is a linear programming methodology to measure the efficiency of multiple decision-making units (DMUs) when the production process presents a structure of multiple inputs and outputs. DEA-Stacking considers the classifiers as the DMUs in DEA. In this approach, the inputs and outputs of a DMU are extracted from the confusion matrix of the model. At the first stage, the classifiers are trained and evaluated. The DEA models take the number of false positive and false negative as the inputs and the number of true positive and true negative as the outputs of the DMUs to find out the efficient one(s) to be the base classifier(s). Several classifiers with an efficiency of 1 will be selected as the base classifiers in stacking. At the second stage, the meta-classifier is also selected by the DEA models. The stacking with each learning algorithm in the set combining the selected base classifier(s) is treated as the DMUs to find the most efficient as the final configuration.

Following a similar approach to the work of using GA and posing the stacking configuration as an

optimization problem, Chen et al [78] use ant colony optimization (ACO) domain dependent stacking configurations. They use the meta-heuristic ACO to determine the level-0 Stacking classifiers with a predefined level-1 classifier as well as the entire Stacking system configuration (level-0 and level-1). Ant colony optimization have been used in data mining for optimization in several other works [79, 80, 81, 82]. ACO is a meta-heuristic algorithm which is inspired by the foraging behavior in real ant colonies.

It is worth mention the difference between GA-based stacking and ACO-based stacking techniques. Though ACO-Stacking and GA-Ensemble are all hybrids of meta-heuristics with stacking ensembles, there are some differences between them. In ACO searching process, different ants can communicate with each other by using pheromone to pass and share information, while during the GA searching process, communication is not allowed. In fact, chromosomes cannot communicate with each other. The only way to share information is by using crossover. The crossover points and the mutation points are selected randomly, so some well-performed stacking may generate poor offspring.

The searching process in GA-Ensemble is therefore more stochastic than that in ACO-Stacking. To escape from sticking in local minima, the weak ants in ACO-Stacking will not be eliminated but simply stop searching in this iteration. In GA-Ensemble, the last n cull chromosomes will be eliminated, and the top m elite chromosomes will be kept for the next generation. The mutation and crossover operations on the elite chromosomes are used to escape from local minima. However, there are no strategies to stop the same weak stacking from being generated again in the following generations, which will be expensive because these weak stacking must be evaluated again. ACO-Stacking is more flexible than GA-Ensemble in meta-classifiers selection. GA-Ensemble can only select either a multiple-response model tree or a majority voting scheme as the meta-classifiers, while ACO-Stacking can select the meta-classifiers from a set of learning algorithms. If the number of base-level classifier candidates in ACO-Stacking is the same as the number of genes representing classifier candidates in GA-Ensemble, the search space of ACO-Stacking is larger than that of GA-Ensemble. Furthermore, if the best meta-classifier for a certain dataset is neither the majority voting scheme nor a model tree, GA-Ensemble is unable to find it.

Particle swarm optimization is a heuristic global optimization method, it is based on swarm intelligence. The algorithm is widely used and rapidly developed for its easy implementation. In [83, 84], PSO is used to select and combine different classifiers to build ensembles.

Recently, Shunmugapriya and Kanmani [85] proposed the use of another meta-heuristic search algorithm to determine which and how many base classifiers to use and what meta-classifier to use based on the domain. Therefore, they have proposed to use an artificial bee colony (ABC) method. The ABC-Stacking has been proposed and implemented in two levels: ABC-Stacking1 (Base-Level Stacking) and ABC-Stacking2 (Meta-Level Stacking). In base-level stacking, the employed and onlooker bees search for the optimal configuration of base classifiers and the meta-classifier is a fixed learning algorithm. In meta-level stacking, while forming the base classifier configuration, the bees simultaneously keep exploring for the best meta-classifier through the pool of classifiers, each time the stacking of selected classifiers is done. The authors compared their results with the studies of Ledezma et al [71, 73] and Chen and Wong [78, 79, 80, 81, 82] and they conclude that the results of the Stacking configurations determined by ABC are comparable to those obtained in the previous study.

Learning from data streams is important, especially when we expect concept drift. Concept drift is the process of changing the true data generating distribution with time. Concept drift can happen as changing in the number of classes, changing in the types of classes, variation within the same class, similarities between different classes, changes in the types and numbers of features, and adding/removing classes. Concept drift is a very practical issue. Designing a learning algorithm that deals with the concept drift (concept drift detection) and changes the models adaptively to capture these drifts is an important issue. Fast adaptive stacking [86] discusses the idea of using stack base classier with data streams. The paper addresses the problem of detecting concept drift in the context of stacking based classifier, identifying bad base learners, training alternative and replacing old base learners, and re-training the meta-classifier. [87] is another published work that propose a pruned Stacking ELMs (PS-ELMs) algorithm for time series prediction (TSP).

Learning from data streams, learning from data streams with concept drift and learning from imbalanced data sets are rarely addressed in stacking-based classifier. Very few works have been published to deal with these practical issues [87].

2.3 The Data Imbalance Problem

A data set is an imbalanced one if the classification categories are not equally represented. Furthermore, the class with the lowest number of instances is usually the class of interest from the point

of view of the learning task. The class imbalance problem has been reported as one of the major causes to severely deteriorate the classification performance of many standards and well-known training algorithms. Recently, data imbalance problem has emerged as one of the main challenges in data mining community. It has attracted significant attention from researchers of different fields due to the fact that data imbalance problem is every common in real-world classification problems.

Many techniques have been suggested to address the problem of imbalance data. These techniques can be categorized into three basic categories, which depends on how they deal with the class imbalance. Algorithm level approaches modify learning algorithms to consider the data imbalance problem [88, 89, 90]. Data level approaches modify (re-balance) the data distribution by using over/under sampling to decrease the effect of the skewed class distribution in the learning process [91, 92, 7]. Cost-sensitive techniques combine algorithm and data level approaches to incorporate different misclassification costs for each class in the learning process [93, 94, 95]. These conventional class imbalance handling methods might suffer from the loss of potentially useful information, unexpected mistakes or increasing the likely hood of overfitting because they may alter the original data distribution.

Existing literature focuses mainly on binary imbalanced classification problem while multi-class imbalanced learning is barely addressed. Most of the binary imbalanced learning solutions are related to bagging and boosting ensemble strategies and up to our knowledge there is no solutions related to stacking ensemble.

2.3.1 Sampling Techniques

Data level approaches modify the data distribution by either undersampling or oversampling. Under-/Over- sampling can be performed randomly by either dropping or duplicating some of the samples in the majority/minority classes respectively. Dropping samples from the majority class may lead to loosing information due to the fact that dropped samples are not used in training. Undersampling is efficient if the database is large and the variation within the classes is very limited. The benefits achieved by duplicating the samples depends on the used base learners. For example, if the used base learner is decision tree, increasing the number of samples by simply duplicating them can lead to overfitting, especially if the original number of samples is very small. As an alternative to random undersampling (RUS) and random oversampling (ROS), synthetic oversampling

is suggested by [14]. Synthetic Minority oversampling technique SMOTE [14] is an oversampling method. The main idea of SMOTE is to create new minority class samples by interpolating several minority class instances that lie together. For each sample one or more (depends on the sampling ratio) of the k nearest neighbors (kNN) is (are) selected. The original sample and its neighbor are used to create new sample. The new instance is generated by random interpolation of both instances. $NS = OS + a \times distance(OS, KNNS)$. NS is the new sample, OS is the original sample, $KNNs$ is one of the neighbors that specified by the kNN and a is a random number between zero and one. Using synthetic oversampling causes the decision boundaries for the minority class to be spread further into the majority class space and thus prevents the overfitting problem. The success of SMOTE algorithm motivated for further research. Modified versions of synthetic minority oversampling technique were suggested in literature, Border-Line SMOTE [96], Adaptive SMOTE [97], Safe SMOTE [98], and Modified SMOTE [99] are among the others.

SMOTE has a well-known drawback. In some cases, SMOTE may suffer the occurrence of over-generalization problem. Synthesized data samples of SMOTE algorithm may spread on both majority and minority classes and thus affecting the classification performance. These problems have been addressed in literature by improving the quality of synthesized samples and avoiding spreading the data on the majority class regions. ADASYN [97] adaptively generates data samples based on their distributions and using KNN. The main difference between ADASYN and SMOTE is that ADASYN uses the distribution to decide about the number of synthetic samples to be generated for each minority sample by adaptively changing the weights of the different minority samples to compensate for the skewed distributions while SMOTE generates the same number of samples for each minority sample. Adaptive Neighbor Synthetic, (ANS) [100] dynamically changes the number of neighbors to be considered for each one of the minority samples. This main difference makes (ANS) parameter free. Borderline-SMOTE [96] generates synthetic samples along the borderline of minority and majority classes. Safe Level SMOTE [98] is define as the number of a positive instances in K nearest neighbors. If the safe level of an instance is close to 0, then the sample is considered as noise. Only samples with sufficient safe level are used to synthesize more samples. Each synthetic instance is generated in safe position by considering the safe level ratio of instances. DBSMOTE, Density Based SMOTE [101], uses DBSCAN clustering algorithm to form clusters within the minority class. Samples are synthesized along the shortest paths from each instance to a pseudo-centroid of a minority class cluster.

Other synthetic sampling techniques apply both of synthetic oversampling for minority samples and undersampling for the majority class. SMOTE-Tomek links [102] removes samples that form Tomek links from both minority and majority class. Tomek links can be defined as follows: given two instances i_a and i_b, $d(i_a, i_b)$ is the distance between the two samples. If there is no instance i_l, such that $d(i_a, i_l) \leq d(i_a, i_b)$ or $d(i_b, i_l) \leq d(i_a, i_b)$, the pair (i_a, i_b) is called a Tomek link. It implies that either these two examples are noise or are borderline samples. Tomek links have been used in undersampling.

2.3.2 Ensemble Classifiers for Data Imbalance Problem

Ensemble classifiers are designed to achieve high classification accuracy on balanced datasets, Ensemble algorithms that are applied directly to imbalanced datasets do not solve the problem implicitly exist in these datasets. Ensemble methods are augmented with sampling techniques to address the data imbalance problem. Different methods were developed based on bagging and boosting algorithms. UnderBagging [103], SMOTE-Bagging [104], Border-Line SMOTE-Bagging [105] and RBBag [106], SMOTE-Boost [107], Border-Line SMOTE-Boost [108], Rus-Boost [109] and Adaptive Rus-Boost [110] are among the others. These algorithms are considered as algorithm level approaches, in the sense that they change the ensemble algorithm but not the base learner. In addition to change the ensemble algorithm, these ensembles include a data level approaches to pre-process the data before learning each classifier.

OverBagging [6] and UnderBagging [103] are two variants of the bagging ensemble classifier to handle the data imbalance problem. Instead of using bootstrap sampling where samples are randomly sampled from the training dataset, imbalance-aware sampling scheme are used. In OverBagging, an oversampling process of the minority class is carried out to account for the data imbalance problem before training each of the base learner. In UnderBagging, undersampling is used instead of oversampling. RBBagging, Roughly Balanced Bagging [106], constructs base learners using a roughly balanced training sets instead of fully balanced datasets. The idea is by using roughly balanced datasets, the constructed base learners will be more divers. Synthetic oversampling is used also with bagging to handle data imbalance problems. SMOTEBagging [104] uses SMOTE algorithm to apply synthetic oversampling. SMOTEBagging [104] differs from the use of random oversampling not only because of the different pre-processing mechanism, but also because of the

way of creating each bag. SMOTEBagging uses different sampling rates for different base learners (iterations), and for each iteration the same number of samples are sampled from both minority and majority classes.

Depending on the base learning type, Adaboost can be really sensitive to imbalanced datasets, misunderstanding samples from minority classes as noisy or hard samples. AdaCost [8] provides a cost sensitive alternative to AdaBoost. The main idea behind AdaCost is to use different weighting scheme. AdaCost differentiates between examples of the minority and majority classes. AdaCost can put emphasis on minority class examples based on user-specified parameter. RareBoost [111] addressed the issue of having rare classes "Data imbalance", instead of considering the difference between minority and majority samples in the weighting process, RareBoost updates the weights according to how well an iteration generated hypothesis distinguishes between false positives and true positives as well as false negatives and true negatives. Like AdaCost, RareBoost suggests using a different scheme for weighting samples and give more attention to minority classes. Some other boosting based techniques uses synthetic oversampling. SMOTEBoost [107] combines data sampling with boosting. The data sampling aspect of SMOTEBoost uses the synthetic minority oversampling technique (SMOTE). AdaC1, C2 and C3 [112] are three cost-sensitive boosting algorithms were developed by introducing cost items into the learning framework of AdaBoost. For each proposed boosting algorithm, its weight update parameter is deduced taking the cost items into consideration.

Cost-sensitive represents another class of class-imbalance learning methods. Cost-sensitive based methods intentionally increase the weights of examples with higher misclassification cost in the boosting process. While some algorithms start the training process with setting higher weights for minority instances, other raise high cost examples' weights in every iteration of the boosting process, for example, AsymBoost [113], AdaCost [8], CSB [114], DataBoost [115], and AdaUBoost [116], are among the others.

Easy-Ensemble [117] combines both of bagging and boosting to deal with the data imbalance problem, since most of the previous approach may suffer from information loss, that some of the majority samples may not be used during the training process. Easy-Ensemble, apply bootstrap sampling to use all minority samples and similar number of samples from the majority class, repeatedly generating multiple classifier (base learners) by using all minority samples and subsets of the majority class. These base learners are combined using AdaBoost algorithm. Many other

algorithms use the same strategy. MultiBoosting [118] combines boosting with bagging/wagging [119] by using boosted ensembles as base learners. Stochastic Gradient Boosting [120], Cocktail Ensemble [121] and BalanceCascade [122] are among the others.

Designing ensembles that can take the advantage of reducing variance and bias error and achieving high accuracy when dealing with data imbalance problems is still a hot research topic. Many papers are published every year to address this challenge. EUSBoost [123], RB-Boost [124], Boundary-Boost [125], BPSO-Adaboost-KNN [126] and Adaptive EUSBoost [127] are among the others.

2.4 Rational Behind Stacking

Stacking based classifier represents one of the areas of meta-learning. Meta-learning studies how to combine/ choose the right bias dynamically, as opposed to base learning where the bias is fixed. In typical inductive learning scenarios, applying base learner (e.g. decision tree, SVM, neural network, NB … etc.) over some data produces a hypothesis that depends on the bias (fixed bias) embedded in the learner. Meta learning includes different areas: building a meta learning of base learners, selecting inductive bias dynamically, building meta-rules to match the task properties with algorithm performance, inductive transfer and learning to learn. In this chapter we consider only the first area of meta-learning: Building a meta-learner of base learners.

Building a meta-learner of base learners is known in the literature under the name of stacking based classifier. It is considered as one of the most common ensemble classifiers in addition to boosting [28, 29, 30, 31] and bagging [16, 17, 18]. Stacking (Stacked generalization) was initially proposed by Wolbert in 1992 [49]. In his paper, he introduced the idea of stacking multiple classifiers together and use the outputs of these classifiers to train / test another classifier. Our study is centered on the classification problem exclusively. The problem is to learn how to assign the correct class (label) to each of a set of different objects.

A learning algorithm is first trained on a set of pre-classified (labeled) examples $T_{train} = (\overline{X_i}, Y_i)_{i=1}^{m}$. Each example $\overline{X_i}$ is characterized by features and is represented as a vector in an n-dimensional feature space, $\overline{X_i} = (X_1, X_2, X_3, ..., X_n)$. Each $\overline{X_k}$ can take on a different number of values. $\overline{X_i}$ is labeled with class Y_i according to unknown target function, $F(\overline{X_i}) = Y_i$. In classification, each Y_i takes one value over a set of fixed categorical values. T_{train} consists $i.i.d$ examples

obtained according to a fixed but unknown joint probability distribution ϕ in the space of possible feature vector X. The goal of the learning algorithm L is to produce a hypothesis h that best approximates F. Given a training set T_{train}, the learning algorithm L will search over a hypothesis space H_L until it finds a hypothesis $h, h \in H_L$ that approximates the function F. The number of hypothesis $\in$ hypothesis space H_L is $|H_L|$.

$|H_L|$ depends on the biased embedded in the learning algorithm L. If the biased is strong, then the number of hypothesis is small. If the bias used with learning algorithm L_A is stronger than the bias used with learning algorithm L_B then ($|H_{LA}|$ < $|H_{LB}|$). In this case, the bias embedded by L_A conveys more extra information than bias in L_B. For example, if the learning algorithm L_A is used to estimate the function F by finding the parameters a and c in the hypothesis function $h_A = ax_0 + c$.The bias is that the function F is linear. If the learning algorithm L_B used to estimate the function F by finding the parameters a,b and c in the hypothesis function $h_B = ax_0 + bx_1 + c$. The bias of L_A is stronger than the bias of L_B and the $|H_{LA}|$ < $|H_{LB}|$. In fact, for this specific case, $H_{LA} \in H_{LB}$.

A task can be defined by a set of training examples T_{train}, where each example is labeled with $Y_i \in Y_1, Y_2, Y_3, ..., Y_c$, where c is the total number of classes. Four parameters are used to define a task: F, the function used to assign labels to examples. Our ultimate goal is to come up with a hypothesis h that can approximate the function F. ϕ, the distribution used to generate examples. m, the number of training samples. $p(T_i)$ is the probability of generating the training set T_i according to ϕ.

The space of all tasks contains many random regions. For this reason, we will assume R_L compromises a subset of structured tasks $S_{struct} \subset S$, where each one of these tasks is a non-random task. A random task can be defined as a task that contains m examples, where each example is defined as $(\overline{X_i} = (X_1, X_2, X_3, ..., X_n), Y_i \in Y_1, Y_2, ..., Y_c)_{i=1}^{m}$ and is generated by using random functions. A random function to generate the numbers $(X_1, X_2, X_3, ..., X_n)$, m sequences have been generated, and a random function is used to generate the labels $(Y_1, Y_2, ..., Y_c)$. It is very difficult to have a learning algorithm L that is defined over the hypothesis sub-space H_L, where a hypothesis h can be found to approximate F if it is random. One dimension along which we can differentiate between structured and random tasks lies in the expected amount of data compression that can be obtained over the training set. Structured tasks usually denote regular patterns over the training sets that commonly lead to the discovery of concise representations. Random tasks on the

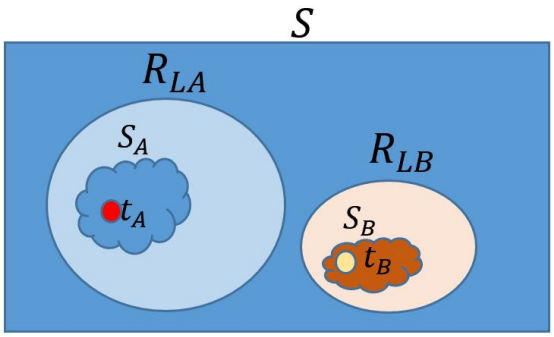

Figure 5: Example of structured tasks and the region of all tasks that can be solved by using two different learning algorithms.

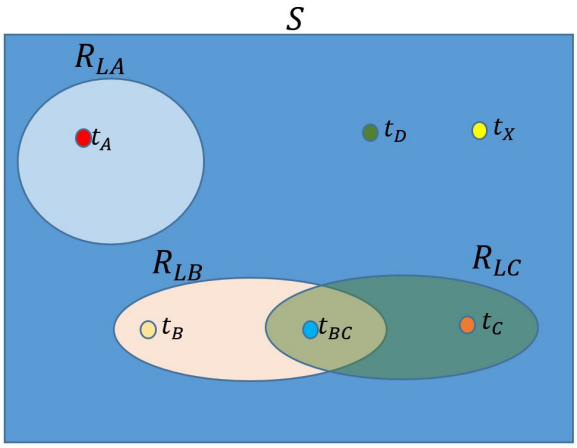

Figure 6: Examples of three different regions and multiple tasks.

other hand are characterized by many irregularities. In general, the assumption always is that the task T is not a random one. Now the question is: can we find a hypothesis h that best approximates the function F?

Figure 6 shows the space of all tasks S. In this space, we define the region R_{LA}, where all tasks $S_A \subset R_{LA}$ can be solved by using the learning algorithm L_A The structured task t_A is one of the tasks that can be solved by the learning algorithm L_A. In the other hand, the region R_{LB}, can be defined in a similar way. t_B is one of the tasks that can be solved by L_B. The biased used in L_B is stronger than the biased used in L_A. The expected number of hypothesis that can be learned by the learning algorithm L_A is higher than the number of hypothesis that can be learned by using the learning algorithm L_B.

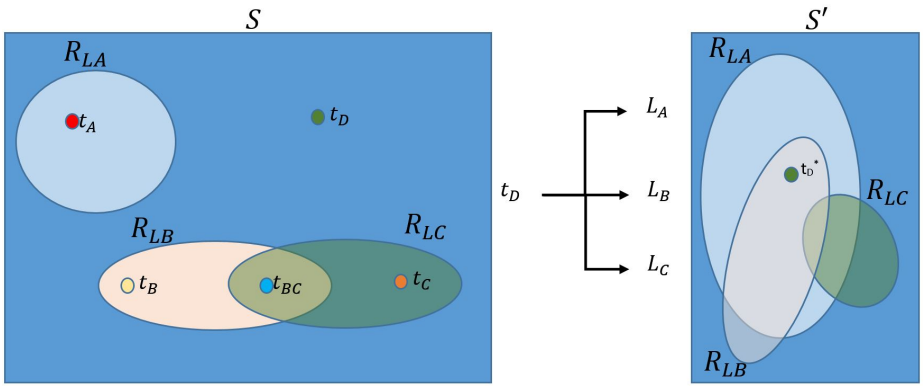

Figure 7: Task transformation by using different learning algorithms. The task t_D cannot be solved by any of the learning algorithms in the original domain (the input domain), while the transformed version (t_D^*) can be solved by using the learning algorithm L_B.

Figure 6 shows examples of three different regions that are represented by three different learning algorithms L_A, L_B, and L_C. The following notes are made: Tasks t_A, t_B and t_C can only be solved by using the learning algorithms L_A, L_B and L_C respectively. Task t_{BC} can be solved by either learning algorithm L_B or L_C. Task t_D cannot be solved by any of the available learning algorithms. In practice, given the task t_D and the set of learning algorithms L_A, L_B and L_C. Finding the most suitable learning algorithm to use with the task t_D can be one of the meta-learning objectives. By mapping tasks and learning algorithms, the bias embedded in a learning algorithm can be changed. Changing the bias embedded in the learning algorithm L_A, such that we either shift or enlarge the region R_{LA} to cover the task t_D, is one of the meta-learning objectives. In staking based classifier, given a limited number of learning algorithms. These algorithms come with fixed biases. Dealing with the task t_D is done by transformation, by transforming the problem from the input domain (space) to another domain (space) where one of the available learning algorithms can be used to solve the task t_D^*. 7 shows the process of transforming the tasks from the input space S to the output space $S^{'}$.

In the original space S, none of the available algorithms was able to solve the task t_D. Using the available learning algorithms, we can transform the task t_D into another domain (Space), where the learning algorithm L_B can be used to solve it (the rational behind stacking). The last example is a description of an area of meta-learning called meat-learning of base learners. Stacked Generalization is considered a form of meta-learning, because the transformation of the training set conveys information about the predictions of the base learners. In stacked generalization, both

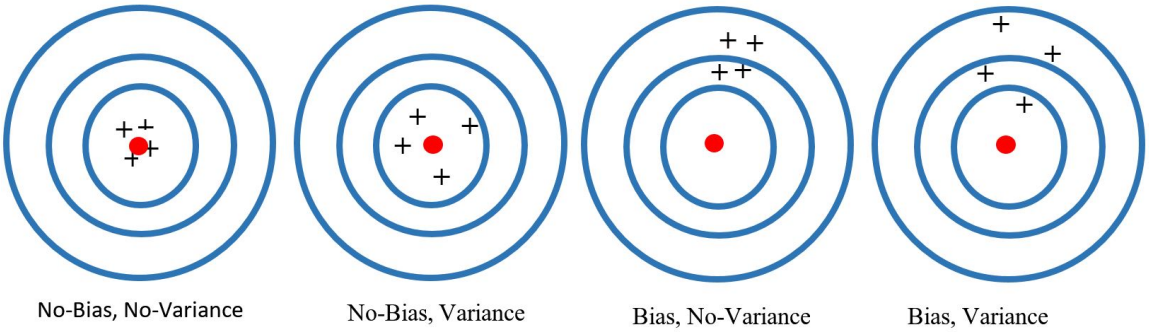

Figure 8: Bias and variance errors. Training different models using the same learning algorithm, but different training set. The red dot is the target. The black +s are the corresponding predictions of these models.

base learners and meta-learner have a fixed form of bias. The dominant task region (e.g. R_{LA}) for the meta-learner may be different from the base learner but ultimately fixed.

2.5 Rational Behind Bagging

Explaining rational behind learning algorithms should be directly related to its generalization error. The simplest way to understand generalization error is to understand its main components. Prediction errors are decomposed into two main sub-components, Bias and Variance errors. Figure 8 depicts bias and variance errors. A classifier (regressor) is trained using different training sets, each time an example $\overline{X_i}$ is tested and the output is reported. In Figure 8, the red $\bullet$ is the target output. The black $+s$ are the predictions of different models when training is performed by using different training sets. There is a tradeoff between a model's ability to minimize bias and variance errors.

Error due to bias: The error due to bias is taken as the difference between the expected (or average predictions) of our models and the correct value which we are trying to predict. Error due to variance: The error due to variance is taken as the variability of a model prediction for a given data point. The variance is how much the prediction of a given point vary between different realizations of the model when it is trained using different data sets. Variance error does not depend on the ground truth.

Given the sample $\overline{X_i}$, the data set D and the learning algorithm L. A model M is generated by applying the learning algorithm L on the data set D, $M_j = L(D_j)$. The prediction $Y_{ij} = M_j(\overline{X_i})$, is

the prediction of the model M_j. Y_i is the ground truth label (the expected label) of the sample $\overline{X_i}$ over the true data generating distribution. $\overline{Y_i}$ is the average prediction of the model using different training sets. Due to the randomness in the underlying data sets, generated models will have a range of predictions. Bias measures how far off "in general" the model's predictions are from the correct value. Bias and variance errors are given by:

$$\overline{Y_i} = \frac{1}{N}\sum_{j=1}^{N} Model_j(\overline{X_i}) \tag{2.3}$$

$$BiasError = Y_i - \overline{Y_i} \tag{2.4}$$

$$VarianceError = \frac{1}{N}\sum_{j=1}^{N}[Model_j(\overline{X_i}) - \overline{Y_i}]^2 \tag{2.5}$$

Theoretically, given a model with enough number of parameters, such that the number of structured tasks that can be learned by using this model is unlimited, and infinite number of data samples, both bias and variance errors can be reduced. Practically, since available number of data samples is limited, there is a tradeoff between bias and variance. In general, reducing bias is more important than reducing variance for a given model. Even if the variance error of a model is not zero, variance error can be reduced by using averaging (the rational behind bagging). Asymptotic consistency guarantees that bias error converges to zero as the number of data samples converges to infinite and asymptotic efficiency guarantees that the resulted model is no worse than any other model you could use when infinite data samples are available. Overfitting and underfitting are two major problems related to variance and bias errors, respectively.

Chapter 3

BAGSTACK CLASSIFIER FOR DATA IMBALANCE PROBLEMS

Ensemble learning algorithms construct a set of base learners (weak learners). These base learners are constructed by either changing the set of training samples that is used to train each one of them (e.g. Bagging, Boosting) or changing the training algorithm (e.g., Stacking). In Chapter 2, differences between these ensemble methods are explained. This chapter presents the proposed classifier "BagStack Classifier for Data Imbalance Problem". The selection problem (selection and mixing perspectives) is defined in Section 3.1. In Section 3.2, the BagStack classifier is covered. Section 3.3 presents the main contributions to deal with the data imbalance problem. Section 3.4 presents experimental results to justify and prove the performance of the BagStack classifier. Finally, Section 3.5 concludes the chapter.

3.1 The Selection Problem

Based on the No-Free lunch theorem, no single machine learning algorithm can outperform all others in all different tasks.Thus, selecting the right algorithm for a specific task is important. The selection problem is the problem of selecting a function (estimator, predictor) among a class of candidate functions of a common parameter of interest.

3.1.1 Definitions

Let $S = S_1, S_2, ..., S_n$ be a set of observations (data samples) of $S \sim P_0$, P_0 is known to be the true data generating distribution. Each one of these samples are defined by the values $S_i = (\overline{X_i}, Y_i)$, where $\overline{X_i}$ is a feature vector and Y_i is the label assigned to the data sample. $\overline{X_i} \in \chi$ and $Y_i \in \varphi$. The points $S_1, S_2, S_3, ..S_n$ define an empirical distribution P_n. Let $\beta_0(.) = \beta(.\backslash P_0)$ be a function of P_0 of interest. β can be an estimator or predictor. If P_n is an empirical distribution such that $P_n \in P_0$, then define the parameter set B as: $\beta(.\backslash P_n) : P \in P_0$. P_n is the empirical distribution based on n samples that are sampled from the distribution P_0. For different set of samples, different β_s can be defined, such that all these $\beta_s \in B$. Let $L()$ be a loss function, where $L()$ is a function of the sample

set S and the function β. Then the loss function is defined as $L(S, \beta)$. An example of a loss function can be $L(S, \beta) = (Y - \beta(X))^2$. Given n samples, it is very common practice to use $\rho\%$ for testing and $(1 - \rho)\%$ for training. Thus, the function β can be written as $\beta = \beta(.\backslash P_{(1-\rho)n})$. where $(1 - \rho)n$ is the total number of samples used for training and $P_{(1-\rho)n}$ is the empirical distribution used for training. Let us define the loss function $L(S, \beta)\ in\ R,\ \beta_0$ is selected by minimizing the loss function over the true data generating distribution:

$$\beta_0 = argmin_{\beta \in B} \int L(., \beta)\ dP_0(.) \tag{3.1}$$

$$\beta_0 = argmin_{\beta \in B}\ E_0\ L(., \beta) \tag{3.2}$$

where β_0 is the function that minimizes the loss function over the true data generating distribution P_0.

Learning algorithm is used to generate the function β. Based on the given learning algorithm, a set of parameters are defined; these parameters control the behavior of the function set β_a, where $\beta_a = \beta_n : T_a(P_n), P_n \in P_0, T_a$ is a given training algorithm and P_n is an empirical distribution.

Let P_n be the empirical distribution of $S_1, S_2, S_3, ..., S_n$ and let $\widehat{\boldsymbol{\beta_k}} = \beta_k(.\backslash P_n) \in B, K = 1, 2, 3, ..., K(n)$ be a collection of functions (predictors, estimators). One can use different learning algorithms to generate this set of functions. One can think about these functions as estimators of the function β_0. The risk (performance) of a function $\beta \in B$, wherein the risk indicates the quality of the function β, is given by:

$$\widetilde{R}(\beta \backslash P_0) = \int L(o, \beta)\ d\boldsymbol{P_0(o)} \tag{3.3}$$

Based on the risk definition, we know that β_0 is the minimum value of all risks defined over the set $\beta \in B$:

$$\beta_0 = argmin_{\beta \in B} \int L(o, \beta)\ d\boldsymbol{P_0(o)} \tag{3.4}$$

$$\beta_0 = argmin_{\beta \in B}\ \widetilde{R}(\beta \backslash \boldsymbol{P_0}) \tag{3.5}$$

where the risk value $\widetilde{R}$ is defined over the true data generating distribution. The function β is generated based on P_n; it is an empirical distribution. P_n is defined over the data samples $S_1, S_2, ..., S_n$ and $KLD(P_n, P_0)$ goes to zero as n goes to infinite.

The risks of a specific function $\widehat{\beta}_k$ over the true data generating distribution and an empirical data distribution are given by:

$$\widetilde{\boldsymbol{R}}_{nk} = \int L(o, \widehat{\boldsymbol{\beta}}_k(.\backslash P_n))\ d\boldsymbol{P_0}(o) \tag{3.6}$$

$$\widehat{\boldsymbol{R}}_{nk} = \int L(o, \widehat{\boldsymbol{\beta}}_k(.\backslash P_n))\ d\boldsymbol{P_n}(o) \tag{3.7}$$

where $\widetilde{\boldsymbol{R}}_{nk}$ is the risk of the function $\widehat{\boldsymbol{\beta}}_k(.\backslash P_n)$, which is trained using the empirical data distribution P_n, over the true data generating distribution P_0. $\widehat{\boldsymbol{R}}_{nk}$ is the risk of the function $\widehat{\boldsymbol{\beta}}_k(.\backslash P_n)$ over the empirical data distribution P_n.

3.1.2 Selection Problem

The performance (risk) of a particular learner depends on how effective its searching strategy is in approximating the optimal predictor defined by the true data generating distribution. Thus, the relative performance of various learners depends on the true data generating distribution. In practice, it is generally impossible to know a priori which learner will perform best for a given prediction problem and data set. In order to define the selection problem, one should define a distance between a function $\widehat{\boldsymbol{\beta_k}}$ and the optimal function β_0 defined over the true data generating distribution P_0. The distance function is given by:

$$d(\widehat{\beta}_k, \beta_0) = \int L(o, \widehat{\beta}_k(.\backslash P_n)) - L(o, \beta_0)\ dP_0(o) \tag{3.8}$$

where $\widehat{\beta}_k$ a function trained using the empirical distribution P_n. β_0 is the optimal predictor and P_0 is the true data generating distribution.

Given a set of β functions **B**, the optimal selector is defined as follows:

$$\widetilde{\beta}_{pn} = argmin_{k=1,2,3,\ldots,k(n)} d_n(\widehat{\beta}_k, \beta_0) \tag{3.9}$$

$$\widetilde{\beta}_{pn} = argmin_{k=1,2,3,\ldots,k(n)} \int L(o, \widehat{\boldsymbol{\beta}}_{\boldsymbol{k}}(\boldsymbol{.\backslash p_n})) - L(o, \boldsymbol{\beta_0})\ d\boldsymbol{P_0} \tag{3.10}$$

Since the term $L(o, \boldsymbol{\beta_0})$ is constant, the minimization problem becomes:

$$\widetilde{\beta}_{pn} = argmin_{k=1,2,3,\ldots,k(n)} \int L(o, \widehat{\boldsymbol{\beta}}_{\boldsymbol{k}}(\boldsymbol{.\backslash p_n}))\ d\boldsymbol{P_0} \tag{3.11}$$

$$\widetilde{\beta}_{pn} = argmin_{k=1,2,3,\ldots,k(n)} \widetilde{R}_{nk} \tag{3.12}$$

where $\widetilde{\beta}_{pn}$ is the optimal selector, which chooses for each data set $S_1, S_2, ..., S_n$, (P_n) the optimal function with the smallest distance from the optimal β_0, or equivalently, with minimal conditional risk $\widetilde{R}_{nk}$.

The optimal benchmark selector always selects the best learner function from a set of functions B. Each one of these learners is trained using the empirical distribution P_n. In the case of the **optimal selector**, evaluating and selecting the best learner is based on the true data generating distribution P_0. In practice, we do not have access to the true data generating distribution. It is unknown distribution, what we have instead is the empirical distribution.

The **data adaptive selector** selects the function $\widehat{\beta}_{pn}$ based on the empirical distribution P_n:

$$\widehat{\beta}_{pn} = argmin_{k=1,2,3,...,k(n)} \int L(o, \widehat{\beta}_k(.\backslash P_n))\, dP_n \tag{3.13}$$

$$\widehat{\beta}_{pn} = argmin_{k=1,2,3,...,k(n)}\, \widehat{R}_{nk} \tag{3.14}$$

The data adaptive selector is asymptotically equivalent with the optimal benchmark selector:

$$\frac{d_n(\widehat{\beta}_{pn}, \beta_0)}{d_n(\widetilde{\beta}_{pn}, \beta_0)} \rightarrow 1 \; in \; probability \; as \; n \; \rightarrow \; \infty \tag{3.15}$$

3.1.3 Mixing Problem

In the last subsection, we defined the selection problem. In some cases, combining learners using various methods results in better performance over a single candidate learner. While selecting a function from a set of functions based on risk estimation is quit direct process, mixing learners necessitates finding and understanding relationships between different available functions, weaknesses, strengths and how they can compensate for each other. Since we have only access to the empirical distribution P_n, finding the right way to mix different learners is a challenging process.

There is a probability that mixing learners based on empirical distributions may lead to overfitting problem; β is perfectly fitting the empirical distribution P_n but is badly fitting the data generating distribution P_0 (Overfitting Problem). Many researchers suggested different methods to avoid overfitting, e.g., cross-validation.

We prefer to discuss the mixing problem within the context of cross-validation based optimization. In the next subsection, we will cover the main types of cross-validation methods, and later we will discuss in more details the cross-validation based mixing problem.

In this thesis, we are interested more in this latter category. Later we will focus more on the selection problem in the context of mixing learners instead of selecting the best one. Mixing learners can be very adaptive and flexible to absorb the selection problem under the more general mixing problem.

3.1.4 Generalized Cross-validation

cross-validation is used in model selection. Define a random vector $C_n \in \{\ 0,\ 1\ \}^n$ for splitting the learning set into validation and training, where:

$$C_{n,i} = \begin{cases} 0,\ if\ the\ i^{th}\ observation\ is\ in\ the\ training\ set \\ 1,\ if\ the\ i^{th}\ observation\ is\ in\ the\ validation\ set \end{cases} \tag{3.16}$$

The different types of cross-validation include: V-fold cross-validation, leave one out cross-validation LOOCV, Monte-Carlo cross-validation… etc. For example with V-fold cross-validation, we have V-Possible outcomes of C_n, each having equal probability.

Monte Carlo Cross-Validation: In Monte-Carlo cross-validation, the learning set is randomly and repeatedly divided into two sets. A training set of $n_0 = n(1-p)$ observations and a validation set of $n_1 = np$ observations. A common choice of p is 10%.

V-Fold Cross-Validation: In V-fold cross-validation, the learning set $S_1, S_2, ..., S_n$ is randomly divided into V-fold (sets) that are mutually exclusive and exhaustive. V-sets, $L_v, v = 1, 2, 3, ..., V$ of as nearly equal size. Each estimator is built on a training set of size $L - L_v$ and tested on L_v. $L_{vi} \cap L_{vj} = \phi$ for each $i \neq j$ and $\cup_{i=1}^{v} L_{vi} = L = \{S_1, S_2, S_3, ..., S_n\}$. C_n in V-fold cross-validation places mass $\frac{1}{V}$ on each of the v-binary vectors. $C_n^v, v = 1, 2, 3, .., V$ is defined as follows. Let $n_v = \lfloor \frac{n}{V} \rfloor$, then for each $v = 1, 2, 3, ..., V-1$, $C_{ni}^V = 1$ for $i = 1 + (v-1)nv,\ ...\ , Vn_v$ and 0 otherwise. For $v = V, C_{ni}^V = 1$ for $i = 1 + (v-1)nv, ..., n$ and 0 otherwise.

Leave One Out Cross-Validation: In Leave One Out Cross-Validation, one of the observations is used for validation and the rest for training. For n-observations, this is equivalent to n-fold cross validation with $p = \frac{1}{n}$. Two important notes about the LOOCV: (1) the number of samples used for validation is not a function of n, $\sum_{i=1}^{n} C_i = 1, n \to \infty$, $\sum_{i=1}^{n} C_i = p \times n = \frac{1}{n} \times n = 1$. (2) It is very clear that there is a bias-variance tradeoff in the selection of p. Large p_s (more validation and less training) produces estimators with larger bias and smaller variance. In the opposite side LOOCV, with small $p = 1$, leads to low bias and high variance.

Bootstrap Cross-Validation: Given a set of learning samples of size n, randomly sample from the data set with replacement. Each time you sample the probability that the observation S_i is not selected $1 - \frac{1}{n}$, where n is the total number of samples. If you sample n-times, then the probability that the sample will not be selected is $(1 - \frac{1}{n})^n = e^{-1} = 0.368$. This means that even if we sample n times, then $\sim$ third of samples are left out.

In Bootstrap Cross-Validation, the definition of the random vector C_n should be modified, in this case the random value (probability) should capture the case when the same value selected multiple times. A simple definition $C_n = 0, 1, 2, ..., n$ where the value indicates how many times the sample has been selected. Thus, if N samples are selected (sampled) then: $P = \sum_{i=1}^{N} \frac{I(C_{ni}=0)}{N}$, where P is the validation percentage. The expected value of P is $(1 - \frac{1}{n})^N$ and if $N = n \rightarrow P = 0.368$.

Cross-validation represents important tool to deal with the selection problem. In practice, using available samples in an efficient way is very important. This efficiency becomes even more important when the total number of samples available for learning (n) is very limited. In the next subsection we will re-define the selection problem (selection and mixing) based on cross-validation.

3.1.5 V-fold Cross-validation Based Selector

cross-validation based selector represents an extension to the general selection problem. Instead of using the same empirical distribution for training and testing, Equations 3.13 and 3.14, which increases the risk of overfitting the training data, the learning set is divided into training and validation. Training the functions is done using the training set and evaluating the functions (estimating the risk) is done using the validation set. Let us define the two empirical distributions: $\boldsymbol{P_{n,C_n}}^0$ is the training empirical distribution and $\boldsymbol{P_{n,C_n}}^1$ is the validation empirical distribution. The selection problem is defined as selecting the function that minimizes the expected risk over all different data splits (V-fold):

$$\widehat{K} = argmin_k E_{cn} \int L(o, \beta_k(.\backslash P^0_{n,C_n}))\, dP^1_{n,C_n} \tag{3.17}$$

Search within a set of K estimators to find the one that minimizes the expected risk over all distributions C_n. The function $\beta_k(.\backslash P^0_{n,C_n})$ is trained by using the training empirical distribution. Risk estimation is done over the validation empirical distribution. Since we are working with discreet

samples (observations), then:

$$\widehat{K} = argmin_k E_{cn}(\frac{1}{np}) \sum_{i=1}^{n} I(C_{n,i} = 1)\ L(S_i, \boldsymbol{\beta_k}(.\backslash \boldsymbol{P^0_{n,C_n}})) \tag{3.18}$$

where np is the number of samples in the validation set. Given V-distributions, V-fold cross-validation, then:

$$\widehat{K} = argmin_k (\frac{1}{V}) \sum_{v=1}^{V} [(\frac{1}{np})\ \sum_{i=1}^{n} [I(C_{n,i,v})\ L(S_i, \boldsymbol{\beta_k}(.\backslash \boldsymbol{P^0_{n,C_n,v}}))]] \tag{3.19}$$

One of the basic questions in machine learning is how to know if **cross-validation based selector** is asymptotically equivalent to the **benchmark selector (the optimal selector)**. In other words, how to validate this selector?

Let us define two cross-validation based conditional risk values:

$$\widehat{R}_{n(1-p)}(k) = E_{cn} \int L(o, \beta_k(.\backslash \boldsymbol{P^0_{n,c_n}}))\ \boldsymbol{dP^1_{n,c_n}}(o) \tag{3.20}$$

$$\widetilde{R}_{n(1-p)}(k) = E_{cn} \int L(o, \beta_k(.\backslash \boldsymbol{P^0_{n,c_n}}))\ \boldsymbol{dP_0} \tag{3.21}$$

where P^0_{n,c_n} is the empirical training distribution and P^1_{n,c_n} is the empirical validation distribution.

Based on these two risk definitions, we can define the data adaptive cross-validation based selector $\widehat{K}$ and the cross-validation based benchmark selector $\widetilde{K}$ to be:

$$\widehat{K} = argmin_k\ \widehat{R}_{n(1-p)}(k) \tag{3.22}$$

$$\widetilde{K} = argmin_k\ \widetilde{R}_{n(1-p)}(k) \tag{3.23}$$

The optimal risk R_{opt} is the minimum risk value can be achieved by using β_0:

$$R_{opt} = E_0\ [L(o, \beta_0)] \tag{3.24}$$

To prove that the **cross-validation based data adaptive selector** is asymptotically equivalent to **cross-validation benchmark selector** we need to show that:

$$\frac{d(\widehat{\beta}_{\widehat{K}}, \beta_0)}{d(\widetilde{\beta}_{\widetilde{K}}, \beta_0)} = \frac{\widetilde{R}_{n(1-p)}(\widehat{K}) - R_{opt}}{\widetilde{R}_{n(1-p)}(\widetilde{K}) - R_{opt}} \rightarrow\ 1\ as\ n\ \rightarrow\ \infty \tag{3.25}$$

This theorem has been proven by Van der Laan and Dudiot in their paper of title “Unified Cross-Validation Methodology for Selection among Estimators and a General Cross-Validated Adaptive Epsilon-Net Estimator” [128]. The real cost of using cross validation is re-training and re-validating multiple functions (that use different learning algorithms or different representations of the same data) V times.

Each time we train using $(V-1) \times \lfloor \frac{n}{V} \rfloor$ samples and test using $\lfloor \frac{n}{V} \rfloor$ samples. Recently, other methods have attracted more attention due to the excess of data in some applications (e.g., some of the computer vision applications) and the high cost of training and testing models (e.g., DNN based models). Instead of dividing the data into V-folds, the data is divided into training, validation and testing. The training data used in training, the validation data used in hyper parameters selection and evaluating the progress while performing the training process. The testing set is used only to test the final model. A very common division is to have 20% testing, 20% validation and 60% training. This method is powerful when the number of available observations is large enough to assume $P_0 = P_{training} = P_{validation} = P_{testing}$ Unfortunately, in many applications, available number of observations may not be large enough to sacrifice a validation set.

3.1.6 V-fold Cross-validation Based Mixer

The performance of a particular learner depends on how effective its searching algorithm (strategy) is in approximating the optimal predictor defined by the true data generating distribution. In practice, the relative performance of various learners depends on the true data generating distribution. In practice, it is generally impossible to know a priori which learner will perform best for a given prediction problem and data set. To solve this problem, some researchers have proposed combining learners in various methods and have exhibited better performance over a single candidate learner. When mixing multiple learners, since the system becomes more complex with more parameters and higher flexibility, the risk of overfitting becomes higher. In 2003, van der Laan suggested a solution to this problem, the super learner [50, 128, 129]. In the context of prediction, this super learner is itself a prediction algorithm, which applies a set of candidate learners to the observed data and chooses the optimal learner based on cross-validation based risk estimation.

The cross-validation selector selects the learner with the best performance on the validation sets. In V-fold cross-validation, the learning set is divided into V folds, mutually exclusive and exhaustive sets of as nearly equal size as possible. Each set and its complement play the role of training and validation. Given V-splits of the learning sample into a training and corresponding validation sample, for each of the V-splits, the estimator is applied to the training and its risk is estimated with the corresponding validation set. For each learner, the V-risks over the V-validation sets are applied

resulting in the so called cross-validation risks. The learner with the minimal cross-validated risk is selected.

It is helpful to consider each learner as an algorithm applied to empirical distributions. Thus, if we index a particular learner with an index K then: $\beta_k = \widehat{\beta}_k(P_n)$ is a mapping $P_n \rightarrow \widehat{\beta}_k(P_n)$ is the mapping from empirical distribution P_n to functions of the covariates. Consider a collection of $K(n)$ learners, $\widehat{\beta}_k, k = 1, 2, 3, ..., k(n)$ in parameter space B. The super learner is a new learner defined as: $\widehat{\beta}(P_n) = \widehat{\beta}_{\widehat{K}(n)}(P_n)$ where $\widehat{K}(n)$ denotes the cross-validation selection described above, which simply selects the learner which perform best in terms of cross-validation risk:

$$\widehat{K}(P_n) = argmin_k \; E_{cn} \sum_{i, C_n(i)=1}^{n} (Y_i - \widehat{\beta}_k(P^0_{n,C_n})(X_i))^2 \tag{3.26}$$

$$\widehat{K}(P_n) = argmin_k \; \frac{1}{V} \sum_{v=1}^{V} \sum_{C_n(i)=1}^{n} (Y_i - \widehat{\beta}_k(P^0_{n,C_n})X_i)^2 \tag{3.27}$$

As defined by Van der Laan [130], the oracle selector is the estimator, among $K(n)$ learners considered, which minimizes risk under the true data generating distribution P_0. In other words, the oracle selector is the best possible estimator given the set of candidate learners considered. It depends on the true data generating distribution P_0 and thus it is unknown.

Loss Functions: In mathematical optimization and machine learning, a loss function or cost function is a function that maps an event or values of one or more variables into a real number intuitively representing some cost associated with the event. An optimization problem seeks to reduce / minimize a loss function. An objective function is either a loss function or its negative (something called a reward function), in this case it is to be maximized. In statistics, typically a loss function is used for parameter estimation, and the event is some function of the difference between estimated and true data values for an instance of data. Given a sample $S_i = (X_i, Y_i)$, where X_i is a data vector $X_1, X_2, ..., X_d$ and $Y_i \in R$. Then, based on a set of observations $S_1, S_2, ..., S_n$ a learner (estimator) is generated β where $\beta = E[Y \backslash X]$. The loss function $L(S_i, \beta)$ is the loss function defined over the observations and the learner. One of the common loss functions is the squared error loss function, where $L(S_i, \beta) = [Y_i - \beta(X_i)]^2$.

In general, a training, regression or optimization process is a searching process to find the parameter set of β, such that $\beta \in B$, where we minimize the expected loss over the true data generating distribution, then:

$$o = argmin_\beta \; E_o[]L(S_i, \beta)] \tag{3.28}$$

This includes changing the parameter set β and minimizing the expected loss:

$$L = argmin_{\beta} \sum_{i=1}^{\infty} L(S_i, \beta) \tag{3.29}$$

$$L = argmin_{\beta} \int L(S_i, \beta)\, dP_0 \tag{3.30}$$

We are given only a set of samples $S_1, S_2, ..., S_n$, thus:

$$L = argmin_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(S_i, \beta) \tag{3.31}$$

Minimizing the loss function is an optimization problem in which we search for the best set of parameters that can reduce the average loss over the training data samples. The final loss value can be a function of: (1) the number of parameters we are trying to optimize; the number of parameters indicates the complexity of the learner β; (2) The number of training samples; with a smaller number of training samples, it is easier to find a function β that fits all training data. Using the wrong complexity level that does not fit with the available number of training samples can result in overfitting problem. Usually, dealing with overfitting problem is done by adding more terms to the loss function, these terms can control: (1) the number of parameters used in the model β; (2) the values of these parameters. Thus, the loss function can be written as:

$$L(S, \beta) = [Y - \beta(X)]^2 + \alpha F(\beta) \tag{3.32}$$

where α is the Lagrange multiplier, $F(\beta)$ a function indicates the complexity of the model or how much the parameters follow a predefined (desired) characteristics.

Naturally, any model is highly optimized for the data it was trained on. The expected error the model exhibits on new data will always be higher than that it exhibits on the training data.

$$Prediction\ Error\ =\ Training\ Error\ +\ Training\ Optimism\ Error \tag{3.33}$$

Here, the training optimism is basically a measure of how much worse our model does on new data compared to the training data. The more optimistic we are, the better our training error will be compared to what the true error is and the worse our training error will as an approximation of the true error. The optimism is a function of the model complexity, as complexity increase so does optimism:

$$True\ Prediction\ Error\ =\ Training\ Error\ +\ F(Model\ Complexity) \tag{3.34}$$

Loss-Based General Mixer: Let $\widehat{\beta}_j, j = 1, 2, 3, ..., J$ be a collection of J candidate learners, which represent mapping from the empirical distribution P_n into the parameter space B consisting of functions of X. The proposed classifier is a loss-based mixer that uses cross-validation (two levels of cross-validation, V-fold and bootstrap cross-validation). Let $v \in 1, 2, 3, ..., V$ index a sample split into validation sample $V(v) \in 1, 2, 3, 4, ..., n$ and training sample $T(v) \in 1, 2, 3, 4, ..., n$ where $T(v)$ is the complement of $V(v)$.

$$V(v) \cup T(v) = 1, 2, 3, ..., n \tag{3.35}$$

$$\cup_{v=1}^{V} V(v) \ = \ 1, 2, 3, ..., n \tag{3.36}$$

$$\cap_{v=1}^{V} V(v) = \phi, V(v_i) \cap V(v_j) \ = \ \phi \ for \ any \ i \neq j \tag{3.37}$$

For any validation set $V(v)$ and its complement $T(v)$, let the learner:

$$\beta_{njv} = \widehat{\beta}_j(P_{nT(v)}) \tag{3.38}$$

where n is the total number of samples, j is the index of the learner, v is the split index and $P_{nT(v)}$ is the empirical distribution of the training data set $T(v)$.

If we manipulate the training data $T(v)$ to extract another training set $T^{'}(v)$ by either synthesizing samples, oversampling, undersampling, splitting/combining classes (categories), then, $P^{'}_{nT(v)}$ is the empirical distribution of the new training set and $\beta^{'}_{njv} = \widehat{\beta}_j(P^{'}_{nT^{'}(v)})$. For now, to keep things simple, let us consider only the probability distribution $P_{nT(v)}$.

For an observation i, let $V(i)$ denotes the validation sample it belongs to, then for $i = 1, 2, 3, ..., n$, we construct a new data set of n-observations as follows: (Y_i, Z_i), where $Z = (\beta_{njv(i)}(X_i), j = 1, 2, 3, 4, ..., J)$. Z is the vector consisting of the J predicted values according to the J estimators trained on the training sample $P_{nT(v(i))}, i = 1, 2, 3, ..., n$. Another input is the mixer learning algorithm (known also in literature as the meta-learner), that takes as inputs the constructed vector Z_i and predicts the output Y_i.

$$M = \widehat{M}(P_{n,y,z}) \tag{3.39}$$

where $P_{n,y,z}$ is the empirical distribution of the observations set $M = m_1, m_2, ..., m_n$. $m_i = (Z_i, Y_i)$ is the meta-data sample, Z_i is the stacked vector and Y_i is the assigned label to the input X_i.

If we consider the loss function $L(m_i, M) = [M(Z_i) - Y_i]^2$ to be the squared error loss function, then the minimum cross-validated risk predictor:

$$\widehat{K} = argmin_k \ \frac{1}{V} \sum_{v=1}^{V} \frac{1}{np} \sum_{i=1, V(v)}^{n} [Y_i - M_k(Z_i = \beta_j(P_{nT(v)}) : \ j = 1, 2, 3, ..., J)]^2 \tag{3.40}$$

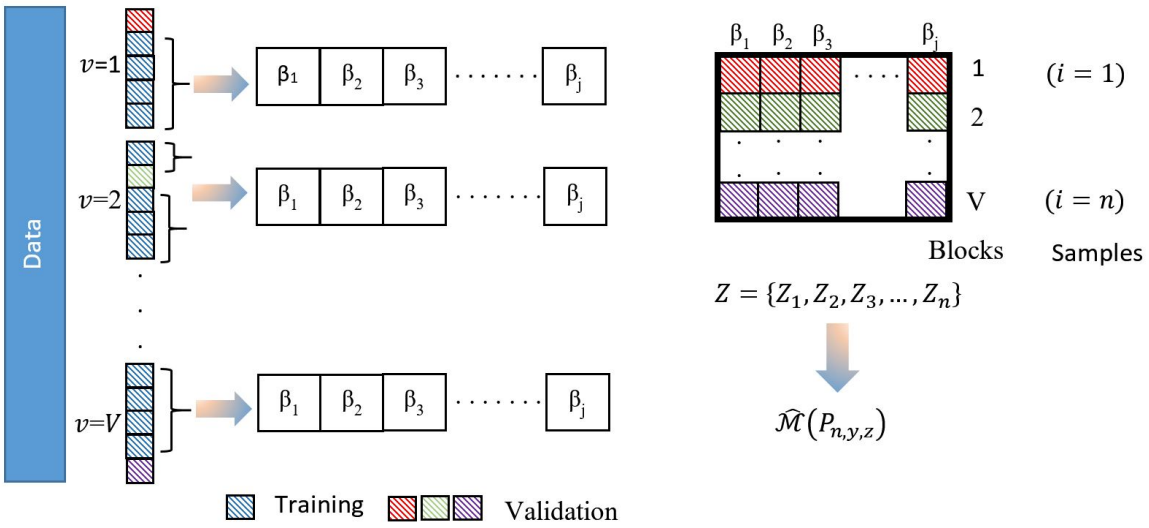

Figure 9: Cross-validation based mixer. V-splits and J learners are used. For each split, we use $(V-1)$ blocks of the data to train each learner and the remaining block is used for testing (validation). The predicted outputs are concatenated to construct the vector Z. V blocks are generated, one for each validation block. The $2-D$ matrix $n \times j$, where n is the number of samples and j is the number of learners, is used to train another learner M, the meta-learner.

where V is the number of splits (V-fold cross validation), p is the percentage of samples in the validation set, n is the total number of samples, thus np is the number of samples in the validation set. Since np and V are constants, the minimization equation can be written as:

$$\widehat{K} = argmin_k \sum_{v=1}^{V} \sum_{i=1,V(v)}^{n} [Y_i - M_k(Z_i = \beta_j(P_{nT(v)}) : \ j = 1,2,3,...,J)]^2 \tag{3.41}$$

$M_{\widehat{k}}(Z_i)$ is the optimal general mixer, which has been selected / generated by minimizing the loss function $L((Z_i, Y_i), M)$. In practice, after training the general mixer, the whole data is used to re-train all base learners. The mixer itself is a learning algorithm and it can be either a parametric predictor or a data adaptive minimum cross-validation general mixer. For the parametric predictor, one can define the minimization of the cross-validated risk criteria to be:

$$M_k = argmin_\omega \ R_{cv}(\omega) \tag{3.42}$$

where $R_{cv}(\omega) = \sum_{i=1}^{n} [Y_i - M(Z_i \backslash \omega)]^2$.

For example, $M(Z \backslash \omega)$ can be the linear regressor:

$$M(Z \backslash \omega) = \omega Z \tag{3.43}$$

$$= W_1 Z_1 \ + \ W_2 Z_2 \ + \ W_3 Z_3 \ + \ ... + \ W_J Z_J \tag{3.44}$$

Setting all the weights W_j to zero except one converts mixing problem to selection problem. In fact, selection problem is a special case of the more general mixing problem. In one of the very advanced mixing techniques, the weights themselves consists of two parts, one is related to the mixing problem and how to mix different learners (mainly the one explained above) and the second part is a function of the input X. One can think about this as an adaptive mixing technique that uses different mixing functions depending on the given input.

Given a set of learners $\beta_i, i = 1, 2, 3, ..., J$, these base learners are trained using: Different empirical distributions, $P_{n1}, P_{n2}, ..., P_{nk}$, where $P_{ni} \in P_n$ and $P_{ni} \in P_0$, different training algorithms $T_{A1}, T_{A2}, ..., T_{AQ}$ and a combination of changing distributions and training algorithms such that:

$$\zeta_i \ |_{i=1:j} = (P_{n1}, T_{A1}),\ ..., (P_{n1}, T_{AQ}),\ ...\ , (P_{nk}, T_{A1}),\ ..., (P_{nk}, T_{AQ}) \tag{3.45}$$

where $J = k \times Q$. Each ζ_i defines a combination of P_k and a learning algorithm T_{Aq}.

A general mixer is another learner that uses the outputs of all learners (base learners) to predict / estimate the final output. In the general mixer (homogeneous), all learners in the general mixer solve the same exact problem. We can define another two different types of general mixers: heterogeneous general mixer and adaptive heterogeneous general mixer. Heterogeneous general mixer consists of different learners that solve different problems. These learners can also use different distributions and different training algorithms. To define a mixer as heterogeneous mixer, at least two different problems should be defined within its base learners, $\beta_i, i = 1, 2, ..., J$, this means that we may still have multiple base learners working on the same problem. In the second type, adaptive heterogeneous general mixer, the adaptation comes in the context of how many learners are assigned to work with a specific problem.

One more variation can be added to base learners, different representations can be used for the same example. For example, in computer vision, different types of features can be extracted and used to represent each sample.

3.2 BagStack Classifier

In this section, we cover the proposed BagStack classifier. BagStack classifier is a stack-based classifier which is designed to deal with the data imbalance problem under the assumption of having multi-class classification problem and small number of samples. The objective is to achieve the

rational behind stacking when dealing with data imbalance problem and small number of samples. We discussed in the beginning of this chapter the cases when stacking is beneficial to be used. The rational behind stacking is to use different learning algorithms of different biases to transform the input sample from the input domain (where none of the available classifiers is capable of solving the problem) into another domain (the meta-data domain) where better performance is achievable by using the available leaning algorithms.

The quality of this transformation is a key factor that affects the quality of the whole classifier. Biased base learners, overfitting, underfitting, information loss and using redundant information are among some of the challenges that we will deal with. BagStack classifier uses two levels of cross-validation, Bootstrap and V-fold cross-validation. The spirit of BagStack classifier is to improve the quality of base learners, the quality of the meta-data and the quality of the meta-classifier.

3.2.1 Data Partitioning

Given a set of observations $S = S_1, S_2, ..., S_N$ of N samples, we will define three levels of partitioning. Level 1: Partitioning the N-observations into learning and testing sets. Level 2: Partitioning the learning set into V folds, where $(V - 1)$ folds are used for training and the last fold is used for validation. Level 3: Randomly, sample from the training set depending on the problem at hand to train each base learner.

In general V-fold cross validation, we define a vector $C_n = 0, 1$, that can take a value of 0 or 1. When $C_n = 0$ the sample is used for training and when $C_n = 1$ the sample is used for validation. In Bootstrap cross validation, we define a vector $C_n = 0, 1, 2, 3, ..., H$, where C_n is how many times a sample is selected. H is the number of times we sample from the set.Thus, H is the maximum number of times a sample can be selected. It is very important to understand that even when we sample N times from a set of N samples, the probability that a sample is not selected is $(1 - \frac{1}{N})$, if we sample N times then, the probability that a sample is not selected is $(1 - \frac{1}{N})^N = e^-1 = 0.36$. This means that 36% of the set will not be selected.

Imbalance-aware bootstrap sampling (oversampling, undersampling) gives attention to labels (classes) when performing the sampling. In oversampling, we sample a number of samples from class C more than the original available samples. It is a bootstrap sampling from class C with percentage $> 100\%$. In undersampling, we sample a number of samples from class C less than

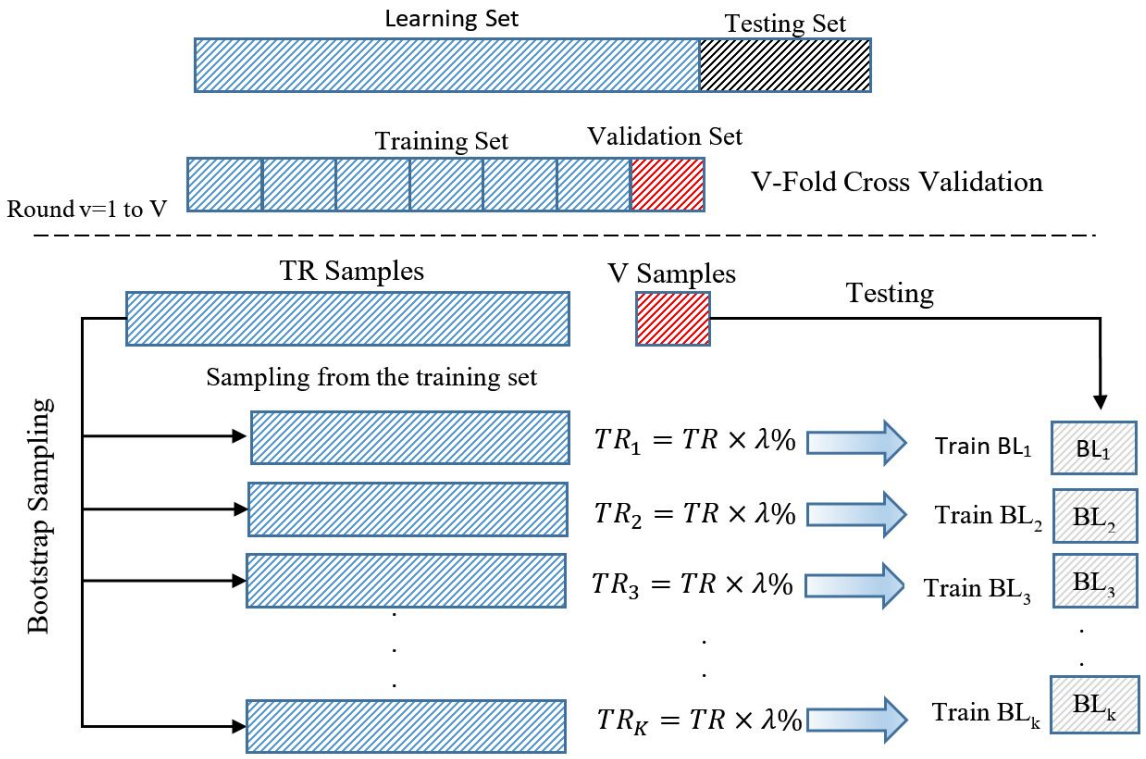

Figure 10: A diagram shows V-fold cross-validation, Bootstrap sampling and the way of combining both of them.

the original available samples. It is a bootstrap sampling from class C with percentage $< 100\%$. Oversampling and undersampling are used to deal with the data imbalance problem. Figure 10 shows the mechanism of sampling data to perform training. In this diagram, we show how one can use both of cross-validation and bootstrap sampling to generate a sub training set to train a base learner.

3.2.2 Information Loss/Redundancy and Overfitting/Underfitting Problems

For now, let us focus mainly on the Bootstrap sampling and consider the two cases oversampling and undersampling. Consider a classification problem of only two classes, binary classification problem, where the number of samples of class B (minority) is less than the number of samples of class A (majority). Dealing with data imbalance problem can be done by using either oversampling class B or undersampling class A.

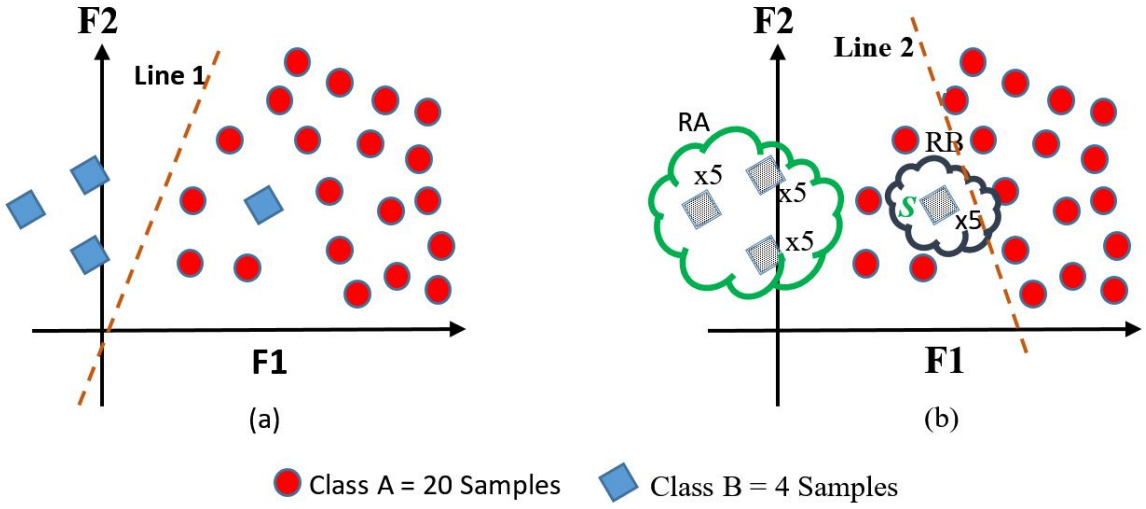

Figure 11: Data imbalance problem. (a) Line 1 represents a candidate solution for the problem. (b) Line 2 represents an alternative solution for the classification problem after solving the data imbalance problem by oversampling class B.

Oversampling minority classes: Sampling $|A|$ samples from the majority class A and the same number of samples from the minority class B, where $|A|$ is the number of samples belonging to the majority class A. This leads to redundant information, multiple copies of the minority samples belonging to the same original sample. This changes the behavior of the learning algorithm; the learning algorithm now is focusing more on the minority samples when minimizing the loss function. The risk of duplicating samples is increasing the tendency to have overfitting problem. Despite the fact that many of the new samples are just copies of the original data, the learning algorithm considers each sample as completely independent and unique sample. Because of duplicating samples, some of the features may mistakenly become more important than the others. These features may not be important. Asking the learning algorithm to focus on these samples will eventually lead to overfitting problem. The learning algorithm will learn specifically the few number of samples available in the original minority set. To explain how oversampling can affect the accuracy let us consider the following example.

Figure 11 depicts a classification problem of two classes, Class A is the majority class and Class B is the minority class. Training a linear model to classify these two classes, such that we maximize the overall accuracy, results into the line 1 hyper plane. Figure 11 (a) depicts this solution. Performing oversampling on class B, such that the total number of samples belonging to class

B becomes equal to $|A|$, then training a linear model to classify these two classes, such that we maximize the overall performance, results into the line 2 hyper plane. Figure 11 (b) depicts this solution. The difference between these two solutions is that the overall accuracy of case 1 (Figure 11(a)) is 96% and the average accuracy is 85%, while the overall accuracy after dealing with the data imbalance problem is 90% and the average accuracy is 90%, Figure 11(b). The learning algorithm gives more attention to the minority class which leads to correctly classifying all samples belonging to the minority class in the training set. In practice, the prediction accuracy on the testing data is more important. The sample S of class B looks like a noisy sample, it is not clear if this sample is a good representation of the class B or not. The distribution of the $Samples \in Class\ A$ is almost a uniform distribution, while the distribution of the $Samples \in Class\ B$ is nonuniform with higher probability to be within the region RA than the region RB.

Adding more samples to this training set can easily make the classifier unstable and thus changing its prediction. In fact, adding more samples and retrain may lead to a different base learner. If the sample S is a noisy sample while all other close samples belong to class A are real samples, the probability that a new sample in this region belongs to class A is higher than the probability that the new sample belongs to class B. This leads to a lower accuracy during the prediction. One of the issues of using oversampling is the phenomenon of giving more attention to wrong features. In fact, duplicating the same sample multiple times may give the illusion that some of the features (features that can be used to separate these samples from the other samples) are more important.

Undersampling majority class: Undersampling is the process of selecting a set of samples from the majority class. In practice, $|B|$ samples are sampled from the majority class, where |B| is the number of samples belonging to the minority class. Undersampling is more common than oversampling, it is used mainly with ensemble techniques (e.g., Bagging and Boosting). Despite the fact that the undersampling technique is powerful, there are some limitations connected to it. To understand these limitations, let us define the ratio of class A (majority) to class B (minority) as ρ:

$$\rho = \frac{|B|}{|A|} = \frac{number\ of\ samples\ in\ the\ minority\ set}{number\ of\ samples\ in\ the\ majority\ set},\ then\ 0 \leq \rho \leq 1 \tag{3.46}$$

The case, when $\rho = 0$, is not one of the cases that we cover. It is called the one class learning problem. In this problem, you learn only one class and you predict either (Yes), the sample belongs to this class, or (No), the sample does not belong to this class. The case when $\rho = \frac{1}{|A|}$ is another interesting problem called learning from one sample. When $\rho = 1$ there is no data imbalance problem. The range that we are interested in is $0 < \rho < 1$. This range is wide and having ρ equals

to 0.99 is not the same as ρ equals to 0.01. ρ significantly affects the number of base learners to be used when constructing an ensemble classifier.

Even though data imbalance problem is mainly defined by the ratio ρ, in practice, there are other factors that play major roles in deciding if there is a data imbalance PROBLEM or not. Data imbalance is mainly related to the ratio, but if there is a problem or not is also related to: $|A|$, $|B|$, the absolute number of samples, the variation within the majority class, the cost of missing the prediction of the minority class and having separable classes or not given a training algorithm.

Having severe data imbalance problem necessitates increasing the number of base learners. Training with small number of base learners may lead to information loss. Some of the majority samples may not be used.Training with large number of base learners may lead to have redundancy. Some of the minority samples may be used multiple times. If the variation within the class is very high and the samples are covering a large region of the data space, undersampling may lead to unstable version of the base learners. Later, we will explain how the ratio ρ,the absolute number of samples $|A|$ and $|B|$, the variation within classes and separability) affect the number of base learners.

Synthetic oversampling: In oversampling technique, the idea is to increase the number of samples of the minority class. The drawback of simply duplicating samples is increasing the tendency toward overfitting. In synthetic oversampling, we increase the number in the minority class by creating (synthesizing) new samples. The main idea is to use available samples to create new samples. SMOTE (Synthetic Minority Oversampling Technique) is one of the most common used techniques. In SMOTE algorithm, for each sample in the minority class: find the KNN samples, randomly select one of the KNN samples S_{KNN}, randomly choose a number between zero and one, and finally, synthesize the new sample such that: $S_{synth} = S_{original} + a \times d(S_{KNN}, S_{original})$, where $d()$ is the distance between the original sample and the S_{KNN} sample. Since the synthetic oversampling technique generates new samples that are not exact copies of the original samples, it reduces the tendency of oversampling toward overfitting.

3.2.3 BagStack Classifier: Mathematical Justification

In the section, we will explain the rational behind the BagStack classifier. We will cover the major concepts behind bagging and stacking and how combining both can be very useful.

The rational behind bagging can be understood by looking mainly at the two major errors in classification. The generalization error G_e is presented as a function of bias and variance errors. The generalization error G_e measures disparity between the predictor/estimator β and the true data generating process f. For a provided sample Y of f at q, where Y is a random sample of $P_f(.\backslash q)$, the generalization error can be defined as the square difference between $\beta(q)$ and the Y:

$$G_e = (\beta(q) - Y)^2 \tag{3.47}$$

If we train multiple versions of β for different training sets, the expected value $\beta(q)$ will be:

$$\beta^*(q) = E[\beta(q)\backslash f, m] \tag{3.48}$$

$$\beta^*(q) = \sum_d (P(d\backslash f, m) \times \beta_d(q)) \tag{3.49}$$

where $\beta^*(q)$ is the expected prediction, f is the true data generating distribution, m is the total number of samples used in the training and $\beta_d(q)$ is the realization of $\beta(q)$ when we use the d training set of m samples to train the predictor $\beta()$. $P(d\backslash f, m)$ is the probability of generating the d training set. The expected generalization error given f, m and q is given by:

$$E[G_e\backslash f, m, q] = E_d[(\beta_d(q) - Y)^2] \tag{3.50}$$

$$= E_d[(\beta_d(q) - E_d[\beta_d(q)] + E_d[\beta_d(q)] + E_d[\beta_d(q)] - Y)^2] \tag{3.51}$$

$$= E_d[(\beta_d(q) - E_d[\beta_d(q)])^2] + E_d[(E_d[\beta_d(q)] - Y)^2] + \ irreducible\ error \tag{3.52}$$

$$= Variance\ +\ (Bias)^2\ +\ irreducible\ error \tag{3.53}$$

If our training algorithm responds with the same hypothesis $\beta_d(q)$, the variance will be zero, since the hypothesis is not changing while changing the training set. The bias will be constant and equals the bias results from any of the given hypothesis. If the training algorithm is unstable, then, the expected error of that algorithm that always guesses $E_d[\beta_d(q)]$ regardless of d is significantly less than that of the training algorithm $\beta_d(q)$. The difficulty in exploiting this effect to aid real world generalization is the fact that we cannot evaluate the expected value $E_d[\beta_d(q)]$. Because we have only one training set and not a set of training sets we cannot evaluate this expected value. In the original paper of Bagging [9], the authors argued that one can produce a mimic of $E_d[\beta_d(q)]$. By using the available training set and Bootstrap we can generate several training sets by sampling from the training set. Repeating this process (sampling and training) to a degree that the mimic of

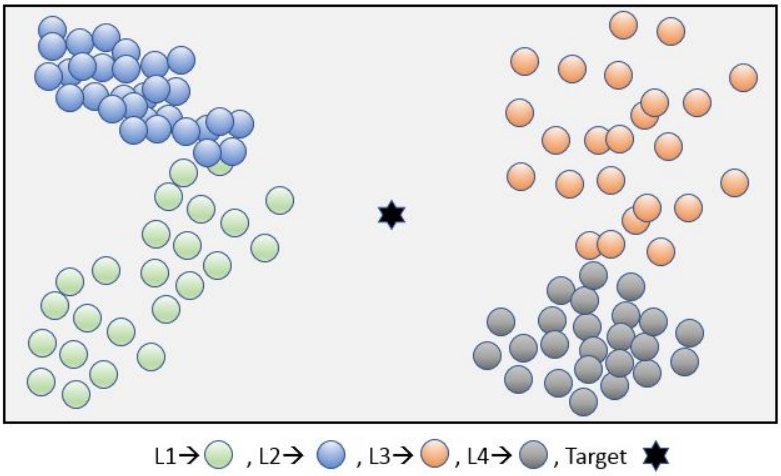

Figure 12: Predictions of the same input sample q using different realizations of 4 different learning algorithms.

$E_d[\beta_d(q)]$ approximates the true $E_d[\beta_d(q)]$, then guessing with $E_d[\beta_d(q)]$ rather than $\beta_d(q)$ will result in lower expected generalization error.

The above explanation is just a mathematical justification of why we believe that bagging can reduce the generalization error. In fact, this is not a proof. There is no mathematical proof that bagging will always reduce the generalization error. (Even boosting has no mathematical proof it will always reduce generalization error). Bagging is very significant when we have unstable learning algorithms. In this case, taking the average of a slightly better than random guessing classifiers can reduce the variance error.

Deciding, if bagging is helpful or not, depends mainly on the generalization error and the main components that construct this error **(bias and variance)**. Bagging is very helpful if the bias error is zero and the variance error is not. In this case, using bagging can reduce the variance error and thus the overall generalization error. In the case when variance error equals to zero, bagging is not helpful. There will be no gain of using it. One of the interesting cases is when both variance and bias errors are not zeros. In the last case, bagging is useful if there is a way to shift the bias of the overall classifier toward the right target.

To explain the rational behind stacking, let us assume that we have multiple learning algorithms $L_1, L_2, \ldots, L_4$, these learning algorithms have different biases. Figure 12 depicts the predictions of each one of these learning algorithm, when using different training sets to train each one of them, on the sample q.

Given a training set TR, using any training algorithm to generate a classifier (1 classifier) will not be helpful and will lead to a large generalization error. $L1$ and $L3$ have both bias and variance errors, $L2$ and $L4$ have only bias error. Using bagging with these learning algorithms can reduce the variance error (if it is not equal to zero), but none of these algorithms will be able to generate a classifier (strong ensemble classifier) with minimum (close to zero) generalization error. None of the bagging ensembles, which has been generated using any of these training algorithms, can lead to zero bias error.

cross-validation is the most common used method for selecting classifiers. It can be used to select the classifier with the minimum bias and variance error. If cross validation can be used to select a classifier, why do we need stacking? In the last example (Figure 12) since none of the classifiers has a non-zero bias error, cross-validation based selector will not be helpful and at the best it will select the classifier with the minimum none-zero bias error.

The only option we have is to understand the outcomes of these learning algorithms. One can come up with a conclusion based on these outcomes. In this case, even wrong predictions may contain useful information that can be used to make the final prediction. This is the answer for why stacking-based classifier is important. Stacking-based classifier uses the outcomes of different learning algorithms to represent the question q in another domain, where one of the available learning algorithms can solve the problem. Figure 13 depicts three different techniques of combining the outcomes of different learning algorithms. Figure 13 (a) shows the case when using K training algorithms, Figure 13 (b) shows the case when we use bagging explicitly to generate different realizations of the same learning algorithm and Figure 13 (c) depicts the case when using bagging implicitly to generate different realizations and combine the outcomes of these base learners using another learning algorithm (meta-classifier).

In fact, the set of learning functions that can be represented (learned) by using bagging to generate different ensembles followed by cross-validation to select the best ensemble classifier is completely contained in the set of functions that can be represented using the classifier depicted in Figure 13 (b). The set of functions that can be represented using the classifier depicted in Figure 13 (b) is completely contained in the BagStack classifier, Figure 13 (c). Stack-classifier consists of two levels of learning. L_0, is the first level, it consists of a set of base learners. L_1, is the second

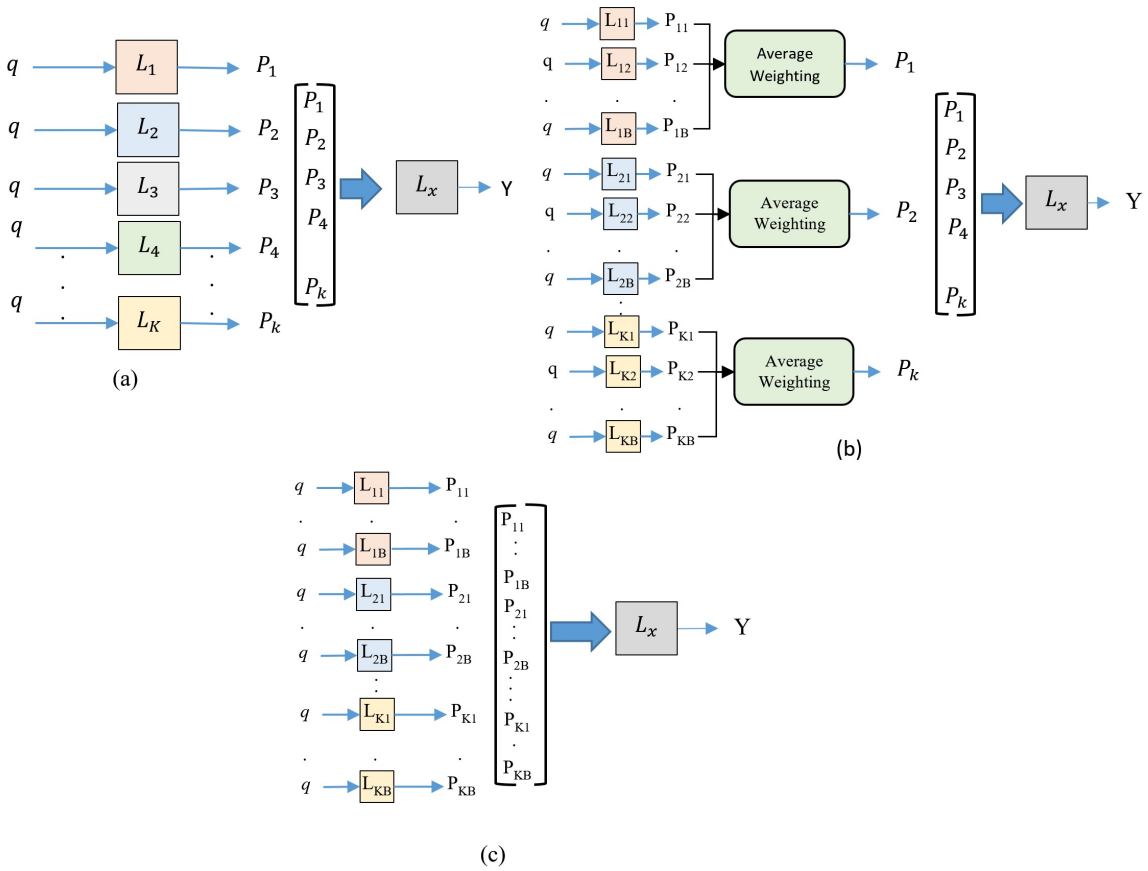

Figure 13: (a) General Stacking based classifier, (b) stacking based classifier using ensemble bagging classifiers using different learning algorithms (explicitly apply bagging followed by stacking) and (c) BagStack Classifier.

level, it consists of the meta-classifier which takes as input the outcomes of the first layer. Data generated by the first layer is called the meta-data.

3.2.4 Cross-validation Based Stack and BagStack Classifiers

In a cross-validation based classifier, the learning data is divided into V-splits. One split is used for validation and the (V-1) splits are used for training.

Algorithm 1 is the training algorithm of stack-based classifier using cross-validation. We have no assumptions about the problems as well as the learning algorithm that is used with each base learner: Different algorithms: SVM, DT, … etc., different problems: one-vs-one, one-vs-all, Multi-Class, … etc., different data representations: different features. The objective function of the

Algorithm 1 Cross-Validation Based Stack-Classifier.

1: **Step 1: Generate the Meta-Data.**
2: Initialization: $BL = \phi$, $M = \phi$.
3: **for each** split v **do**
4: Use (V-1) splits, all $splits \in V$ except v, to construct the training set T_R.
5: Use the last split v to construct the validation set T_V.
6: **for each** base learner nbl **do**
7: Use the training set T_R to train the base learner BL_{nbl}.
8: Add this base learner to the base learning set:$BL_{nbl} \rightarrow BL$.
9: **end for**
10: **for each** sample in the validation set v **do**
11: Test the current sample to construct the vector Z by concatenating the predictions of all base learners in the base learning set BL.
12: Add the vector Z to the meta data set M.
13: **end for**
14: Reset the base learners set $BL \leftarrow \phi$.
15: **end for**
16: **Step 2: Use the meta-data M to train the meta-classifier ML.**
17: **Step 3: Use all the data in the learning data set to re-train each one of the base learners.**

meta-classifier $(M_i = (z_i, y_i))$ is to reduce the risk (average risk) across different splits. The $Loss\, function = min\, E_v[Loss(m_i, ML)]$, where m_i is a sample meta-data. Then the meta-learning algorithm:

$$ML = argmin_{ML} E_v[Loss(m_i, ML)] \tag{3.54}$$

$$= argmin_{ML} \sum_{i=1}^{N} (y_i - ML(m_i))^2 \tag{3.55}$$

where N is the number of samples belonging to the learning set. Improving the quality of the meta-data is important to improve the training of the meta-classifier. The first layer of base learners is a transformation process that transform the sample from input domain (X_i, Y_i) to the meta-data domain (Z_i, Y_i).

How to guarantee the quality of this transformation? The first step to improve the meta-data is to improve the quality of the base learners. Since we have no pre-assumptions about the performance of the given training algorithms on the training data, we need to improve the performance of these base learners. Reducing variance error is done by generating multiple base learners using the same training algorithm and solving the same problem. In bagging, the average of the outcomes is used to represent the final outcome. In BagStack, the meta-classifier learns how to use the outcomes the best way to maximize the overall accuracy. In cross-validation based BagStack classifier (Algorithm 2), the data used to generate the meta-data have not been seen by the base classifiers during the current iteration of training. This improves the quality of the meta-data.

Algorithm 2 Cross-Validation Based BagStack-Classifier.

Step 1: Generate the Meta-Data.
Initialization: $BL = \phi$, $M = \phi$.
for each split v **do**
    Use (V-1) splits, all $splits \in V$ except v, to construct the training set T_R.
    Use the last split v to construct the validation set T_V.
    for each training algorithm L **do**
        for each base learner nbl **do**
            Randomly sample from the training set T_R using Bootstrap sampling.
            Train the current base learner BL_{nbl}.
            Add this base learner to the base learning set: $BL_{nbl} \rightarrow BL$.
        end for
    end for
    for each sample in the validation set **do**
        Test the current sample to construct the vector Z by concatenating the predictions of all base learners in the base learning set BL.
        Add vector Z to the meta-data set M.
    end for
    Reset the base learners set $BL \leftarrow \phi$.
end for
Step 2: Use the learning data, retrain the BL.
for each training algorithm L **do**
    for each base learner nbl need to be trained by the current training algorithm **do**
        Randomly sample from the training set $(T_R \cup T_V = V - Splits)$ using Bootstrap sampling.
        Train the current base learner BL_{nbl}.
        Add this base learner to the base learning set: $BL_{nbl} \rightarrow BL$.
    end for
end for
Step 3: Using the meta-data, train the meta-classifier.

In fact, Algorithm 2 does not answer the following important questions: how many base learners to be used to solve the same problem using the same training algorithm? Do we need to use the same number for different problems / different training algorithms? What is the percentage of samples to be sampled from the training data when training a base learner? Is it the same percentage for all problems? In this algorithm, we do not consider also the types of problems that need to be solved by different base learners. We assume that all base learners solve the same problem but using different training algorithms.

The meta-data is a very important aspect of stacking-based classifiers. The meta-data can be either labels, probabilities or confidence values. While labels indicate the classes without giving any information about certainty, probabilities and confidence values give more information. These two values provide more information about how much the base learner is sure about its prediction. This is used by the meta-classifier to make the final decision.

Algorithm 3 MC-Classification Using one-vs-all Binary Classifiers.

1: **for each** class c **do**
2: Train a binary classifier to solve the problem $Class\ C\ vs.\ all\ other\ classes$.
3: **end for**

Algorithm 4 MC-Classification Using one-vs-one Binary Classifiers.

1: **for each** class c_1 **do**
2: **for each** class c_2, $s.t\ c_1 \neq c_2$ **do**
3: Train a binary classifier to solve the problem $Class\ c_1\ vs.\ c_2$.
4: Add this base learner to the base learner set BL.
5: **end for**
6: **end for**

3.2.5 Cross-validation Based Multi-class Classifier and Heterogeneous General Mixer

Multi-class classification is the problem of classifying three or more classes (dealing with two classes is a binary classification problem). Multi-class classification problem is an interesting problem since most of available training algorithms are designed mainly to deal with binary classification problems.

In this section, we will study two different techniques that are used to convert the multi-class classification into several binary classification problems. These binary classification problems can be different in the context of types of problems and types of training algorithms. Some of the problems can be easy and separable, others can be more difficult and need extra ordinary effort to be solved.

In the one-vs-all category, the MC-classification problem is transformed to multi- one-vs-all binary problems. In it is simplest format, the number of binary classification problems is directly related to the total number classes.

To perform prediction, given a question X. Test the question using all base learners and pick the class c of the classifier that predicts c with the highest probability. If no classifier predicts c, pick the class c of the classifier that predicts (all) with the smallest probability. Even though this prediction technique (transformation of a multi-class problem to binary problems) looks good and makes sense, there are two main problems related to it. (1) It is not clear how this algorithm deals with the bias error and variance error within the binary classifiers. What if one of the classifiers is biased toward the class c, meaning the binary classifier of $c\ vs.\ all$ always predicts c. This may happen if the number of samples belong to class c is very large. This will affect the whole prediction

technique leading to a classifier that is biased toward predicting the class c. (2) The information used to generate the final prediction is only related to the classifier of max or min probability. All other outcomes are ignored. An alternative technique is to consider the output of all these classifiers as meta-data that can be combined to construct a feature vector. Instead of using max or min a classifier is trained to generate the final prediction. Since we are using the same training algorithm, the main objective of the meta-classifier is to understand the dependency (correlation) between different predictions (different problems) to predict the final label.

In this context, the transformation of MC classification problem into a C binary classification problems, the stacking used mainly to transform the problem from one domain into another different domain where making a decision about the final prediction is easier and more accurate. You can think about this as breaking a difficult problem into a set of smaller and simpler problems.

One of the major limitations of converting the problem (MC-problem) to a one-vs-all classification problem is the data imbalance problem. In fact, one-vs-all classification problem is inherently imbalance, usually the number of samples belonging to the (all class) is more than the one class. Moreover, if the MC problem has already a data imbalance problem, transforming the problem to a one-vs-all will make the problem more imbalance.

In one-vs-one, the multi-class classification problem is transformed into a set of one vs. one classification problems. In each one of these problems, we try to distinguish between two different classes. Testing can be done by considering each one of these base learners. For example, we can find the sum/average of probabilities of the class c over all classifiers that contains class c as one of its classes. The class with the max probability is selected. If two classes return the same max probability, randomly, pick one of them. Again, if one of the classes is dominant and thus bias error exist, all classifiers that are trained using data from this class and any other classes will be biased which leads to overall biased predictions. An alternative, one may consider processing the outcomes of these base learners using another meta-classifier.

The one-vs-one has the advantage of being able to focus on a very simple problem. Less data imbalance and more information can be used through the transformation process (since no undersampling is required). For each class, $C-1$ base learners are created, thus the total number of base learners will be $NBL = \sum_{c=1}^{C}$ (C-c). During the testing process, each sample will be tested by $NBL - C + 1$ base learners that have not seen this type of sample during the training process (when testing class A with a base learner that is trained on class B vs. D). The outputs of these base

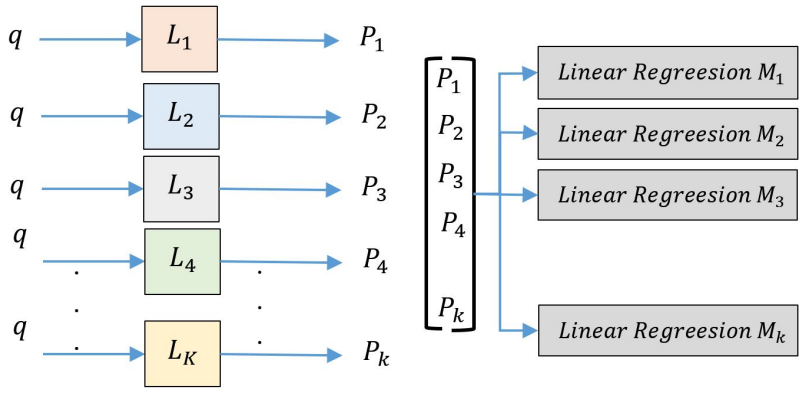

Figure 14: Using MLR as a meta-classifier with one-vs-all base learners.

learners are considered as random guessings. The collinearity level is higher in this transformation than the $one\ vs.\ all$ based transformation. There is no way to deal with bias and variance errors. Meta-classifier is looking at completely different problems. The meta-classifier will only study the relation between the outcomes to come up with final prediction.

Using ensemble-based multi-response DT regression EB-MDTR as a meta-classifier for the heterogeneous general mixer: For all previous analysis, we consider a single meta-classifier that takes as input the outcomes of the base learners. In heterogeneous general mixer, different types of problems are defined at the base learner level. Categorizing these different problems and defining multiple specialized meta-classifier is useful to help each one of these meta-classifiers to focus on a very specific problem, usually in the form of: $does\ the\ sample\ belong\ to\ class\ c\ or\ not?$

In [56, 57, 58, 59], the authors suggested using the MLR, wherein a linear regression problem is defined per class. The inputs (feature input) of each one can be different. The outcomes of the base learners can be grouped into different features. In $one\ vs.\ all$ implementation, the outcomes of base learners are combined to construct a single feature. Figure 15 depicts the process of using the MLR with $one\ vs.\ all$ implementation.

Even though all these models $M_1 - M_k$ use the same feature input, they use different labels during the training for the same feature input. If the sample S_i, which is used to generate the vector Z_i, is labeled as class c_j, the vector Z_i is labeled as 1 when training the model M_j and 0 when training all other models. We are expecting that each model $M_1 - M_k$ will take care of understanding the relation between the outcomes of different base learners and reducing the bias. It is very important to notice that the probabilities $P_0, P_1, ..., P_k$ are generated by using different base

learners and different problems. It is not the same as bagging, where all outcomes are generated by base learners which solve the same problem.

In this thesis, we propose the ensemble-based multi-response DT regression as a metaclassifier. Instead of using linear regression, we use DT regression. Instead of fitting one regression function for each class we fit an ensemble of DT regression for each class. We randomly sample from the training data (meta-data) and fit each one of these DT regression functions. In the prediction phase, the output of each model corresponding to a specific class is defined as the average of all predictions (bagging). The final prediction of the BagStack classifier is defined as the class corresponds to the ensemble model of the maximum prediction.

Instead of using bagging of decision tree regression functions, we can use random forest regression. Random forest uses another level of randomness that improves the ability of the model to resist overfitting. In bagging, we randomly select samples from the training set with replacement to construct a new subset, usually we select ($60\% - 80\%$) of the training data to train each model in the ensemble. In random forest, we randomly select samples from the training set to construct a new subset that is used to train each model (DT) in the ensemble. Random forest uses another level of randomness. At each node in the tree, instead of using all the features to select one to split the data at this node, random forest selects randomly a subset of the features at each node and only this subset is used in the evaluation to select one to split the data.

Increasing the number of base learners increases the dimension of the meta-data. In the case when we have high imbalance ratios between classes, high numbers of base learners are used to achieve a specific coverage percentage, the higher the number of base learners used to solve a specific problem, the higher the probability that two features are highly corelated. In theory, if Cor is a measure of the corelation between features in the meta-data, at some point if Cor is greater than a threshold, random forest converges to bagging of decision tree. To increase the resistance of the random forest against overfitting, we introduce a new level of randomness. the new level is the depth of the tree. For the first tree in the ensemble we use optimization techniques to find the optimal depth (D), for all subsequent trees we set the max depth parameter D' to be $D' = D - D \times \lambda$, where lambda is a small positive random number (e.g., 0 - 0.15).

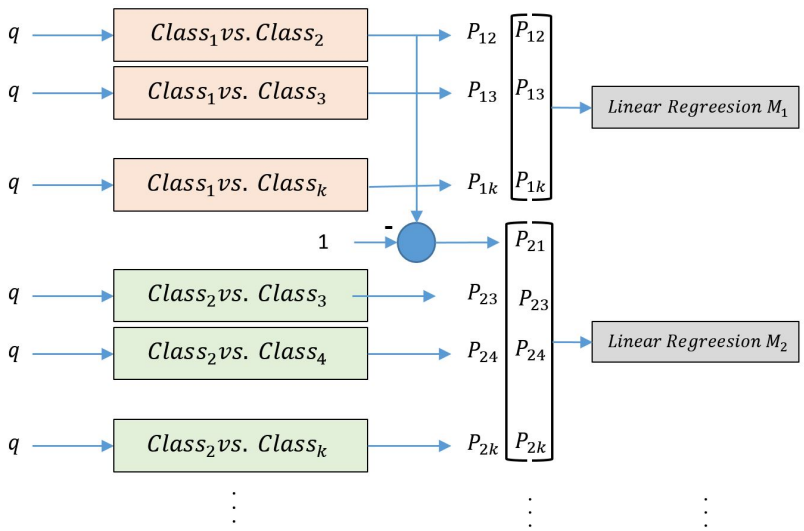

Figure 15: Using MLR as a meta-classifier with one-vs-one base learners.

3.3 BagStack Classifier for Data Imbalance Problem

Given a set of learning samples $S = S_1, S_2, S_3, ..., S_N$, where each $S_i = (X_i, Y_i)$, X_i is the input feature and Y_i is the label. $Y_i \in 1, 2, 3, ..., C$, C is the number classes. BagStack classifier uses the concept of bootstrap sampling and V-fold cross validation to deal with both of variance and bias errors. While bagging is used to reduce variance error, stacking is used to perform general selection and reduce the bias error.

In our final implementation, we use bootstrap and V-fold cross-validation together at two different levels. The MC classification problem is transformed into a one vs. one classification problem. The number of base learners are assigned to each problem (one-vs-one) is specified adaptively as a function of the ratio ρ of the minority class to the majority class, $0 \leq \rho \leq 1$, the absolute number of samples of each class and the variation within classes. The main idea behind making the number of base learners per problem adaptive is assign different numbers of base learners to different problems.

In this section, we will explain in detail the BagStack classifier, splitting the data and sampling, defining the base learners, generating the meta-data and different meta-features, the different meta-classifiers and how we will perform the final prediction.

Data imbalance problem represents one of the major problems in machine learning. Data imbal-

ance is important for the following reasons: (1) It is a property of many of the practical problems, in many cases, the minority class is more important. (2) Data imbalance results into biased classifiers.

BagStack classifier deals with the data imbalance problem at different levels using different techniques: Adaptive number of base learners, avoiding overfitting, underfitting and information loss, and variance-based adaptive synthetic minority technique (VA-SMOTE). Algorithm 5 represents the process of training a BagStack classifier using Cross-Validation and one-vs-one binary classifiers.

3.3.1 Adaptive Number of Base Learners

Given a set of base learners $BL_{set} = BL_1,\ BL_2,\ ,...,\ BL_{NBL}$, where (NBL) is the total number of base learners to be used. Given a set of learning algorithms $L = L_1, L_2, ..., L_{LA}$, where LA is the total number of learning algorithms. For a given C-classification problem, where C is the number of classes, $NBLpLA$ is the number of base learners per learning algorithm, where $NBLpLA = \frac{NBL}{LA}$, the total number of base learners divided by the total number of learning algorithms. For C classes, we define E problems (one vs. one problem), where $E = \sum_{i=1}^{C}(C - i)$. Let us define the problem $Prob_e(AB)$ as the one-vs-one classification problem $Class\ A\ vs.\ Class\ B$.

$|A|$ is the number of samples belonging to class A and available for training (usually the training set during generating the meta-data and learning set during generating the final base learners). In the same context, $|B|$ is the number of samples belonging to class B. The higher the data imbalance ratio for a specific problem, the more base learners assigned to solve it. This is motivated by the concept of using bagging and undersampling to solve the data imbalance problem. In fact, bagging can be used with undersampling where the number of samples that are sampled from class A (the minority) and class B (the majority) is equal the number of samples $|A|$. To avoid information loss and to guarantee that all (most) of the samples in B are used we need to train higher number of base learners as we have higher data imbalance ratio. Figure 16 depicts the process of using undersampling to train class A vs. class B.

Algorithm 5 Cross-Validation Based BagStack Classifier Using one-vs-one Binary Classifiers.

Two levels of sampling, V-fold cross-validation and Bootstrap sampling.
Transforming the MC Classification into one-vs-one binary classifiers.
T_R: Training set, T_V: Validation set, TL: Learning set, NBL: The number of base learners, LA: Learning algorithms, V: V-folds, C : The total number of classes.
Initialize $BLSET = \phi$, $M = \phi$, $MLSET = \phi$.
Step 1: Generate Meta-data.
for each split v in the V-splits **do**
    Use the $(V-1)$ splits to construct the training data set T_R.
    Use the split v to construct the validation set T_V.
    Define the number of base learners per learning algorithm LA:$NBLpLA = \frac{NBL}{LA}$, where LA is the total number of training algorithms.
    For C classes, define the set of one-vs-one problems.
    for each problem e **do**
        Find the number of base learners $NBLLA_e$ to be used to solve the problem e, where: $NBLpLA = \sum_{e=1}^{\frac{c\times(C-1)}{2}} NBLLA_e$
    end for
    $BLSET = \phi$
    for each learning algorithm LA **do**
        for each defined problem $e: \ class\ A\ vs.\ class\ B$ **do**
            for each $BL: \ 1\ to\ NBLLA_e$ **do**
                Sample from class A and class B, T_R.
                Using the learning algorithm LA, train the base learner BL, $BLSET \leftarrow BL$.
            end for
        end for
    end for
    By using the validation set T_V, test each base learner.
    for each sample in the T_V set **do**
        for each base learner in the $BLSET$ **do**
            Test the sample.
            Construct the feature vector Z_i.
        end for
        Add the meta-sample (Z_i, Y_i) to the meta-data set $MLSET$.
    end for
end for

Step 2: Generate the final base learners.
$BLSET = \phi$
for each learning algorithm LA **do**
  for each defined problem e: $class\ A\ vs.\ class\ B$ **do**
    for each base learner BL : 1 to $NBLLA_e$ **do**
      Sample from class A, class B, T_L.
      Using the learning algorithm LA, train the base learner BL.
      $BLSET \leftarrow BL$
    end for
  end for
end for
Step 3: Generate the Meta-Classifier/Estimator
for each class c in the class set $1, 2, 3, ..., C$ **do**
  $MC_SET = \phi$
  for each sample in the set $MLSET(Z_i, Y_i)$ **do**
    Define the vector $K[]$.
    Counter=1
    for each base learner, $j = 1 : nbl$ **do**
      $Z[j]$ has been generated based on the problem $Class\ A\ vs.\ Class\ B,\ 0 \leq Z[j] \leq 1$.
      If $A = c$, $K[counter] = Z[j]$, $counter = counter + 1$
      If $B = c$, $K[counter] = 1 - Z[j]$, $counter = counter + 1$
    end for
    Add the sample $(K_i, Y_i \rightarrow MC_SET)$.
  end for
  By using the sample in MC_SET, fit a linear regression model MLC.
  $MLC \rightarrow MLSET$**.**
end for

Algorithm 6 Finding Number of Base Learners.

$\rho[] \leftarrow\ empty$**.**
for each problem $Prob_e(AB) \in Set\ E$ **do**
  Find the ratio $\rho(Prob_e(A, B)) = \frac{min(|A|,|B|)}{max(|A|,|B|)}$
end for
for each problem $Prob_e(A, B) \in Ste\ E$ **do**
  Normalize the ratio as: $\rho(Prob_e(A, B)) = \frac{(\frac{1}{\rho(Probe_e(A,B))})}{\sum_{i=1}^{E}(\frac{1}{\rho[i]})}$
  $NBLLA_e(Prob_e(A, B)) = \rho(Prob_e(A, B))\ \times\ NBLpLA$
end for

In fact, the ratio of the number of samples belonging to majority class to the number of samples belonging to minority class is not the only factor that we can use to evaluate the data imbalance problem. Three more different parameters should also be considered: The absolute number of samples. Mainly, the absolute number of samples of the minority class, the variation within the majority class, the used training algorithm.

The data imbalance problems (A= 10 vs. B=100) and (B= 100 vs. D=1000) have the same data

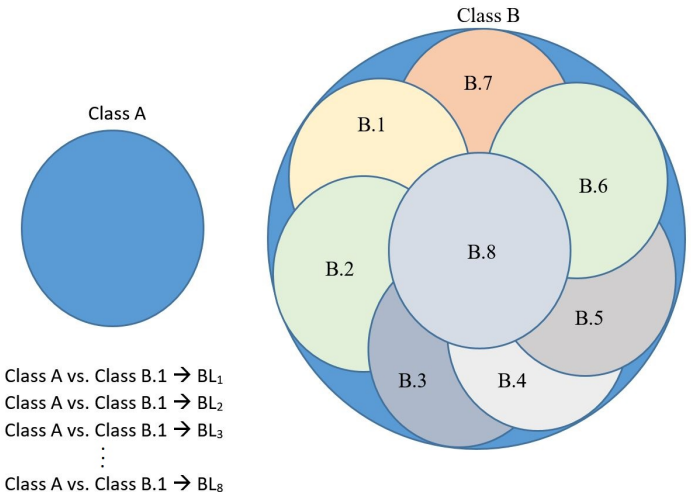

Figure 16: Using undersampling to train a data imbalance problem Class A vs. Class B.

imbalance ratio. Using undersampling and one base learner will not be sufficient for the first problem, the information loss in the majority class will be significant. In the second problem, using one base learner is enough to capture most of the information in the two classes. Using data imbalance ratio to specify the number of base learners to be assigned for each one of these problems may not be sufficient. Instead of using the same number of base learners for the two problems, it will be better to give more base learners to the first problem.

Variation within the class represents another factor that should be considered when assigning base learners to different problems. If the variation within the class D is higher than the variation within classes A and B, using more base learners to represent this class can improve the performance. Algorithm 7 shows the way of calculating the number of base learners assigned to each problem when we consider the imbalance ratio, the number of samples in the minority class and the variation between the majority class.

The last factor is the used training algorithm. Instead of using the same $NBLpLA$ of base learners for each training algorithm, different numbers for different training algorithms can be used. Known learning algorithms to be unstable are assigned more base learners than stable learning algorithms. Measuring the stability of an algorithm is done through repeatedly training / testing the learning algorithm to measure the variance error corresponding to this training algorithm. Assigning more base learners is beneficial to reduce the variance error.

3.3.2 Avoiding Overfitting, Underfitting and Information Loss

Bagging uses bootstrap sampling to select a subset of samples from the training set to train each base learner. Each time $P\%$ of the training set is used to perform training. In general bootstrap sampling, $P\%$ of the training data is sampled without giving any attention to different classes. If the original data is imbalanced, then randomly sampling results in a new training data set will also be imbalanced.

Algorithm 7 Finding Number of Base Learners.

1: **for each** $Prob_e(AB) \in Set\ E$ **do**
2: Find the ratio: $\rho(Probe_e(A,B)) = \frac{min(|A|,|B|)}{max(|A|,|B|)}$
3: **end for**
4: **for each** problem $Prob_e(AB) \in Set\ E$ **do**
5: Normalize the ratio as: $\rho(Prob_e(A,B)) = \frac{(\frac{1}{\rho(Prob_e(A,B))})}{\sum_{i=1}^{|E|}(\frac{1}{\rho[i]})}$
6: **end for**
7: **for each** problem $Prob_e(AB) \in Set\ E$ **do** $\delta(Prob_e(A,B)) = \frac{min(|A|,|B|)}{\sum_{Prob_e}^{|E|} min(|A|,|B|)}$
8: **end for**
9: **for each** problem $Probe_e(AB) \in Set\ E$ **do** $\lambda(Prob_e(A,B)) = \frac{Variation(Maj(A,B))}{\sum_{probe}^{|E|} Variation(Maj(A,B))}$
10: **end for**
11: Variation() is a function used to find the variation within a class. The total number of base learners assigned to each problem is given by the following formulas:
12: $ACC(Prob_e(A,B)) = \alpha_1 \times \rho(Prob_e(A,B)) + \alpha_2 \times \delta(Prob_e(A,b)) + \alpha_3 \times \lambda(Prob_e(A,B))$, where $\alpha_1 + \alpha_2 + \alpha_3 = 1$
13: $NBLLA_e(Prob_e(A,B)) = \frac{ACC(Prob_e(A,B))}{\sum_{Prob_e}^{|E|} ACC(Prob_e)} \times NBLpLA$

When we use bootstrap sampling, each time we sample, the probability that a data point (instance) is selected is $\frac{1}{N}$, the probability that a data point is not selected is $1 - \frac{1}{N}$, sampling N times from the training sample results in probability of $(1 - \frac{1}{N})^N = e^{(-1)} = 0.36$ that a sample is not selected. In other words, if we select N samples from a training set of size N with replacement, 36% of the samples will not be selected. In bagging, since we perform bootstrap sampling multiple times (depending on the number of base learners), the probability that a sample is not selected is getting smaller. For example, for two base learners, if we sample N samples to train each base learner, the probability that a sample is not selected $(1 - \frac{1}{N})^{2\times N} = 0.36^2 = 0.129\%$ thus 12.9% of the training data samples will not be selected.

How is this related to BagStack classifier? In BagStack classifier we use undersampling to deal with the data imbalance problem. For a given problem $Class\ A\ vs.\ Class\ B$, if $|A| < |B|$, How to

ensure that at least $F\%$ of the majority class samples are selected and used in the training process? How many base learners we need to assign for this problem to guarantee this percentage?

The probability that a sample belongs to the majority class is not selected (Sample Not Selected $SNotS$) when performing random undersampling:

$$P_{SNotS} = (1 - \frac{1}{|Majority|})^{|Minority|} \tag{3.56}$$

where $|Minority|$ is the number of samples belonging to the minority class and $|Majority|$ is the number of samples belonging to the majority class.

To guarantee that the percentage (coverage percentage of the majority class) is higher than $F\%$:

$$(1 - \frac{1}{|Majority|})^{|Minority|} \leq 1 - F\% \tag{3.57}$$

To find the total number of base learners to train:

$$(1 - \frac{1}{|Majority|})^{|Minority| \times NBL} \leq 1 - F\% \tag{3.58}$$

$$|Minority| \times NBL \times \log(1 - \frac{1}{|Majority|}) = \log(1 - F\%) \tag{3.59}$$

$$NBL = \frac{\log(1 - F\%)}{|Minority| \times \log(1 - \frac{1}{|Majority|})} \tag{3.60}$$

For example, given the number of samples in the minority class is 100 and the number of samples in the majority class is 230, to guarantee a usage percentage of $F\% = 90\%$, the total number of base learners needs to be used is:

$$NBL = \frac{\log(1 - F\%)}{|Minority| \times \log(1 - \frac{1}{|Majority|})} \tag{3.61}$$

$$NBL = \frac{\log(1 - 90\%)}{100 \times \log(1 - \frac{1}{230})} = \frac{-1}{100 \times -0.0018} \tag{3.62}$$

$$NBL = 5.28 = 6 \; Base \; Learners \tag{3.63}$$

To guarantee that at least 90% of the majority class samples are selected, 6 base learners should be used. The whole problem of specifying the number of base learners for each problem (class A vs. class B) can be re-designed to deal with the assumption (restriction) that at least ($F\%$, the coverage percentage) of the majority class of each problem should be used.

Algorithm 8 Finding the total number of base learners NBL.

1: $NBL = 0$**, the total number of base learners**.
2: $F\%$**, the minimum coverage percentage of the majority class**.
3: **for each** problem $Prob_e(AB) \in Set\ E$ **do**
4: MAJ_c is the majority class.
5: MIN_c is the minority class.
6: The number of base learners assigned to the problem $Prob_e(AB)$.
7: $NBLLA_e(Prob_e(A,B)) = \frac{\log(1-F)}{|MIN_c| \times \log(1-\frac{1}{|MAJ_c|})}$.
8: $NBL = NBL + NBLLA_e(Prob_e(A,B))$
9: **end for**

While considering the usage percentage of the majority class in our calculations is an interesting idea, things can dramatically go out of control. Let us consider a severe data imbalance problem where the data imbalance ratio is 1:7, this is a real example from our real defect database that we are working with (Class $D5$ = 70 samples: Class $D9$ = 10 samples). For this data imbalance ratio, to guarantee that the majority coverage percentage is 90%:

$$NBL = \frac{\log(0.1)}{10 \times \log(1-\frac{1}{70})} = \frac{-1}{-0.0625} = 16 \tag{3.64}$$

16-base learners are needed to guarantee that at least 90% of the majority class are used. Training more base learners is an expensive process. One of the alternative solutions is to use a mix of oversampling and undersampling. Instead of using one constraint, tow constraints are used at the same time: the minimum usage percentage $(F\%)$ and the maximum number of base learners NBL. We change the question to be: if at least $F\%$ of the majority class samples should be selected (at least once) and the total number of base learners should be NBL, then, how many samples of the majority class should be sampled each time?

$$NBL = \frac{log(1-F\%)}{|n| \times \log(1-\frac{1}{|Majority|})} \tag{3.65}$$

$$|n| = \frac{log(1-F\%)}{NBL \times \log(1-\frac{1}{|Majority|})} \tag{3.66}$$

At least $|n|$ samples need to be sampled from the majority class each time to guarantee the target usage percentage $F\%$. The actual number of samples to be used is directly related to the number of samples in the minority class:

$$Q = Max(|Minority|, |n|) \tag{3.67}$$

- If $|n| \leq |Minority|$, then $Q = |Minority|$. This results in usage percentage greater than the target. There is no need to perform oversampling on the minority class. Furthermore, the number of base learners can be reduced while satisfying the usage percentage.
- if $|n| > |Minority|$, then $Q = |n|$. In this case, oversampling is applied on the minority class.

In some severe cases of data imbalance problem and under the restrictions of using small number of base learners and large coverage percentage, the $|n|$ can be greater than the available number of samples in the minority class. Oversampling the minority class by just duplicating the samples may lead to overfitting problem. An alternative is to use a synthetic oversampling technique. In the next subsection, we will introduce the Variance-Based Adaptive SMOTE algorithm, VA-SMOTE, to synthetically oversample the minority class.

3.3.3 Variance-based Adaptive Synthetic Minority Technique (VA-SMOTE) Algorithm

In synthetic oversampling, instead of oversampling the minority class by duplicating samples, new samples are synthesized, such that one or more samples (original samples) are used to synthesize new samples. SMOTE [14] is an oversampling method that is used to create new minority samples by interpolating several minority-class instances that lie together. For each sample, one or more of the K-nearest neighbors $(KNNs)$ are selected. The original sample and its neighbors are used to create new samples. The new instance is generated by random interpolation as follows:

$$NS \; = \; OS \; + a \; \times \; distance(OS, \; KNN_s) \tag{3.68}$$

where NS is the new sample, OS is the original sample and KNN_s is one of the neighbors sample. a is a random number between 0 and 1, and $distance()$ is the distance between the two samples (the original sample and the KNNs sample). Even though this algorithm is very simple, it is proven in literature that it is very efficient. In fact, using synthetic oversampling causes the decision boundaries of the minority class to be spread further into the majority class space and thus avoiding the overfitting problem.

In this subsection, we will highlight some of the major drawbacks of SMOTE algorithm, more specifically we will address the problem of SMOTE algorithm when it is used with sever data imbalance problem and large within class variation. To explain the behavior of SMOTE algorithm under different conditions let us consider the classification problem depicted in Figure 17 (a).

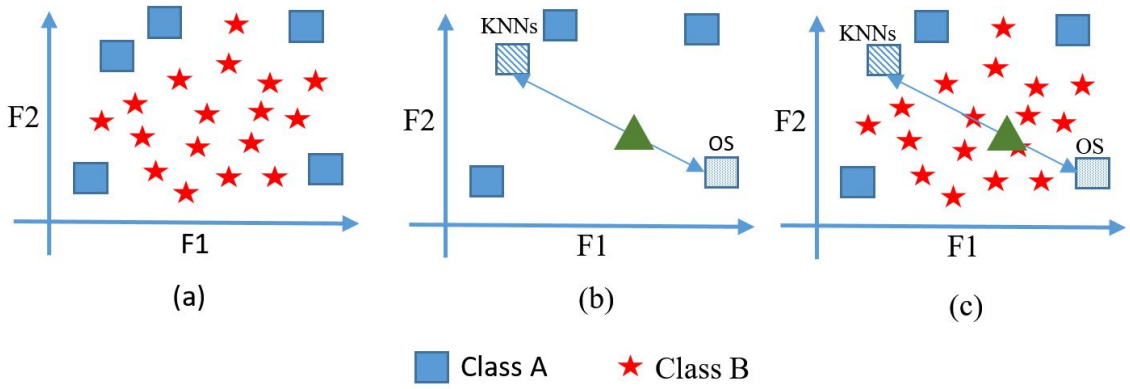

Figure 17: Applying SMOTE algorithm on severe data imbalance problem with large within minority class variation.

In this case, the ratio $\frac{|B|}{|A|}$ is quite large. This data imbalance problem is considered as severe data imbalance problem. The variation within the minority class A is large and the absolute number of samples belong to class A (the minority class) is small. If we use the 5-NN to synthesize new samples, as depicted in Figure 17 (b), the new sample (green triangle) can be anywhere along the line between the original sample OS and the KNN sample (KNN_s). Creating a new sample in this region of the 2-D space will affect the training negatively. By looking at the original data, we know that this region of the 2-D space is mainly related to the majority class. The new sample creates a new separate region (island), instead of extending the minority class region. In fact, training a base learner to classify these training examples may overfit the training data while trying to classify this isolated region correctly, Figure 17 (c).

In this work, we propose an alternative to the original SMOTE algorithm, where we consider both variation and number of samples in account. SMOTE algorithm includes specifying the KNN of the original sample. If the absolute number of minority samples is small with large variation within the class, there is no guarantee that these KNN samples are real near neighbors. In SMOTE algorithm, the random variable (a) can take a value between 0 and 1, $0 \leq a \leq 1$, thus the new sample can be anywhere on the line between the two samples: $NS = OS + a \times distance(OS, KNN_s)$. If $a = 0$, the new sample is exactly the same as the original sample (general oversampling). In fact, SMOTE algorithm is a generalization form of the general oversampling approach. In the modified version, VA-SMOTE, the random variable a can take a value between 0 and ω, where $0 \leq \omega \leq 1$. ω is given by:

$$\omega = 1 - G \tag{3.69}$$

$$G = \frac{1}{\sqrt[z]{|Minority|}} \tag{3.70}$$

where z is a positive number related to the variation within the minority class.

Larger variation results in larger z values and smaller variation results in smaller z values. If we fix z and increase the number of samples in the minority class $1 \rightarrow \infty$, the value of G goes from $1 \rightarrow 0$, thus the value of ω goes from $0 \rightarrow 1$. The more samples the larger ω, the less samples the smaller ω. When we have sufficient number of samples the modified version of the SMOTE algorithm converges to the original SMOTE algorithm $(0 \leq a \leq 1)$ and when we have very small number of samples and the variation within the minority class is large the modified version converges to the original oversampling algorithm $0 \leq a \leq 0 + \lambda,\ where\ \lambda\ is\ very\ small\ value$. But how to specify the value of z? What if the variation among one dimension is greater than the others? z can be given by:

$$z = (\frac{1}{d}) \sum_{i=1}^{d} std(V_i) \tag{3.71}$$

where V_i is a $1 \times |Minority|$ 1-D vector and d is the dimension of the data sample belongs to the minority class. V_i is constructed by copying the i^{th} value of each data sample.

z represents the average of standard deviations across each dimension. Even though calculating scalar z is a very simple way to get idea about the variation it may fail in some cases. Figure 18 depicts a very special case of data variation of two classes along two dimensions, $F1$ and $F2$.

The sample point of class B (lower left corner) represents the original sample and the five green data points represent the $5 - NN$ data samples. The scalar z value is specified as the average of the different standard deviations across different dimensions:

$$z = \frac{std(V_1) + std(V_2)}{2} \tag{3.72}$$

$std(V_1) < std(V_2)$, the variation across the first dimension is smaller than the variation across the second dimension.

The proposed BagStack classifier uses a combination of synthetic oversampling and undersampling. Having small number of samples in the minority class can make the undersampling process very challenging. Different samples may have different impacts on the decision boundary (e.g. SVM, only few samples affect and decide the decision boundary). Considering all the samples equal in the undersampling process may not be sufficient. But, how to define important samples (samples that have significant impact)?

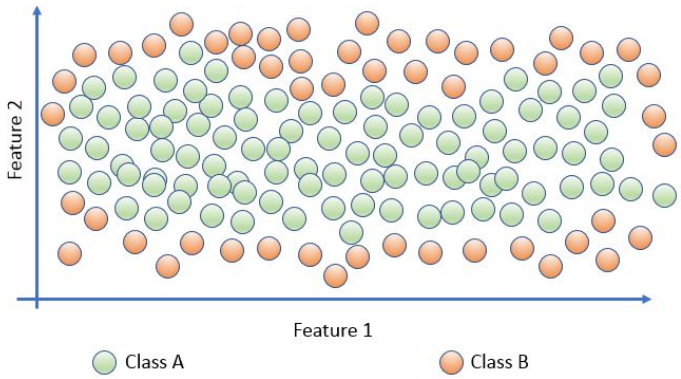

Figure 18: Data imbalance problem with different variations across different dimensions.

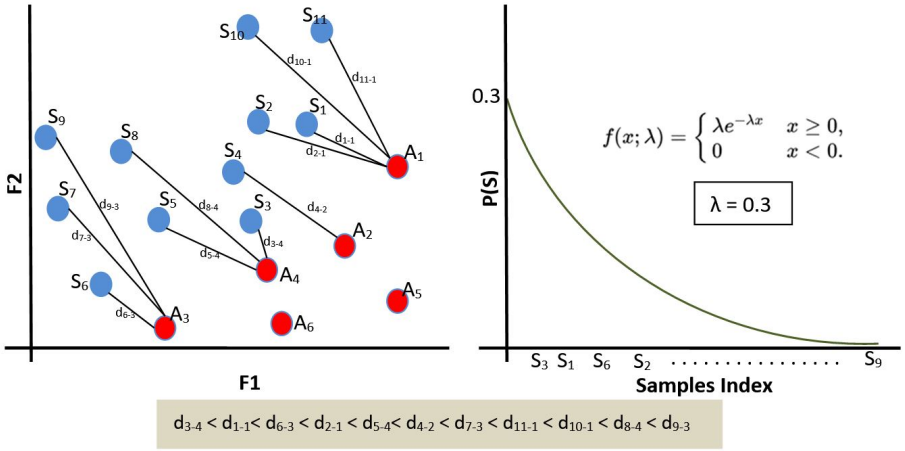

Figure 19: Adaptive undersampling of the majority class.

The proposed undersampling approach is motivated by the Border-Line SMOTE approach. The basic idea is to find the most difficult samples belonging to the majority class. Each sample $S_i \in Majority$ class is assigned a weight $W_d = min_v\ distance(S_i, R_v)$, where R_v is the set of minority samples. The samples with small W_d are more difficult samples. Then the samples are ordered based on the distance weights and exponential distribution is used to sample from the majority class. By using exponential distribution, the algorithm focuses more on the difficult samples, but also support diversity by considering other samples. Using the same concept in specifying the most difficult samples when applying the undersampling, we can select the most difficult samples from the minority class to be used for synthesized other samples. Figure 19 depicts the proposed adaptive undersampling technique.

Algorithm 9 Variation-Based SMOTE algorithm VA-SMOTE.

Minority-Class: the set of samples belonging to the minority class.
N: the number of samples to be synthesized.
|Minority|: the total number of samples in the minority class.
Step 1: Find the variation based parameter $\bar{Z}$:
for each dimension d: **do**
    counter=1
    for each sample S in the minority-class: **do**
        $V_d[Counter] = S[d]$
        counter=counter+1
    end for
    $Z[d] = std[V_d]$, where V_d is a vector constructed by copying values from minority samples across the d^th dimension.
end for
Step 2: Find ω:
$Z = \frac{\sum_{i=1}^{d} Z[i]}{d}$
$\omega = 1 - \frac{1}{\sqrt[Z]{|Minority|}}$
Step 3: Find the KNN of each sample in the minority class.
$NS_{set} = \phi$
Step 4: Synthesize new samples.
for each new sample i=1:N **do**
    Randomly, select one of the minority class samples. The original sample OS.
    Randomly, select one of the original sample OS nearest neighbour, KNN_S
    $a = random\ number\ (0 \leq a \leq \omega)$
    $NS = OS + a \times distance(OS, KNN_S)$
    Add the new sample NS to the synthesized sample set:
    $NS_{set} \leftarrow NS$
end for

3.3.4 Data Imbalance Problem at the Meta-data Level

If the learning data set is imbalanced and the standard cross-validation is used to generate the meta-data, the meta-data will also be imbalanced. Using imbalanced data (meta-data) set to train the meta-classifier may result in a biased meta-classifier. The simplest way to deal with this problem is to perform general oversampling/undersampling at the meta-data which may lead to over-fitting/loosing important information. In this thesis, we propose a new cross-validation procedure, imbalance-aware cross-validation. Figure 20 depicts the process of dividing the data into different sets, learning, testing, training and validation. Figure 21 depicts the standard cross-validation and Figure 22 depicts the class aware cross-validation.

In imbalance-aware cross-validation process, the set of all samples belonging to the same class

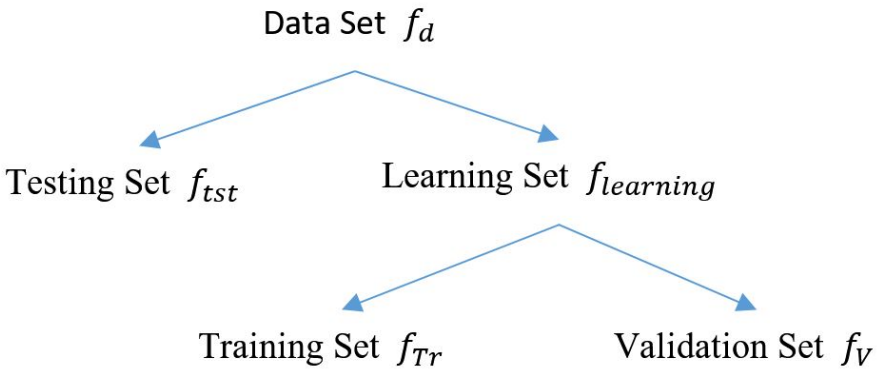

Figure 20: Dividing data into different sets.

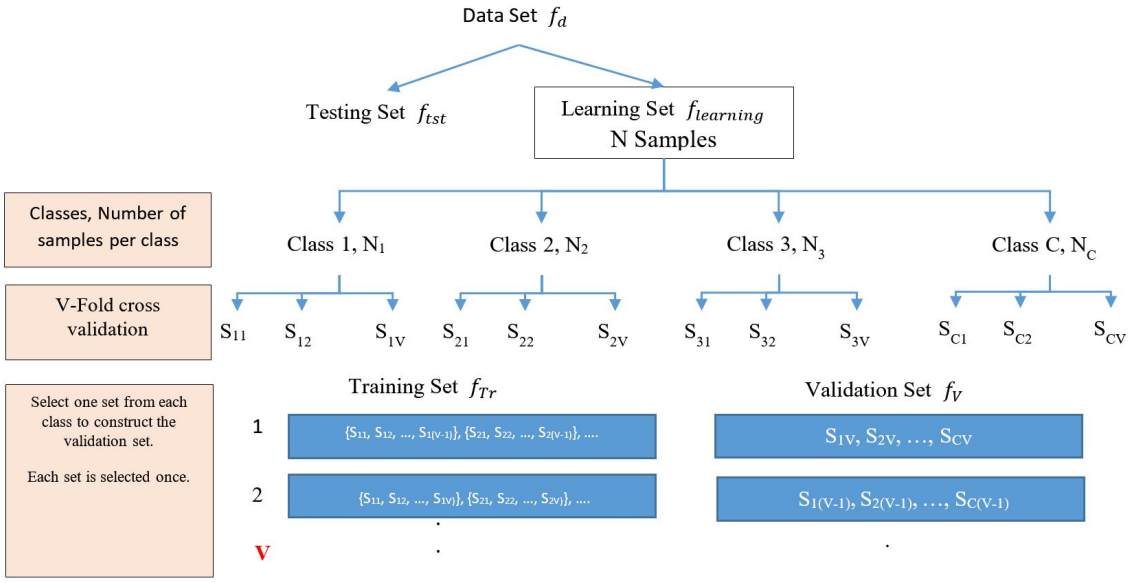

Figure 21: Standard cross-validation.

in the learning set is divided into M splits. Based on the number of samples belonging to the majority class and the number of samples belonging to the minority class, repeatedly, construct your training set and validation set by randomly sampling splits from different classes. Each time, one split is selected randomly from each class to construct the validation set and all other sets are used to construct the training set. Repeat the process for $\left\lceil \frac{|Majority|}{|Minority|} \right\rceil$ times. Each time, the training set is used to train the base learners and the validation set is used to generate the meta-data.

The next step is to perform undersampling on the meta-data by sampling from each class the

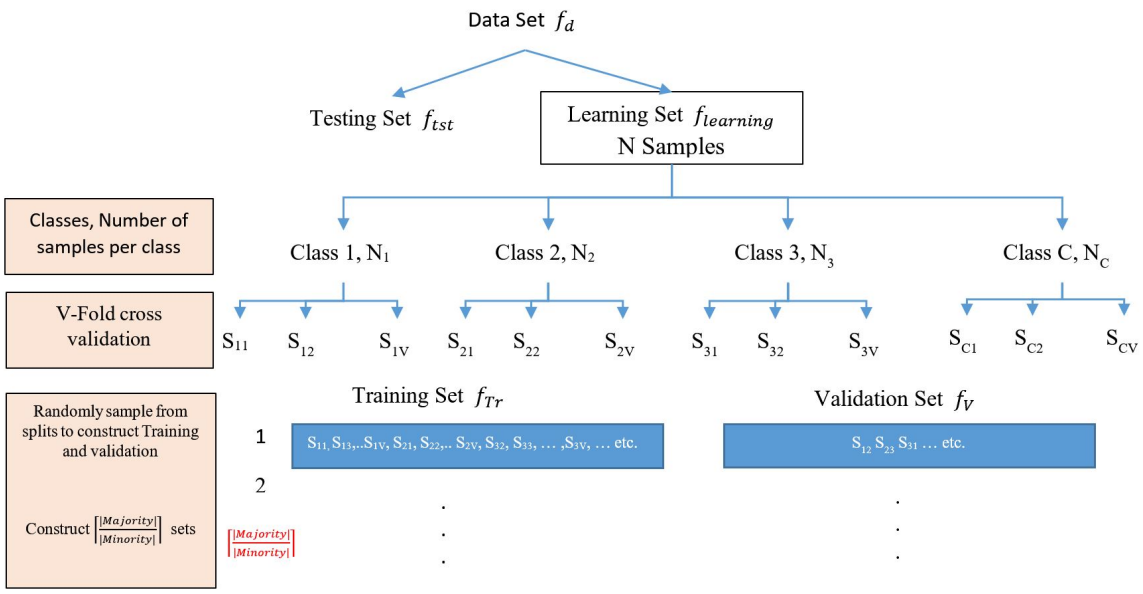

Figure 22: Imbalance-aware cross-validation.

number of samples belonging to the minority class. Even though there are multiple samples in the meta-data corresponding to the same data sample in the original learning set, the probability that these samples have been generated by using exactly the same training set is very low, since we randomly select different splits to construct the training set. These changes in the training set (different data samples) result in slightly different base learners and thus different predictions (changes in confidence). The main advantage of using this method over the general oversampling method, that the oversampling is applied at different domain (the input domain) and the samples are different at the output domain (meta-data) due to changes in the transformation (base learners). This will reduce the probability to have overfitting problems. Figure 22 depicts the class aware cross-validation process.

Testing the same sample multiple times using different base learners that have been trained using different training set results in multiple samples in the meta-data that correspond to the same data sample in the learning set. Because the base learners are not exactly the same, these samples will be slightly different.

3.4 Experimental Results

In this section, we present the experimental results to verify and justify the BagStack classifier. Different experiments were designed to show the effect of different parameters that control the behavior as well as the performance of the BagStack classifier. In subsection 3.4.1, we present the experimental setup, more than 100 data sets are used, balanced and imbalanced data sets with imbalance ratio varying between 1 and 40, binary and multi-class classification problems, data sets with small and large number of samples. More details are presented in subsection 3.4.1. Subsections 3.4.2 to 3.4.10 present the results of different experiments.

3.4.1 Experimental Setup

In this section, we present a description of the datasets that we used in our analysis. We present a full description of each one of the parameters that we are interested in. We use KEEL-dataset, a dataset repository that includes different datasets [131]. The KEEL-dataset repository provides a detailed categorization of different datasets and a description of their characteristics. The datasets are organized in several categories and sub-categories. The categories of the datasets are derived from the topics addressed in the experimental studies. Some of them are usually found in the literature, like supervised (classification) datasets, unsupervised and regression problems.

The classification datasets include all the supervised datasets. All these datasets contain one or more attributes which label the instances, mapping them into different classes. They distinguish four subcategories of classification datasets: Standard datasets, imbalanced datasets; imbalanced datasets are standard classification datasets where the class distribution is highly skewed among the classes, multi instance datasets; multi-instance datasets represent problems where there is a many-to-one relationship between feature vectors and its output attribute, and datasets with missing values. In our experiment, we use datasets from the standard and the imbalance dataset subcategories [132].

For different imbalanced datasets, they provide also processed versions where they use different synthetic oversampling techniques to oversample minority classes to achieve data balance. For these datasets, they divide the data into 5-folds, they apply synthetic oversampling only on the training part.

Table 1, Table 2 and Table 3 show the different datasets that are used in our analysis and their description. We include the name of the dataset, the number of classes, the number of samples in the minority class, the number of samples in the majority class, and the imbalance ratio. The imbalance ratio varies between 1 and 40. We divide the whole dataset into 5 categories based on the imbalance ratio. Later we will show how changing different parameters affect the performance on different data sets of different imbalance ratios.

In the following experiments, we address the effect of changing the parameters that control the behavior of the BagStack classifier. The number of training algorithms (3.4.2), the coverage percentage (3.4.3), using the number of base learners constraint (3.4.4), increasing the number of regression functions in the ensemble of the EB-MDTR meta-classifier (3.4.5), using different meta-classifiers (3.4.6), the imbalance-aware cross-validation (3.4.7), testing different types of meta-data processing methods (3.4.8). In the last two subsections, we present a performance comparison to other well-known ensemble classifiers (3.4.9) and justify the efficiency of using BagStack classifier on images datasets (3.4.10).

3.4.2 Increasing the Number of Training Algorithms

One of the main concepts behind BagStack classifier is to combine different algorithms that have different biases. Adding more training algorithms should provide more information that can be used by the meta-classifier. This information can be used to make a robust decision about the classification. In this experiment, we use 100 data sets that have different data imbalance ratios, the imbalance ratio varies between 1 and 40. All other parameters that control the behavior of the BagStack classifier are fixed. We turn off the imbalance-aware cross-validation and use the standard V-fold cross-validation with V=5. We use only the coverage percentage constraint to specify the number of base learners for each problem (coverage percentage=0.85). No pre-processing for the data at the input level or at the meta-level is used. As a meta-classifier, we use non-linear support vector machine with RBF kernel function. Table 4 shows the main training algorithms that are used as base learners as well as the main parameters that are used with each one of them. Table 5 shows the results of using different sets of training algorithms. We report the

Table 1: Different datasets and their descriptions: Imbalance Ratio 1-2.

Dataset Name	# Classes	Minority	Majority	IR	1/IR
iris	3	50	50	1	1
movement_libras	15	24	24	1	1
movement_libras	7	330	330	1	1
texture	11	500	500	1	1
vowel	11	90	90	1	1
twonorm	2	3697	3703	0.998	1.0
ring	2	3664	2736	0.9807	1.02
optdigits	10	554	572	0.9685	1.03
mammographic	2	403	427	0.9438	1.06
tae	3	49	52	0.94231	1.06
penbased	10	1055	1144	0.922	1.08
penbased	10	105	115	0.91304	1.10
vehicle	4	199	218	0.91284	1.10
monk-2	2	204	228	0.89474	1.12
sonar	2	97	111	0.87387	1.14
banana	2	2376	2924	0.81259	1.23
australian	3	307	383	0.80157	1.25
heart	2	120	150	0.8	1.25
bupa	2	145	200	0.725	1.38
wine	3	48	71	0.67606	1.48
wine	3	48	71	0.67606	1.48
led7digit	10	37	57	0.64912	1.54
wdbc	2	212	357	0.59384	1.68
bands	2	135	230	0.58696	1.70
ionosphere	2	126	225	0.56	1.79
glass1	2	76	138	0.55072	1.82
magic	2	6688	12332	0.54233	1.84
ecoli-0_vs_1	2	77	143	0.53846	1.86
wisconsin	2	239	444	0.53829	1.86
wisconsin	2	239	444	0.53829	1.86
pima	2	268	500	0.536	1.87
pima	2	268	500	0.536	1.87
contraceptive	3	333	629	0.52941	1.89
contraceptive	3	333	629	0.52941	1.89

percentage (number) of data sets that experienced increase in the overall accuracy and the average accuracy after adding one more training algorithm each time. These results are the average of 30 random splits, where 80% is used for learning (training and validation) and 20% is used for testing.

The most significant increase in the overall accuracy and the average accuracy over all data sets happens after adding the first few training algorithms. Adding more training algorithms increases the number of used base learners, this increment increases the dimension of the meta-data which increases the tendency of the meta-classifier to over-fit the meta-data for some data sets. In fact, for the last three to four training algorithms we added, we started experiencing reduction in the overall classification accuracy due to overfitting problem. The used meta-classifier in this experiment is support vector machine with non-linear kernel function (RBF). Table 6 depicts the average of the overall accuracies and the average of the average class-accuracies over the 100 datasets. These

Table 2: Different datasets and their descriptions: Imbalance Ratio 2-9.

Dataset Name	# Classes	Minority	Majority	IR	1/IR
iris0	2	50	100	0.5	2.0
glass0	2	70	144	0.48611	2.06
titanic	2	711	1490	0.47718	2.10
phoneme	2	1586	3818	0.4154	2.41
satimage	6	626	1533	0.40835	2.45
yeast1	2	429	1055	0.40664	2.46
marketing	9	505	1255	0.40239	2.49
haberman	2	81	225	0.36	2.78
haberman	2	81	225	0.36	2.78
vehicle1	2	217	629	0.34499	2.9
vehicle0	2	199	647	0.30757	3.25
ecoli1	2	77	259	0.2973	3.36
spectfheart	2	55	212	0.25943	3.85
appendicitis	2	21	85	0.24706	4.05
newthyroid	3	30	150	0.2	5.0
new-thyroid	3	30	150	0.2	5.0
new-thyroid1	2	35	180	0.1944	5.14
new-thyroid2	2	35	180	0.1944	5.14
hepatitis	2	13	67	0.19403	5.15
ecoli2	2	52	284	0.13831	5.46
dermatology	6	20	111	0.18018	5.55
dermatology	6	20	111	0.18018	5.55
balance	3	49	288	0.17017	5.88
balance	3	49	288	0.17017	5.88
segment0	2	329	1979	0.16625	6.02
glass6	2	29	185	0.15676	6.38
yeast3	2	163	1321	0.12339	8.10
glass	6	9	76	0.11842	8.44
glass	6	9	76	0.11842	8.44
ecoli3	2	35	301	0.11628	8.60
page-blocks0	2	559	4913	0.11378	8.79

results are the average of 30 random splits, where 80% is used for learning and 20% is used for testing.

3.4.3 Increasing the Coverage Percentage

In this experiment, we study the effect of increasing the coverage (usage) percentage. We know that increasing the coverage percentage increases the number of used base learners. Table 7 shows how changing the coverage percentage affects both overall and average class-accuracies. In Table 7, we present the number of data sets that experience increment in overall accuracy and average class-accuracy when increasing the coverage percentage. In this experiment, we fix all other parameters. We use three different training algorithms, svm, svm linear regression (LASSO) and leastsquares linear regression (lasso) as base learners. We turn off the imbalance-aware cross-

Table 3: Different datasets and their descriptions: Imbalance Ratio 9-40.

Dataset Name	# Classes	Minority	Majority	IR	1/IR
yeast-2_vs_4	2	51	463	0.11015	9.08
glass-0-1-5_vs_2	2	17	155	0.10968	9.12
yeast-0-2-5-6_vs_3-7-8-9	2	99	905	0.10939	9.14
yeast-0-2-5-7-9_vs_3-6-8	2	99	905	0.10939	9.14
ecoli-0-4-6_vs_5	2	20	183	0.10929	9.15
ecoli-0-1_vs_2-3-5	2	24	220	0.10909	9.17
ecoli-0-2-6-7_vs_3-5	2	22	202	0.10891	9.18
glass-0-4_vs_5	2	9	83	0.10843	9.22
ecoli-0-3-4-7_vs_5-6	2	25	232	0.10776	9.28
vowel0	2	90	898	0.10022	9.98
zoo	7	4	41	0.097561	10.25
ecoli-0-1-4-7_vs_2-3-5-6	2	29	307	0.094463	10.59
led7digit-0-2-4-5-6-7-8-9_vs_1	2	37	406	0.091133	10.97
ecoli-0-1_vs_5	2	20	220	0.090909	11.0
glass-0-1-4-6_vs_2	2	17	188	0.090426	11.06
glass2	2	17	197	0.086294	11.59
cleveland	5	13	160	0.08125	12.31
ecoli-0-1-4-6_vs_5	2	20	260	0.076923	13.0
cleveland-0_vs_4	2	13	164	0.079268	12.62
shuttle-c0-vs-c4	2	123	1706	0.072098	13.87
yeast-1_vs_7	2	30	429	0.06993	14.30
glass4	2	13	201	0.064677	15.46
coil2000	2	586	9236	0.063447	15.76
page-blocks-1-3_vs_4	2	28	444	0.063063	15.86
dermatology-6	2	20	338	0.059172	16.90
zoo-3	2	5	96	0.052083	19.20
shuttle-6_vs_2-3	2	10	220	0.045455	22.0
yeast-1-4-5-8_vs_7	2	30	663	0.045249	22.10
yeast4	2	51	1433	0.03559	28.10
winequality-red-4	2	53	1546	0.034282	29.17
poker-9_vs_7	2	8	236	0.033898	29.50
yeast-1-2-8-9_vs_7	2	30	917	0.032715	30.57
yeast5	2	44	1440	0.030556	32.73
ecoli-0-1-3-7_vs_2-6	2	7	274	0.025547	39.14
thyroid	3	17	666	0.025526	39.18

Table 4: Training Algorithms.

Training Algorithm	Parameters
Decision Tree	Maximum number of splits = 400.
SVM	Regression, LKF, Zscore normalization.
Linear regression	SVM, lasso, solver= sparsa, lambda = 0.1, No-Opt.
Linear regression	LS, lasso, solver= sparsa, lambda = 0.1, No-Opt.
Linear regression	SVM, ridge, lambda = 0.5, No-Opt.
Linear regression	LS, ridge, lambda = 0.5, No-Opt.
SVM	Classification, LKF, standardize = 1.
Random Forest	Classification, NBL = 5.
Regression-Bagging	Ensemble regression, NBL =10
Regression-Boosting	Ensemble regression, LSBoost, NBL =10

validation. We use nonlinear support vector machine with RBF kernel function as a meta-classifier. These results are the average of 30 random splits, where 80% is used for learning and 20% is used for testing.

Table 5: The total number of datasets that experienced increase in the overall accuracy / average accuracy when using more training algorithms.

Training Algorithms	Overall Accuracy	Average Accuracy
DT $\rightarrow$ DT+SVMR	89	85
+ LR, SVM, Lasso	74	70
+ LR, Leastsquares, Lasso	76	70
+ LR, SVM, Ridge	78	71
+ LR, Leastsquares, Ridge	86	81
+Classification, SVM	70	62
+ Classification, RF	64	56
+ Ensemble Regression, Bagging	78	67
+ Ensemble Regression, LSBoost	65	57

Table 6: The average of the overall-accuracy and the average accuracy over the 100 datasets when using more training algorithms.

Training Algorithms	$\overline{Overall\ Accuracy}$	$\overline{Average\ Class\ Accuracy}$
DT	81.6%	69.04%
+SVMR	86.18%	74.59%
+ LR, SVM, Lasso	85.16%	73.26%
+ LR, Leastsquares, Lasso	84.57%	72.41%
+ LR, SVM, Ridge	84.00%	71.66%
+ LR, Leastsquares, Ridge	85.11%	73.10%
+Classification, SVM	83.96%	71.36%
+ Classification, RF	79.02%	65.78%
+ Ensemble Regression, Bagging	77.67%	63.41%
+ Ensemble Regression, LSBoost	76.53%	60.87%

Table 7: The total number of datasets that experienced increase in the overall accuracy / average class-accuracy when increasing the coverage percentage.

Coverage Percentage	Overall Accuracy	Average Accuracy
$50\% \rightarrow 60\%$	77	69
$60\% \rightarrow 70\%$	81	74
$70\% \rightarrow 80\%$	84	75
$80\% \rightarrow 90\%$	82	74
$90\% \rightarrow 95\%$	80	72

Increasing the coverage percentage increases the number of base learners; adding more base-learners increases the tendency of the meta-classifier (SVM-RBF) to overfit the meta-data.

3.4.4 Using Different Constraints: Number of Base-learners

The total number of base learners to be used per each training algorithm is controlled by using one of the following constraints: the coverage percentage, the number of base learners or both of them. In this experiment, we test the performance of the BagStack classifier when using the number of base learners as constrain. In this experiment, we fix all other parameters. We use three different training algorithms, SVM, SVM linear regression (LASSO) and least squares linear

regression (LASSO) as base learners. We turn off the imbalance-aware cross-validation. Random Forest is used as a meta-classifier. The number of base learners used for each data set is a function of the number of base learners that is used with the corresponding data set when using only coverage percentage constraint of 85%. We consider this number of base learners as the base to achieve a coverage percentage of 85%. In each experiment, we reduce the number of base learners. $NBL_{DS} = NBL_{DS}(Coverage\ Percentage = 85\%) \times Reduction\ Factor$. $NBL_{DS}(Coverage\ Percentage = 85\%)$ is the number of base learners used to deal with the dataset DS when the using 85% coverage percentage. $Reduction\ Factor$ is a real number that takes a value between 0 and 1. Decreasing the number of base learners decreases the coverage percentage of the majority class in each dataset. In this experiment we tested different reduction factor, 1.0 (no reduction, uses the same number of base learners that used when the coverage percentage is 85%), 0.6 and 0.2. We run these experiments using only the number of base learners as constrain. Table 8 shows the number of data sets that experienced reduction in the overall accuracy and the average accuracy when changing the reduction factor. Increasing / decreasing the number of base learners affects the overall accuracy and the average accuracy for each dataset in a different way. In some cases, reducing the number of base learners improves the accuracy, the direct explanation for this is that the less number of base learners reduce the tendency of meta-classifier to overfit the meta-data.

Table 8: The total number of datasets that experienced reduction in the overall accuracy and average accuracy when changing the reduction factor.

Reduction Factor	Overall Accuracy	Average Accuracy
$1.0 \rightarrow 0.6$	69	56
$0.6 \rightarrow 0.2$	69	60

3.4.5 Increasing the Number of Regression Functions in the Ensemble Meta-classifier (EB-MDTR)

In the proposed classifier, we propose an ensemble-based multi-response DT regression meta-classifier EB-MDTR. In the standard MLR, a linear regression function is defined for each class. In the proposed meta-classifier, an ensemble of Decision Tree (DT) regression functions are defined

for each class. In this section, we address the effect of changing the number of DT regression functions when building each ensemble.

We fix all other parameters. We use ten different training algorithms (Table 4) as base learners. We turn off the imbalance-aware cross-validation and use the standard cross-validation with V=5 folds. Only the coverage percentage constraint is used (85%). Two versions of the proposed EB-MDTR are used. In the first version (EB-MDTR0), each ensemble linear regression model uses the predictions of all base learners to construct the meta-data instance. In the second version (EB-MDTR1), each class-related ensemble linear regression model uses the predictions of base learners that have been trained on samples belonging to the same class to construct the meta-data instance. These results are the average of 30 random splits, where 80% is used for learning and 20% is used for testing.

Table 9: The total number of datasets that experienced increase in the overall accuracy / average accuracy when changing the meta-classifier.

Meta-Classifier	**Overall Accuracy**	**Average Accuracy**
MLR (Optimize lambda) $\rightarrow$ EB-MDTR0(4)	70	81
EB-MDTR0(4) $\rightarrow$ EB-MDTR0(12)	74	71
EB-MDTR0(12)$\rightarrow$ EB-MDTR0(20)	58	51
MLR (Optimize lambda) $\rightarrow$ EB-MDTR0(4)	70	81
MLR (Optimize lambda) $\rightarrow$ EB-MDTR0(12)	81	85
MLR (Optimize lambda) $\rightarrow$ EB-MDTR0(20)	87	90
MLR (Optimize lambda) $\rightarrow$ EB-MDTR1(4)	70	81
EB-MDTR1(4)$\rightarrow$ EB-MDTR1(12)	82	74
EB-MDTR1(12)$\rightarrow$ EB-MDTR1(20)	77	72
MLR (Optimize lambda) $\rightarrow$ EB-MDTR1(4)	69	81
MLR (Optimize lambda)$\rightarrow$ EB-MDTR1(12)	82	87
MLR (Optimize lambda) $\rightarrow$EB-MDTR1(20)	83	87

Table 10: The average of overall accuracies and average accuracies over 100 data sets when using different meta-classifiers.

Meta-Classifier	**Overall Accuracy**	**Average Accuracy**
MLR (Optimize lambda)	83.88%	67.77%
EB-MDTR0(4)	84.62%	73.51%
EB-MDTR0(12)	86.36%	75.55%
EB-MDTR0(20)	86.42%	75.41%
MLR (Optimize lambda)	83.90%	67.01%
EB-MDTR1(4)	84.36%	73.83%
EB-MDTR1(12)	85.93%	75.42%
EB-MDTR1(20)	86.62%	75.45%

Table 9 depicts the number of datasets that experienced increase in the overall accuracy / average accuracy. By switching from the standard MLR to the proposed EB-MDTR meta-classifier,

70 datasets experienced increase in the overall accuracy and 81 datasets experienced increase in the average accuracy. This was the case for both EB-MDTR0 and EB-MDTR1. Increasing the number of DT regression functions in the ensemble of each class increases the number of datasets that experienced increase in both overall and average accuracies. This increase was more significant when using the 4 DT regression functions and 12 DT regression functions. For the maximum number of DT regression functions in the ensemble (20 DT ensemble functions), 87 datasets experienced increase in the overall accuracy when using EB-MDTR0 and 83 datasets when using EB-MDTR1. In the other hand, 90 datasets experienced increase in the average accuracy when using EB-MDTR0 and 87 when using EB-MDTR1.

Table 10 depicts the average of overall accuracies and the average of average accuracies overall the 100 datasets when using different meta-classifiers. Three important notes. **First**, the results for both EB-MDTR1 and EB-MDTR0 are comparable. Both meta-classifiers can handle the overfitting tendency due to the increase in the dimension of the meta-data. Bagging-Based MDTR is able to handle overfitting problem and variance error. This reduces the demand for knowledge and experience when using the BagStack classifier. **Second**, it is well known that adding more base learners to the bagging ensemble reduces the variance error, but eventually will converge to a minimum value. In our experiment, we notice that increasing the number of the MDTR functions reduces the generalization error and improve the classification accuracy (both overall and average accuracies). Since the increase in performance when increasing the number of base learners from 12 to 20 is not really significant, we think that using only 20 MDTR functions to build each ensemble is reasonable. In reality, adding more functions may still reduce variance error, but we do not think that going above 25-30 functions will add any value. There will be always tradeoff between the number of MDTR regression functions that construct the ensemble, the computational complexity and the gain (reduction in generalization error GE) that can be achieved. **Third**,the difference in the overall accuracy when using MLR and EB-MDTR is around 2.5, while the difference in the average accuracy is around 8. This big difference in improvement can be explained as follow. Imbalanced datasets introduce biased errors. It is well known that biased errors cannot be reduced by using bagging, but changing the regression function is a major contributor in reducing the sensitivity to imbalance datasets and generating biased classifiers. In fact, linear regression functions that are used in the MLR meta-classifier are trained by minimizing the least square error under a regularization constraint (lasso regularization). This minimization problem has higher tendency to generate biased

regressor functions when the data is imbalanced. However, DT regression functions are non-linear and are built by using binary split trees where at each node a regression-based splitting rule is used. These regression decision trees have lower tendency to generate biased classifiers when the data is imbalanced.

3.4.6 Using Different Meta-classifiers

In this experiment, we study the performance of changing the meta-classifier. We have tested 9 different meta-classifiers. SVM, MLR, EB-MDTR, RF [133], bagging, RUSBoost [134], Subspace-Boosting [135], TotalBoost [136] and LPBoost [137]. For bagging and Boosting, we use cross-validation-based optimization to optimize the number of learners (Decision Tree) and the maximum number of splits. Table 11 presents the number of times each meta-classifier outperformed all the others (slightly outperformed with $\mp 2\%$ of the best performing, according to randomly splitting the datasets). We turn off the imbalance-aware cross-validation and use the standard V-fold cross-validation with V=5. We use only the coverage percentage constraint to specify the number of base learners for each problem (coverage percentage=0.85). No pre-processing for the data at the input level or at the meta-level is used. These results are the average of 30 random splits, where 80% is used for learning and 20% is used for testing.

Both MLR and the proposed EB-MDTR have been tested in two different modes, in the first one, all predictions that are generated by the base classifiers are concatenated to construct the meta-data instance (MLR0, EB-MDTR0), in the second one, for each class-related linear regression function (ensemble), the corresponding predictions by base learners that include the same class are concatenated to construct the meta-data instance (MLR1, EB-MDTR1) (similar to the same concept used by StackC). 25 linear regression functions are used to build one ensemble linear regression function corresponding to each class. These results are the average of 30 random splits, where 80% is used for learning and 20% is used for testing.

Three major points are concluded. RF, Bagging, EB-MDTR0 and EB-MDTR1 are very comparable. In specific, RF performance is very comparable to using EB-MDTR. RF wins more data sets than EB-MDTR1 but less than EB-MDTR0. SVM (RBF) and other boosting techniques show higher tendency to overfit the meta-data. Having bad base learners (base learners that use noisy training

Table 11: Performance comparison of using different meta-classifiers.

Meta-Classifier	Winning Times	Average overall accuracy
SVM-RBF	36	76.06%
RF	77	87.82%
Bagging	74	87.27%
Subspace	20	69.9%
TotalBoost	47	84.4%
LPBoost	23	81.95%
MLR0	29	83.88%
MLR1	37	83.90%
EB-MDTR0	78	86.42%
EB-MDTR1	74	86.62%

data or unhelpful training algorithms) may affect these meta-classifiers and thus lead to overfitting problem. The proposed Multi-response linear decision tree regression proofs its ability to resist overfitting. In fact, the result that EB-MDTR0 wins more datasets than EB-MDTR1 contradicts the results published in StackC. In StackC, the authors argue that using predictions from base learners that have been trained on classes that include class of interest (the class that we are generating the linear regression function to) reduces the tendency of the linear regression to overfit the data.

It is worth mentioning the correlation between the Random Forest and the proposed Bagging-based MDTR. In RF, a set of decision trees is trained. Each decision tree is trained on a subset of the training data that is randomly generated by Bootstrap sampling from the original data set (learning set, Meta-data set). When training the decision tree, at each node a random subset of the features (attributes) is used to make the split decision. This is correlated to what is happening when building the Bagging-based MDTR. In EB-MDTR, we build an ensemble of regression decision tree functions that are trained on different subsets of the training data (learning data) using bootstrap sampling.

3.4.7 Using Imbalance-aware Cross-validation to Deal with the Data Imbalance Problem at the Meta-level

Dealing with the data imbalance problem at the input level by using the concept of bag-stack classifier improves the quality of the base learners and the quality of the meta-data. Using imbalance learning data results into imbalance meta-data. Having imbalance meta-data increases the tendency of the meta-classifier to be biased toward the majority class. Three different solutions can be applied; (1) Processing the learning set to deal with the data imbalance problem by us-

ing either oversampling or synthetic oversampling; (2) processing the meta-data to deal with the data imbalance problem by using oversampling or synthetic oversampling and (3) using imbalance-aware cross-validation to deal with the data imbalance problem. The main difference between the 1^{st} and the 2^{nd} options is by applying the oversampling technique at the input space or at the meta-data space. In the last approach, several versions of the transformation functions (base-learners) are used to generate several meta-data instances by using the same input instance followed by undersampling. While this section main concern is to show and verify the impact of imbalance-aware cross-validation on the average-class accuracy, the next two sub-sections (3.6.8 and 3.6.9) focus on the first two solutions, comparing imbalance-aware cross-validation to other oversampling techniques that are applied at the meta-data space (3.6.8) and the input space (3.6.9).

In this experiment, we test the ability of the imbalance-aware cross-validation to deal with the data imbalance problem using 100 data sets. We use three different training algorithms, svm, svm linear regression (lasso) and leastsquares linear regression (lasso) as base learners. For each data set, we run the experiment twice, once using the standard cross-validation technique with V-Fold equal 5 and other when using the imbalance-aware cross-validation. It is well known that different training algorithms have different sensitivity levels for data imbalance problem. In order to make sure that the results are independent of the meta-classifier, we repeat the whole experiment using different meta-classifiers. In this experiment, we use eight different meta-classifiers: SVM, Random Forest, Ensemble Bagging, MLR, EB-MDTR, LPBoost, Subspace-Boosting and TotalBoost. Both MLR and the proposed EB-MDTR have been tested in two different modes, in the first one, all predictions that are generated by the base classifiers are concatenated to construct the meta-data instance (MLR0, EB-MDTR0), in the second one, for each class-related linear regression function (ensemble), the corresponding predictions by base learners that include the same class are concatenated to construct the meta-data instance (MLR1, EB-MDTR1) (similar to the same concept used by StackC). Decision tree is used as the base learner in all ensemble meta-classifiers except the Subspace where KNN is used. cross-validation is used to optimize the number of base learners and the maximum number of splits in all ensemble classifiers.

Table 12 depicts the number (percentage) of datasets that experienced increase in the overall and average accuracies when using imbalance-aware cross-validation. These results are the average of 30 random splits, where 80% is used for learning and 20% is used for testing.

Table 12: The number of datasets that experienced increase in the overall, average and both accuracies when using imbalance-aware cross-validation.

Meta-Classifier	IR 1-2		IR 2-5		IR 5-40		Total	
	OA	AA	OA	AA	OA	AA	OA	AA
SVM	42.8%	54.3%	33.3%	93.3%	30.0%	90.0%	35.0%	78.0%
RF	48.6%	62.9%	20.0%	80.0%	10.0%	90.0%	25.0%	79.0%
Bagging	42.9%	60.0%	13.3%	93.3%	16.0%	88.0%	25.0%	79.0%
Subspace	57.1%	68.6%	33.3%	100%	34.0%	94.0%	42.0%	86.0%
TotalBoost	42.9%	57.1%	33.3%	86.7%	26.0%	78.0%	33.0%	72.0%
LPBoost	54.3%	60.0%	46.7%	80.0%	22.0%	90.0%	37.0%	78.0%
MLR0	37.1%	45.7%	20.0%	86.7%	32.0%	86.0%	32.0%	72.0%
MLR1	51.4%	65.7%	60.0%	100%	34.0%	88.0%	44.0%	82.0%
EB-MDTR0	54.3%	65.7%	20.0%	86.7%	18.0%	94.0%	31.0%	84.0%
EB-MDTR1	54.3%	60.0%	20.0%	93.3%	16.0%	92.0%	30.0%	81.0%

Table 12 is divided into different categories according to different imbalance ratios. This experiment was designed to show only how using imbalance-aware cross-validation can improve the average accuracy (AA) and not to compare different meta-classifiers. The main purpose of trying different meta-classifiers is to show that the proposed technique has a consistent effect when using different meta-classifiers. In general, we experienced increase in average accuracy when using imbalance-aware cross-validation comparing to using the standard cross-validation. In fact, using imbalance-aware cross-validation followed by undersampling at the meta-data level improves the average accuracy. This improvement in the average accuracy may affect the overall accuracy negatively, the higher the imbalance ratio the higher the improvement in the average accuracy and the less improvement (negative impact) in the overall accuracy.

3.4.8 Testing Different Meta-data Processing Methods

Dealing with the data imbalance problem at the meta-data level can be done by applying over sampling (synthetic oversampling) at the meta-data. In this experiment, we compare the performance of using imbalance-aware cross-validation to deal with data imbalance problem to other oversampling techniques that can be used to synthesize meta-data instances. 100 data sets are used. We report the number of data sets on which each of the processing methods outperformed all others (or slightly outperformed by $\mp 2\%$ of the highest performing) and the average of average accuracies over 100 datasets. We use three different training algorithms, SVM, SVM regression (lasso) and least squares regression (lasso) as base learners. We use the coverage percentage (85%) constraint to control the number of base-learners.

Since different meta-classifiers have different sensitivity to data imbalance problem, we test different meta-classifiers. Nonlinear support vector machine with RBF kernel function, Random Forest, ensemble bagging, MLR0, MLR1, EBMDTR0 and EBMDTR1 are used as meta-classifiers. These results are the average of 30 random splits, where 80% is used for learning and 20% is used for testing. Imbalance-aware cross-validation achieves the highest average accuracy among the 100 datasets and wins (slightly wins within $\mp 2\%$) the highest number of datasets. Random forest and EBMDTR1 achieves the highest average accuracy.

Table 13: The average of average accuracies over 100 datasets and the number of datasets each meta-data processing methods win when using different meta-classifiers.

Classifier	Metric	Meta-Data Processing Method					
		SMOTE	BL-SMOTE	VASMOTE	ROS	RUS	IACV+RUS
RF	**AA**	79.3	78.4	79.08	78.1	81.4	81.9
	DS	18	13	18	10	21	53
SVM (RBF)	**AA**	73.9	73.0	73.7	74.	76.3	79.7
	DS	17	14	20	23	23	66
Bagging	**AA**	78.3	77.8	78.3	77.4	80.8	81.2
	DS	11	14	11	11	30	56
MLR0	**AA**	66.8	65.4	66.9	66.5	66.2	70.6
	DS	16	7	7	12	16	52
MLR1	**AA**	65.9	64.5	66.5	66.0	66.5	74.6
	DS	11	7	14	11	12	55
EBMDTR0	**AA**	70.1	68.3	68.8	68.6	71.1	79.3
	DS	8	12	7	10	16	64
EBMDTR1	**AA**	68.6	67.8	68.8	68.3	70.7	80.9
	DS	14	7	11	8	12	66

3.4.9 Performance Comparison to Other Well-known Ensemble Classifiers

BagStack classifier was designed to deal with data imbalance problem when different training algorithms (base learners) of different biases are combined. Instead of using bagging or boosting, BagStack classifier uses the concepts of bagging and stacking to train and combine several training algorithms. BagStack classifier addresses the data imbalance problem at different levels. In literature, many other ensemble classifiers were proposed to deal with the data imbalance problem. In this experiment, we compare the performance of our classifier to other ensemble classifiers that have been designed mainly to deal with data imbalance problem or to other ensemble classifiers that uses a pre-processed data sets using one of the most common synthetic oversampling methods: ENN-SMOTE, SMOTE and TomeLinks algorithms.

Forty-six data sets were used in this experiment. Each data set is divided into five folds. Each

time four folds are combined to construct the training set and one-fold is used for testing. Two versions of these five folds are exists. The first one is the original data set, as described above, four folds are combined to construct the training set and one-fold is used for testing. The second version is the processed one. In this version the training set is processed using one of the aforementioned synthetic oversampling techniques. The training set is balanced by synthesizing samples from minority classes. The testing set is kept as it's without any processing. It is very important to highlight that using synthesized samples in the testing process is not legitimate. Synthesized samples can only be used in the training, the testing process should be done on original samples. This is the main reason, why we divide the data form the beginning into several folds and into training/testing sets before applying any synthetic oversampling technique.

Different classifiers and ensemble classifiers are used. Linear Support Vector Machine, None-Linear Support Vector Machine with radial Basis Function, Decision Tree, Random Forest, AdaBoost (M1, M2), Bagging, Random undersampling boosting RUSBoost, Subspace-Boost, TotalBoost, LPBoost. In all ensemble classifiers except subspace we use decision tree as the base learner, for subspace we use KNN. We optimize both of the number of base learners and the maximum number of splits. These classifiers were tested using the processed (balanced) datasets. We test each classifier using three different versions of the dataset, one that was processed by using ENN-SMOTE algorithm, another by using SMOTE algorithm and the last one is by using TomeLinks algorithm.

On the other hand, we use BagStack classifier in six different configurations, we change both of types of base learners and the type of meta-classifier. In one configuration, we use three different training algorithms, SVM, SVM regression (LASSO) and least squares regression (LASSO) as base learners. In another configuration, we use 10 different training algorithms. We use three different meta-classifier, Random forest where we optimize the number of decision tress, Ensemble-Based Multi-Response Decision Tree Regression with concatenating the outputs of all base learners to construct a meta-data instance (EBMDTR0) and Ensemble-Based Multi-Response Decision Tree Regression with concatenating the outputs of base learners that use the target class (the class that we are training the regression function to predict) as one of the two classes that are used to train the base learner (EBMDTR1).

Table 14 shows the comparison of different methods in terms of overall classification accuracy. Instead of reporting the average of overall accuracies over all datasets (the forty-six data sets),

we report the number of data sets on which each one of the classifiers outperformed all others (or slightly outperformed ∓ 1 of the best performance). Since our classifier has been designed mainly to deal with data imbalance problem, it has been designed to maximize and focus more on the average accuracy and not the overall accuracy, the BagStack classifier was able to win a significant number of datasets, the main competitor is the Random Forest classifier. This is highly expected, according to well-known studies [138] that address a full comparison between large number of classifiers on large number of datasets, Random Forest and Support Vector Machine (RBF) represent ones of the best classifiers that one can start with. Random Forest is a Bagging-Based Ensemble classifier that is known to resist overfitting the training data. BagStack classifier shows high performance when more and different training algorithms are used as base learners. This gives the classifier the ability to analyze more complex relations that are built based on different biases induced by using different training algorithms.

Table 14: Comparison of different methods in terms of overall classification accuracy.

Classifier	**ENN**	**SMOTE**	**TomeLinks**	**IACV**
SVM-Linear	7	8	5	-
SVM-Nonlinear RBF	15	15	13	-
Decision Tree	4	2	4	-
Random Forest	15	23	12	-
AdaBoost (M1, M2)	10	18	8	-
Bagging	14	21	12	-
RUSBoost	0	5	1	-
Subspace-Boost	7	7	7	-
TotalBoost	2	7	1	-
LPBoost	3	9	10	-
BagStack (RF-3BL)	6	6	4	10
BagStack (EBMDTR0-3BL)	0	0	0	2
BagStack (EBMDTR1-3BL)	0	0	0	2
BagStack (RF-10BL)	15	20	16	19
BagStack (EBMDTR0-10BL)	9	9	7	18
BagStack (EBMDTR1-10BL)	9	9	7	18

Table 15 shows the comparison of different methods in terms of average accuracy. Similar to overall accuracy, instead of reporting the average of the average accuracies over all datasets, we report the number of datasets on which each classifier outperformed the other classifiers (slightly outperformed the other classifiers, ∓ 1 of the best performance). BagStack classifier outperformed all other classifiers achieving the highest number of data sets. In fact, this table shows two important aspects regarding the performance of the BagStack classifier, the first one is the supreme of Imbalance-Aware cross-validation over all other synthetic oversampling techniques when using BagStack Classifier. This demonstrates that the quality of the synthesized meta-data instances

by using different versions of base learners using slightly different training data is higher than the quality of transforming multiple instances that have been synthesized in the input space and transformed using the same set of base learners. The other aspect is that BagStack classifier was able to beat all other classifiers and outperformed all of them on the highest number of datasets.

Table 16 shows the comparison of different classifiers in terms of overall and average accuracies. In this table, we report the number of data sets on which each one of the classifiers outperformed (slightly outperformed within ∓ 1 of the best performance) all other classifiers, we consider both of overall and average-class accuracy. This table shows which classifier is better in handling the tradeoff between maximizing the overall accuracy and maximizing the average-class accuracy. The BagStack classifier was able to beat all other classifiers.

Table 15: Comparison of different methods in terms of average accuracy.

Classifier	**ENN**	**SMOTE**	**TomeLinks**	**IACV**
SVM-Linear	10	11	8	-
SVM-Nonlinear RBF	5	3	6	-
Decision Tree	4	2	2	-
Random Forest	10	7	7	-
AdaBoost (M1, M2)	8	5	8	-
Bagging	11	8	7	-
RUSBoost	2	2	1	-
Subspace-Boost	0	0	0	-
TotalBoost	2	2	2	-
LPBoost	5	3	6	-
BagStack (RF-3BL)	5	3	3	8
BagStack (EBMDTR0-3BL)	0	0	0	2
BagStack (EBMDTR1-3BL)	0	1	0	2
BagStack (RF-10BL)	11	7	5	18
BagStack (EBMDTR0-10BL)	2	4	6	20
BagStack (EBMDTR1-10BL)	2	4	6	21

3.4.10 BagStack Classifier on Images Datasets

In order to justify using the BagStack classifier on images datasets, we tested the BagStack classifier on five different datasets. The process of images classification can be divided into two main steps: Features extraction (image representation) and features classification. For the last two decades, many images-related features have been invented and have been used successfully in image-classification tasks, Scale Invariant Feature Transform (SIFT) [139], Speed Up Robust Features (SURF) [140], Histogram of Gradients (HOG) [141], Shape Context Descriptor [142], Self-

Table 16: Comparison of different methods in terms of overall classification accuracy and average accuracy.

Classifier	ENN	SMOTE	TomeLinks	IACV
SVM-Linear	17	19	13	-
SVM-Nonlinear RBF	20	18	19	-
Decision Tree	8	4	6	-
Random Forest	25	30	19	-
AdaBoost (M1, M2)	18	23	16	-
Bagging	25	29	19	-
RUSBoost	2	7	2	-
Subspace-Boost	7	7	7	-
TotalBoost	4	9	3	-
LPBoost	8	12	16	-
BagStack (RF-3BL)	11	9	7	18
BagStack (EBMDTR0-3BL)	0	0	0	4
BagStack (EBMDTR1-3BL)	0	1	0	4
BagStack (RF-10BL)	26	27	21	37
BagStack (EBMDTR0-10BL)	11	13	13	38
BagStack (EBMDTR1-10BL)	11	13	13	39

Similarity Descriptor SSIM [143], Gist [144], Local Binary Pattern LBP [145] and Geometric Blur [146] among the others. These features are hand engineered features. In other words, to extract any of these features there is a set of operations (steps) that should be done. Many of them are local features which decodes a local region in the image that may contain few number of pixels into a feature vector. Features extracted from the same image can be combined together using Bag of Visual Words (BoW) [147] or sparse coding [148] scheme to construct a single, rich and efficient representation of the image. This representation is used to train a classification model (e.g. BagStack classification model).

During the last few years, the trend in image classification has been shifted away from using hand-engineering features toward learning these features by using training data. In other words, the features extraction process is not considered anymore as a separate process. Modern classification models (DNN models, e.g. Convolution Neural Networks [149, 150, 151, 152, 153, 154, 155], Deep Belief Networks [156] … etc.) takes images as inputs (2-D, 3-D images). The features extraction process is embedded in the model itself and it is done internally as part of the classification process. The extracted features represented by filters coefficients are task-dependent.

Convolution Neural Networks demand hundreds of thousands of images to achieve high classification accuracy. This number of images may not be available for many different types of image-classification problems. DNN models that have been trained using large number of images from one domain can be used to extract features from other images in another domain. In fact, it is well-known that in CNN the first few layers perform features extraction. These layers change the

representation of the image into another representation that is used as input for later layers. The network can be divided into two major parts, the convolution layers and the fully connected layers. Convolution layers extracted features from the image, the first few fully connected layers process the output of the last convolution layer to generate a rich representation of the input image. This rich representation is used as input for the last few fully connected layers that learn how to classify these representations into different classes. The outputs of the first few fully connected layers can be used as features. These features can be used with other classifiers, e.g. SVM, Random Forest, and BagStack Classifier.

In this section, we will use two different types of features: SIFT and DNN-Based Features. SIFT features are extracted, sparse coded (Dictionary Size 2000, Lambda = 0.2) and combined using max-pooling to come up with a single rich representation of the image. Two DNN models (AlexNet [149] and VGG-16 [150]) are used. The outputs of the fully connected layers 17 and 19 of AlexNet [149], 33 and 35 of VGG-16 [150] are used as features resulting in 4-DNN based Features and in total 5 features.

Five different images datasets are used, Birds [157], Butterflies [158], Scene-15 [159], Surface-Defect Dataset (NEUSDD) [160] and STL-10 [161]. Birds and Butterflies datasets are small datasets of 6 and 7 classes respectively. Scene-15 datasets contains 15 different categories (bedroom, coast, highway … etc.), Surface-Defect dataset contains 6 different types of surface defects, these defects are captured from jet engine blades.

The STL-10 dataset is inspired by the CIFAR-10 dataset but with some modifications. Each class has fewer labeled training examples than in CIFAR-10. SLT-10 consists of three different parts, training, testing and unlabeled data. Originally, the data set is designed to take advantage of the unlabeled data through unsupervised learning. Since this is not the case for BagStack classifier, we do not use the unlabeled data, we use only the training and the testing sets. Figure 23 shows sample images from each one of these datasets.

The BagStack classifier was used, two different types of base learners, Linear Support Vector Machine and Random Forest Classifier ($N = 5$). We use the BagStack classifier in two different configurations. In the first configuration, we use only the coverage percentage constraint to decide the total number of base learners (Coverage Percentage = 90%), we turn off the imbalance-aware cross-validation and use Random Forest as a meta-classifier. In the second configuration, we use

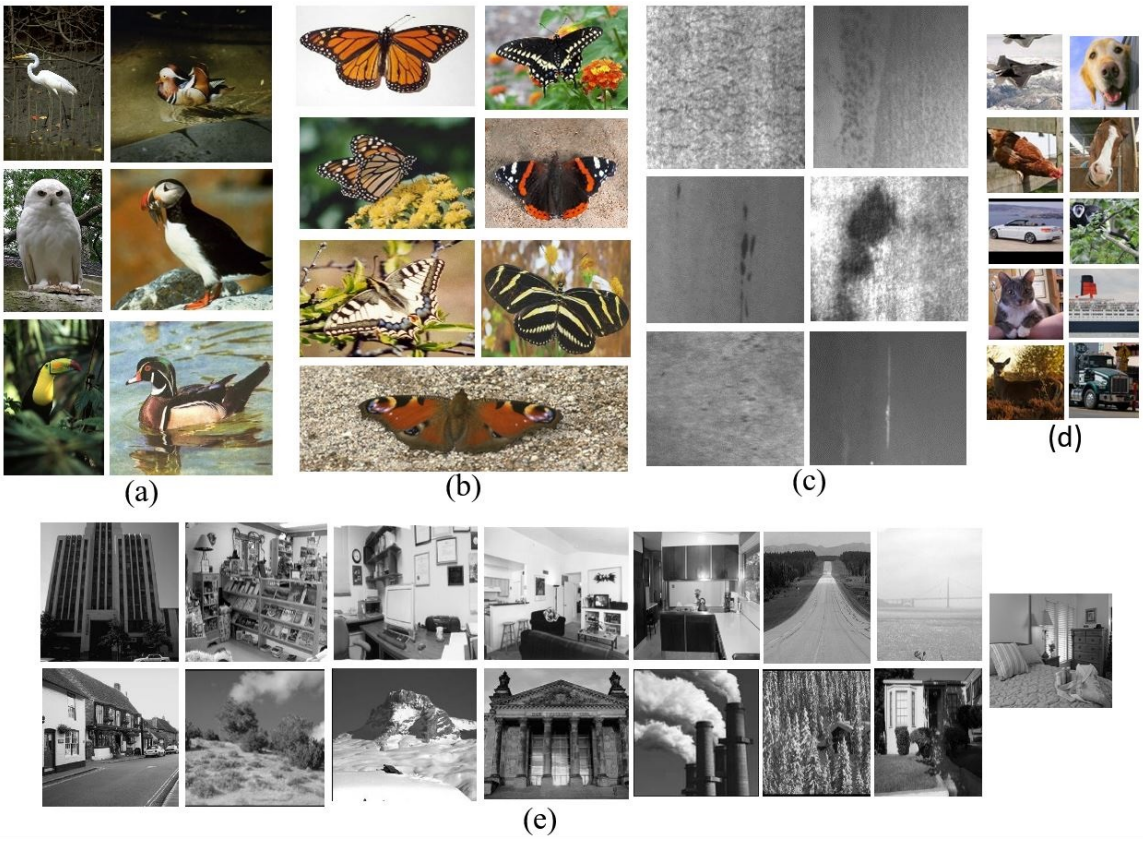

Figure 23: Sample images of different datasets. (a) Birds dataset (6-classes), (b) butterflies dataset (7-classes), (c) Surface Defects (6-classes), (d) SLT-10 (10 classes) and (e) Scene-15 (15-classes).

only the coverage percentage constraint to decide the total number of base learners (Coverage Percentage = 80%), we turn on the imbalance-aware cross-validation and use Random Forest as a meta-classifier.

Table 17 shows the overall classification accuracy, the average accuracy and the standard deviation for 20 random splits. Each time 80% of the dataset is used for training and 20% of the dataset is sued for testing. Using imbalance-aware cross-validation improves both of overall accuracy and average accuracy and reduces standard deviation (variance error).

Table 17: The overall classification accuracy and average accuracy on different images datasets.

Dataset	IACV = OFF				IACV = ON			
	OA	STD	AA	STD	OA	STD	AA	STD
Birds	97.92%	±1.7	97.92%	±1.7	97.92%	±0.5	97.92%	±0.5
Butterflies	95.97%	±1.1	95.6%	±1.3	96.37%	±0.5	96.65%	±0.7
Surface-Defects	100%	±0.0	92.87%	±0.9	93.1%	±0.2	93.44%	±0.2
SLT-10	92.8%	±0.9	92.87%	±0.9	93.1%	±0.2	93.44%	±0.2
Scene-15	92.9%	±0.7	92.76%	±0.7	93.26%	±0.5	93.47%	±0.5

3.4.11 Conclusion

In this chapter,the BagStack classifier is introduced, a classifier that uses both bagging and stacking concepts to deal with data imbalance problems when different training algorithms of different biases are being used.

We first explained the rational behind stacking based classifiers and the rational behind bagging classifiers. A stacking based classifier is used to transform the problem from one domain (the input domain) to another domain (the meta-data domain) where one of the available classifiers can achieve higher accuracy as compared to applying different training algorithms on the input domain. The Bagging ensemble technique is used to reduce the variance error.

We introduced both selection and mixing problems. A mixing problem is a generalization of the selection problem. Homogeneous and heterogeneous mixing problems are introduced. We introduce the BagStack classifier and the main contributions to deal with the data imbalance problem.

In the last section, we verified the performance of the BagStack classifier on more than 100 datasets. We addressed many aspects of the classifier trying to explain the effect of changing parameters that control the behavior of the classifier. We compare the classifier to other well-known ensemble classifiers that have been designed to deal with the data imbalance problem. Finally, we verify the performance of the BagStack classifier on image datasets.

Chapter 4

MULTIFEATURE, SPARSE-BASED APPROACH FOR DEFECTS CLASSIFICATION IN SEMICONDUCTOR UNITS

Automated inspection systems play an important role in manufacturing to guarantee higher quality and reduce production costs. Currently available defect detection and classification systems are customized and hard-wired to the detection of particular classes of defects and cannot deal with new unknown classes of defects. This issue is aggravated by the very small sample size of available anomalies for learning and by the data imbalance problem, since the number of defective samples is significantly much smaller than the number of normal samples. This work presents a novel multifeature, sparse-based defect detection and classification approach that uses the stacking and bagging concepts to enhance the classification accuracy. The stacking-based classifier is augmented with a novel adaptive over/undersampling technique to deal with the data imbalance problem. A simple pruning technique is proposed to eliminate bad base learners. Shortage of defective units, similarities within different classes of defects, wide variation within the same defect class, and data imbalance are the main challenges to deal with. Experimental results on real-world data from Intel show that the proposed approach results in a high classification accuracy as compared with the existing methods.

4.1 Introduction

Automated visual inspection (AVI) systems are gaining growing interest in the manufacturing industry. Inspection is an important process to detect defective products and keep these from reaching customers. Humans can be engaged in the inspection process but, due to issues, such as tiredness, consistency, and boredom, their performance is often unreliable. In some cases, such as when components to be inspected are very small and the production rate is very high, the use of manual inspection is not possible. Therefore, the AVI systems started playing important roles in advanced manufacturing to guarantee the quality of products in a timely manner [162].

Different sources of defects lead to large variations in defect types. Figure 24, 25 and 26 depict different types of defects. Currently available defect detection and classification systems are cus-

tomized and hard-wired to the detection of particular classes of defects and cannot deal with new unknown classes of defects. The inability to adapt to new types of defect results in additional handling and additional inspection time depending on the number and types of defects being inspected, and in the failure of detecting new or unknown defects. Dealing with new unknown defects necessitates redesigning the existing automated algorithms, which causes long development cycles, and delays, and which requires human effort to support constantly the design and development of the system.

Detection and classification tools can provide important information about defects' types, sizes, locations, distributions, and repetitions. Determining the stage of production responsible for the fault and classifying these defects are important when corrective actions need to be taken. This classification system allows the defect reduction team to identify the most important defect types and, in many cases, to provide a first pass indication of the most likely source of the defects and thus focus efforts on fixing the issue. Data mining methods have been used successfully in the field of data classification; however, to the best of our knowledge, no data mining method was proposed for defect detection and classification for semiconductor units in the manufacturing industry. Some prior works applied data mining methods to assessing the quality of semiconductor wafers and equipment but not semiconductor units [163, 164, 165, 166, 167]. The main difference between these two levels that the similarities between two wafers are higher than the similarities between two units related to the same product. Mainly, all subsequent steps after wafer fabrication, cutting, adding components and using epoxy to glue die onto the substrate, may introduce different types of variations. Discriminating between acceptable variations and defects is more challenging at the unit level.

In this work, we present a multifeature, sparse-based (instead of using the original representations of features, we use corresponding sparse representations) defect detection and classification approach that is adaptive to the number and type of defects and features. The approach adopts the stacking concept [ensemble stacking] and designs a novel stacking-based classifier to achieve high classification accuracy. In addition to the problem of designing the classification model, this work addresses related problems that include pre-processing techniques, features extraction and selection, and improving the data quality to help the classifier achieve the best performance. To deal with the data imbalance problem, the proposed stacking-based classifier is augmented with a novel adaptive sampling technique; the proposed adaptive sampling technique includes adaptive

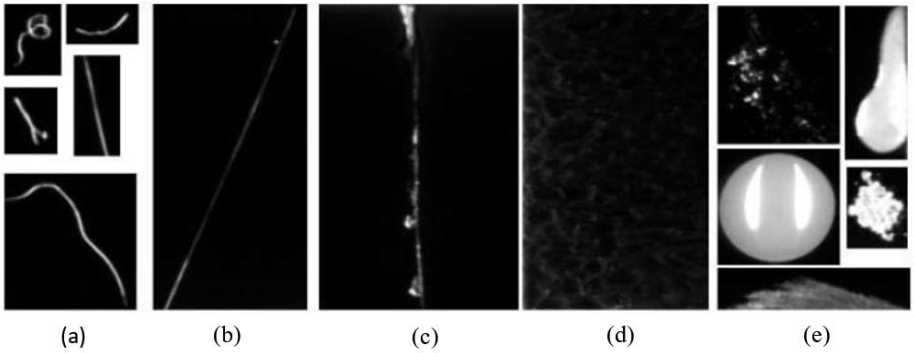

Figure 24: Defects in the die area. (a) Foreign material on die. (b) Scratch on die. (c) Crack. (d) Fingerprint on die. (e) Epoxy on die.

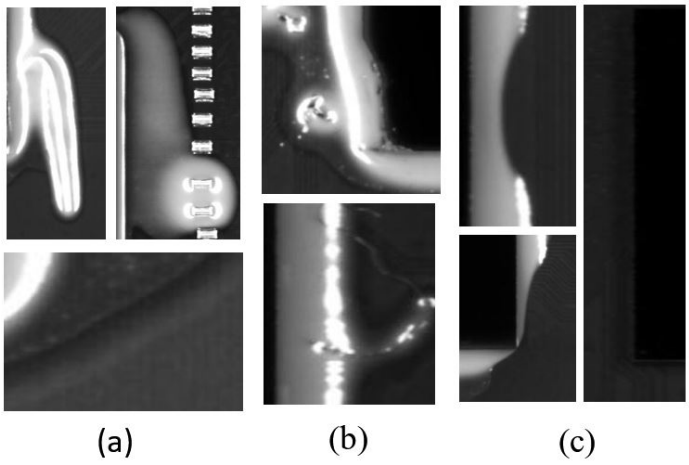

Figure 25: Defects in the epoxy area. (a) Excess epoxy. (b) FM on epoxy. (c) Missing Epoxy.

undersampling and a new synthetic oversampling method. The proposed approach is evaluated using real data collected at Intel and is shown to lead high classification accuracy. The approach is cost-effective, robust, and flexible in that it is easy to train by an operator using a small-size sample set of defects.

4.2 Types of Defects and Database

Defects can occur due to a variety of causes, including unintended human interaction, failures of machines that are used in the manufacturing process, low quality materials, or any unexpected

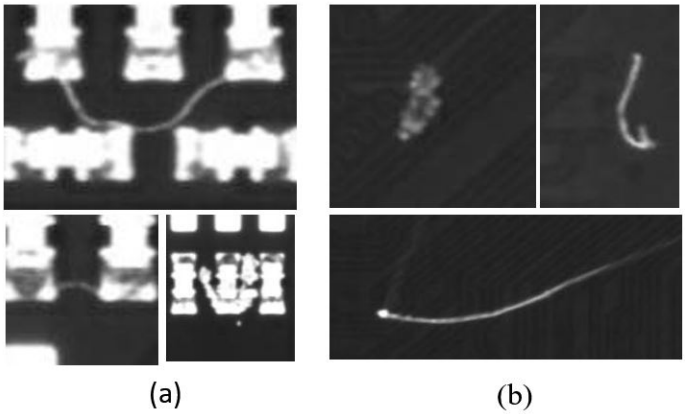

Figure 26: Other Defects. (a) Damage component. (b) FM on substrate.

events, such as power failure. Defects can affect one or more parts of the semiconductor unit, such as the die, epoxy, and substrate regions of the unit. The types of defects may differ depending on the manufacturing stage. Some of these defect types include cracks, fingerprints, epoxy on die, and defects due to foreign material, such as hair, threads, fibers, dust, or fluids. The basic objective to be achieved is to reduce underrejection and overrejection rates.

Many defects are considered in this paper, including foreign material on die, epoxy on die, scratches, fingerprints, and cracks. Figure 24, Figure 25 and Figure 26 shows the types of defects we are working with. The non-uniform distribution of defects can be considered as an additional challenge. For example, the probability of having foreign material is much higher than having scratch on die.

4.3 System Overview

The proposed system uses the means of image processing and data mining to perform defect detection and classification. Inputs of the system are grayscale images of semiconductor units. Images are taken by using high resolution line scan cameras $(\sim 16\ \mu m \setminus pixel)$; the used lighting condition is red bar lights. Images are captured using line scan cameras as the tray of parts scans under the camera. By using change detection and image processing means, defects are detected and segmented. The first step is to use image processing means to perform segmentation. As a result, the unit image will be divided into three separate parts (sub-images): die, epoxy, and substrate. Every part will be processed separately. The segmented sub-images are further processed

for detecting and segmenting out potential defect samples. In order to characterize the detected defect samples, each segmented defect sample is cropped (manually) by enclosing it within a tight bounding box, and is forwarded to the second processing block in which a suitable image representation is generated by extracting different features. We found that no single feature is good for all possible defect types. Therefore, in order to enable the system to adapt to different types of defects, multiple feature types $F_1,\ F_2,\ ...\ ,\ F_N$ are used as part of the proposed approach.

The shortage of defective samples emphasizes maximizing the usage of available defects' images in order to achieve a high classification accuracy. For this purpose, a background features subtraction is introduced to eliminate features due to background noise. The extracted non-background features are sparse coded. Sparse codes corresponding to the non-background features are generated by using separate dictionaries, one for each feature type. The resulting sparse multifeature representation is forwarded to the final stage, where the classification process takes place. The classification process adopts a stacking-based technique to combine different features. The proposed classifier is novel in terms of how subtasks are assigned to different BL_s , how features are distributed. The proposed classifier is augmented with a novel adaptive sampling technique to deal with the data imbalance problem when training each BL, and with a new ensemble pruning method to eliminate bad BL_s. More details about the defects' representation, feature extraction, and classification framework are presented in the next Sections.

4.4 Image Representation

The BagStack classifiers consists of base learners that are trained to solve different problems using different subsets of the data (Bagging), different training algorithms and different representations (Stacking – different bias classifiers). We consider eight different features, Scale-Invariant feature transform (SIFT) [139], Histogram of Gradients (HOG) [141], self-similarity descriptor (SSIM) [143], geometric blur [146], and deep neural network (DNN)-based features [149, 150, 151, 152, 153, 154, 155, 156]. SIFT [139] represents a patch of the image in terms of the weighted response of a gradient histogram and results in scale-, rotation-, and shift-invariant features. The number of extracted features can be controlled by setting a threshold value in SIFT, with a higher threshold resulting in a lower number of features. When using small threshold values (e.g., at zero threshold) most of the extracted features is not related to the defect of interest

but is mainly irrelevant background features. Increasing the SIFT threshold value can reduce the number of features, but cannot guarantee rejecting background features while keeping relevant defect-related features. Image processing can be used to set the background to zero, and thus, no features will be detected. Unfortunately, this will create strong edges causing artificial features to be detected according to these strong edges. These artificial features are shared among different defects, and thus affect the classification accuracy adversely. Eliminating background features in an efficient way is important. For this purpose, we train a non-linear SVM classifier to distinguish between defect-related features and background-related features. Background-related features are filtered out. Like SIFT, the HOG [141],[168] also describes an image patch in terms of image gradient histograms. In this work, we use the slightly improved HOG formulation described in [168].

Sparse signal representation arises in applications of compressed sensing and has been adopted successfully in image processing and computer vision applications. In sparse coding, the extracted features can be represented using a sparse representation [148] with a reduced number of nonzero elements. Given a predefined sample set $D = [c_1,\ c_2,\ ...\ ,\ c_k] \in R^{d \times k}$, also referred to as a dictionary D, and an input sample $y \in R^{d \times 1}$, where k is the number of elements in D and d is the feature dimension; the sparse representation problem can be formulated as the following minimization problem, where $x \in R^{k \times 1}$ is the sparse representation: $argmin_x \|X\|_1\ s.t\ D.x\ =\ y$, Our sparse coding framework consists of the following steps: 1) dictionary construction; 2) sparse coding; and 3) pooling. These three steps are described in the remaining part of this section.

Given a set of images $\{I_1, I_2, ..., I_n\}$, where n is the number of images. For each image I_i, a set of features $y_1^i, y_2^i, ..., y_{bi}^i$ is extracted, where $i\ (1 \leq i \leq n)$ is the image index, and b_i is the number of features in the i_{th}. Each type of feature is processed separately to generate a corresponding dictionary (one for each feature type). For all features belonging to a feature type, $y_1^1, ..., y_{b1}^1, ..., y_1^n, ..., y_{bn}^n$ with a total number $\sum_i^n b_i$, we apply $K - means$ clustering on all $\sum_i^n b_i$ features to get the dictionary $D = [c_1,\ c_2,\ ...\ , c_k]$, where the k^{th} dictionary element (also called atom) c_k is the centroid of the k_th cluster. Atoms (elements) in the dictionary are not required to be orthogonal. Sparse coding is done by finding a linear combination of a "few" atoms from D that is "close" to the signal y. Separate dictionaries will be used for different types of features.

The dictionary D contains atoms that can be used to represent each feature in the image (in here, the term "image" is used to refer to a candidate defect image region in die, epoxy, or substrate). Two main methods can be used: hard assignment and soft assignment. In hard assignment, extracted

feature is compared with atoms in the dictionary, and the closest atom in terms of Euclidean distance is used to represent it. One major limitation of hard assignment is that the representation is obtained by considering only the closest atom in the dictionary. However, it is possible that a feature may be close to multiple atoms. Recent studies, such as linear spatial pyramid matching (SPM) using sparse coding for image classification [169] and locality-constrained linear coding for image classification [170], show that soft assignment approaches, such as sparse coding, which assign multiple close atoms to represent features usually yield better results than hard assignment approaches. In this sparse coding framework, each feature is represented as a sparse linear combination of atoms in the dictionary corresponding to that feature type. The $\ell_1 - norm$ is used to generate the sparse code.

Let $D = [c_1, c_2, ..., c_k]$ be the dictionary where c_i values, $i = 1, ..., K$, are the atoms. Denote y_b^i as the b_{th} feature in the i_{th} image. We try to find a sparse linear combination of atoms to represent the feature y_b^i. Mathematically, we solve the following optimization problem:

$$X_b^i = argmin_x \frac{1}{2}\|Dx - y_b^i\|^2 + \lambda\|x\|_1, s.t.\ x \geq 0 \tag{4.1}$$

Where $x_b^i \in R^{k\times 1}$ is the sparse code for the feature y_b^i, $\lambda > 0$ is a tunable parameter, and $\|.\|_1$ denotes the $\ell_1 - norm$. A larger λ yields a code x_b^i that has more zero elements (higher level of sparsity). The above problem is well known as "LASSO" [171] with an additional non-negativity constraint. This problem can be efficiently solved by many packages, such as the sparse learning with efficient projections and stochastic coordinate descent approach [172].

Pooling plays a very important role in image classification, which is usually used to achieve more compact image representations that are more robust to noise and variation. Let X^i be the collection of sparse codes for features in the i^{th} image, such that $X^i = [x_1^i, x_2^i, ..., x_b^i]$. We define the pooling function by $h^i = p(X^i)$, where h^i is the pooled image representation from X^i. The pooling function $p(.)$ is applied to each row of X^i. Thus, the pooling function p yields a pooled image representation h^i of size $k \times 1$, where K is the number of atoms (base elements) in the dictionary. Different pooling generates different image representations. There are different choices of pooling functions commonly used in the image processing community, such as average pooling, $\ell_2 - norm$ pooling, and max pooling. In average pooling, each code is treated equally. Suppose u_m^i is the m^{th} row of X^i, then the pooled image representation is computed as $h_m^i = avg(u_m^i),\ m = 1, 2, 3, ..., k$. Average pooling is the most commonly used. However, average pooling treats each code equally, which may degrade the performance, since there may exist a lot of codes irreverent

to image defects. Like average pooling, $\ell_2 - norm$ pooling utilizes all information among codes. In $\ell_2 - norm$ pooling, the pooled image representation is obtained as $h_m^i = \|u_m^i\|, m = 1, 2, 3, ..., k$, where $\|u_m^i\|_2$ values represent the $\ell_2 - norm$ of u_m^i. The $\ell_2 - norm$ pooling usually works better when the codes are not very sparse. In contrast to average pooling and $\ell_2 - norm$ pooling, max pooling only selects the strongest signal among the codes. The pooled image representation is obtained as $h_m^i = max(u_m^i),\ m = 1, 2, 3, ..., K$. Theoretical analysis of pooling functions suggests that max pooling leads to the best results when the codes are sparse. In fact, methods with max pooling have been shown to achieve the state-of-the-art results in image classification [173].

4.5 Classification Framework

In the multi-class classification problem, where multiple features are needed to distinguish between different classes, designing a strong classifier that can deal with these classes in an efficient way is a challenge. In many cases, one may have to deal with making decisions regarding which features to use, and which base- \ meta- classifier(s) would be the most appropriate for the considered problem. Using multiple features can be done by using ensemble methods, e.g., bagging, boosting, stacking, or classification hierarchy. Ensemble techniques were shown to result in improved performance in terms of classification accuracy. In this work, we propose a stacking-based classifier. Stacking is a multilayer ensemble method of combining multiple models that have been learned for the classification task(s). The output of a former layer is taken as the input information of the subsequent layer in order to learn a classification model, so as to obtain a combined classification result with better classification performance as compared with an individual classifier.

In the literature, multiple versions of stacking techniques were developed. The first version of the stacking ensemble technique was introduced in [49] and was later extended to regression problem. A detailed review of more recent stacking is presented in the literature review. The major limitation of these stacking techniques is the demand for large number of samples to perform training. Furthermore, these methods do not offer strategies to deal with the data imbalance problem nor strategies to eliminate bad BL_s.

The proposed stacking-based classifier is designed to deal with the data imbalance problem, small number of training samples, similarity between classes and variation within the same class by integrating novel adaptive over-/under- sampling technique [adaptive undersampling and new syn-

thetic minority oversampling technique (SMOTE)-based synthetic oversampling], a new ensemble pruning to eliminate bad BL_S, and metadata synthetic oversampling.

In this section, we introduced the first version of BagStack classifier. It is the classifier that used to generate the results at the end of this chapter. Since in the first phase, we worked mainly with manually cropped defects on the *die region*, the number of defects to consider was slightly small. In fact, many of the parameters that are adaptively specified (in BagStack Classifier) in the run time based on the task at hand are heuristically optimized in this first version.

4.5.1 Stacking: One-versus-all Confidence-based System

Ensemble methods for classification have been shown to achieve a better classification accuracy than the individual component classifiers if the component classifiers perform better than chance (better than random guessing) and are diverse, which gives the ability to eliminate independent errors. Meta-learning techniques employ a metaclassifier that generalizes over the space of outputs from base-level classifiers. The proposed metaclassifier is a stacking-based classifier that uses one-versus-all (OVA) trained base classifiers. More details about the training of BL_S and the metaclassifier are given in the following.

Figure 27 depicts the proposed classification framework. Let M be the number of classes. Every sample will be represented by using a set of N features: $F_1,\ F_2,\ ...\ F_N$. In one implementation, our system makes use of three features ($F_1 = F_{SIFT},\ F_2 = F_{HOG},\ F_3 = F_{SHXT}$). However, the proposed system can be designed to use other features. For each $class\ m,\ m = 1, 2, ..., M$, $Q\ BL_S$ are used for each feature resulting in $Q \times N\ BL_S$ for N features to implement an (OVA) classifier for the corresponding $class\ m$. This results in a total of $Q \times N \times M\ BL_S$· $BL,\ BL_{m,q:F_i}$, assigned to a $class\ m$, where $1 \leq m \leq M$, $1 \leq q \leq Q$ and $1 \leq i \leq N$, is trained to perform the classification problem ($Class\ m\ vs.\ All$) based on feature F_i. Every time a base classifier $BL_{m,q:F_i}$ is trained, L samples are sampled per class and adaptive sampling (up/down), depending on the number of samples of the class, is applied. **This is different from the BagStack classifier, where the number of base learners assigned to each problem is adaptively specified depending on the data imbalance ratio and other constraints**.

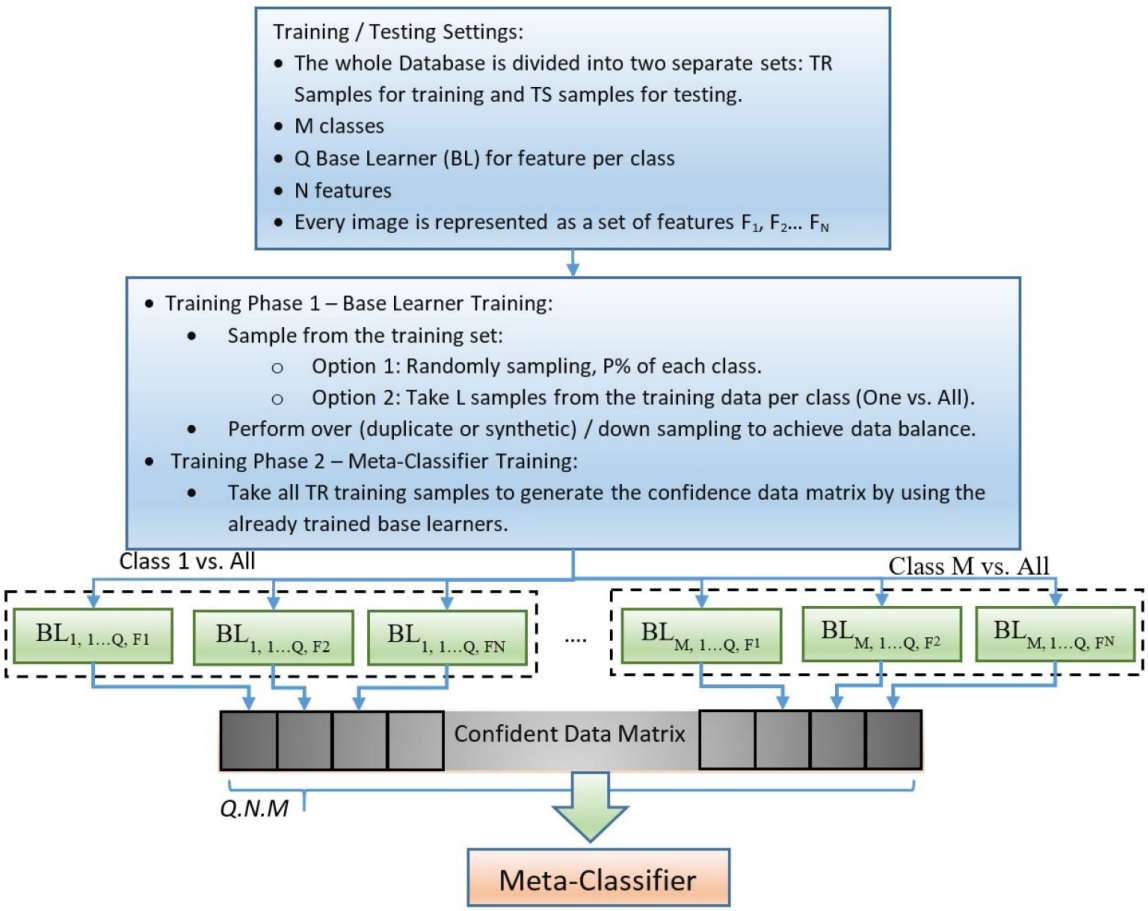

Figure 27: Stacking-based, OVA confidence-based system. For training, the confidence data fed to the metaclassifier correspond to TR training samples, and thus takes the form of a TR x (Q · N · M) matrix. For testing, the confidence data are (Q · N · M)-length vector.

As indicated previously, during the training process, L samples per class are randomly sampled from the training data and used to train a base classifier. For training the metaclassifier, the entire training data are used to generate a $TR \times (Q.N.M)$ confidence matrix, where TR is the size of the training data, Q is the number of BL_S used for each feature per class, M is the number of considered classes, and N is the number of features. For each training sample, the confidence outputs of BL_S are concatenated to generate a row of the confidence matrix. The confidence matrix entries are then used to train the metaclassifier.

In the case where the training set is sufficiently large, it is recommended to divide the training data into two subsets. The first subset can be used to train the first layer of classifiers (BL_S) and the second subset can be used to test these classifiers and generate the confidence matrix that will be used to train the metaclassifier. In our case, we have two main problems: the lack of a

large training data set and the severe data imbalance problem. The value L should be selected, such that BL_S will not suffer any overfitting problem. Selecting L to be very large will affect the quality of metadata that is used to train the metaclassifier. Selecting L to be very small may lead to overfitting problem. Adaptive data sampling and ensemble pruning strategies are proposed to deal with this challenge. To enhance the quality of the classifier (the strong classifier), we need to be sure that the values generated by the BL_S can be reliably used by the metaclassifier to generate final decisions. Increasing L can enhance the quality of each BL, since more training samples are used to train each BL. At the same time, since we are sampling randomly from the training data, if L is very large, this creates multiple copies of the same BL, thus losing the diversity property. Diversity is very important in stacking-based classifiers and represents one of the major success factors. In addition to that, because we are testing on the whole training data, if a relatively large L value is used, most of the training data samples are used in training BL_S and, thus, have been seen by BL_S before generating the confidence matrix. The BL_S would erroneously generate a high confidence value even when the corresponding feature is not a good one for the considered classification problem. **This is different from the BagStack classifier where imbalance-aware cross validation is used to generate the meta-data.**

4.5.2 Adaptive Sampling

The proposed sampling technique is designed mainly to deal with the lack of training samples as well as the data imbalance problem. Each time a BL is trained, L samples are selected per class (Class c versus All) where $All = \bigcup_{i=1,i\neq c}^{m} Class_i$. L samples are sampled from $Class\ c$ and $(L \setminus M)$ samples are sampled per each class ($Class_i \in All$). For each class, if the available number of samples is greater than the target number of samples (L for $Class\ c$ and $(L \setminus M)$ for all other classes), undersampling is applied, and if the number of samples is less than the target number of samples, synthetic oversampling is applied.

The proposed synthetic approach is a modified version of SMOTE algorithm. The SMOTE algorithm has been modified to consider three important factors: similarity between different classes, variation within the same class, and the small number of samples. SMOTE algorithm includes specifying the $KNNs$. In some cases, the number of samples of the class that needs to be over-sampled is very close to k. Because of the variation within the same class, there is no guarantee that the

KNN samples are near neighbors. In SMOTE, the random number a can take any value between zero and one, and thus the new synthetic sample can be anywhere between the two original samples. If these two samples are not neighbors and because of variations within the class, there is a probability that the synthesized sample will be closer to the other class (the majority class). In the modified version, the random number a can take a value between 0 and $\omega = 1 - \frac{1}{\sqrt[z]{n}}$, where n is the number of samples and z is a positive integer number. Smaller number of samples results in smaller ω and larger number of samples results in larger ω. In other words, when the number of samples belonging to the minority class is very small, we enforce the synthesized sample to be closer to the original sample by choosing a smaller ω.

If the data imbalance ratio is very severe and since the numbers of samples of minority classes are much smaller than the numbers of samples of the majority classes, undersampling can lead to loss information. The proposed undersampling approach is motivated by the Border-Line SMOTE approach. The basic idea is to specify the most difficult samples belonging to the majority class. Each sample $s_i \in \ majority\ class$ is assigned a distance weight $Wd = min_v\ distance(s_i, r_v)$, where $r_v \in minority\ class$. The samples with a small Wd are more difficult samples. Then, the samples are ordered based on the distance weights and the exponential distribution is used for sampling. By doing this, we allow the algorithm to focus more on difficult samples, but also support diversity by considering other samples.

4.5.3 Ensemble Pruning

Ensemble learning consists of training a number of BL_S whose predictions are then combined to yield a single classification decision. In general, using complementary classifiers can lead to a higher accuracy than individual classifiers in the ensemble. The main drawback of the ensemble techniques is the need to have full access to all these BL_S. The results generated by these BL_S should be available before generating the final decision. Usually, using ensemble strategies may push the demand to higher computational capabilities and more storage. In some cases, defining bad BL_S that do not contribute toward higher accuracy is also necessary. Furthermore, eliminating these BL_S can reduce the computational complexity and storage requirements for testing new samples.

To address these issues, several methods have been developed to select a subset of the classi-

fiers within the ensemble that solves the classification problem in an optimal way. These methods are named pruning. Given a set of BL_S, confidence data matrix, and the corresponding labels of each row of the confidence matrix, the proposed pruning method can analyze the ability of each BL to distinguish between the target $Class\ c$ and the all other classes $(Class\ c\ vs.\ all)$. The basic idea is to use the entropy to measure the quality of each BL. Each column (BL) of the confidence data matrix is processed separately. A weight is assigned to each BL based on it is ability to distinguish between the two classes. These weights are calculated as follows. The first step is to apply a positive/negative rectification on the confidence matrix. Any value greater than or equal to zero is set to one $(+1)$ and any value less than zero is set to negative one (-1). In the second step, we find the sum of all elements belonging to the same label (label c and label all). If the two sums $(sum_c\ and\ sum_all)$ have the same sign, the efficiency of this BL to distinguish between the two classes is low, and thus the weight of this BL is set to zero. If the two sums have different signs, the weight of the BL is set to be the average of the normalized sums, where the normalized sum related to $Class\ c$ is the division of sum_c by the number of samples that have the label c and the normalized sum related to the class all is the division of sum_all by the number of samples that have other labels.

Given that only $B\%$ of the BL_S need to be kept. The simplest way is to select $B\%$ of the BL_S that have the highest weights. But, this may affect the diversity property, which is significant factor in the success of ensemble strategy. Instead of selecting $B\%$ of BL_S that have the highest weights, we divide the selection process into subset selection problems. Each set of BL_S that solve the problem $Class\ cvs.all$ is processed separately. $(B\% \setminus (M)) \times Q.N.M\ BL_S$ of the highest weights of each set are selected, where $Q.N.M$ is the total number of BL_S and M is the number of classes. The confidence data matrix is modified by removing columns corresponding to deleted BL_S. The rest of the confidence data matrix is used to train the metaclassifier. **There are two benefits of applying pruning: (1) reduce the computational complexity and (2) reduce the tendency of the meta-classifier to overfit the meta-data**. We noticed that reducing the number of base learners can improve the performance if the meta-classifier is used without any regularization (e.g., SVM(RBF) is known to overfit the data if not regularized carefully). In the BagStack classifier, we propose the EB-MDTR classifier, which has better ability to resist overfitting and thus applying pruning in this case may not improve the performance.

4.5.4 Meta-data Synthetic Oversampling

The data confidence matrix, which is used to train the metaclassifier, was generated by using an imbalanced data set. Therefore, achieving a high quality metaclassifier necessitates dealing with the data imbalance problem at the metadata level. For this purpose, the aforementioned SMOTE algorithm is used to deal with the data imbalance problem by oversampling the minority classes. Again, BagStack classifier handles this issue using a different strategy. **In the proposed BagStack classifier, we propose imbalance-aware cross validation to handle the data imbalance problem at the meta data level**.

4.6 Experimental Results

In this section, we evaluate the main two components of the system: the defect representation framework and the classification framework. We evaluate the proposed system from two aspects: classification accuracy and stability. The performance results are obtained by applying the proposed system on real-world data from Intel.

4.6.1 Database and Experimental Setup

The database provided by Intel Corporation contains images for more than 3000 units. More than 95% of these images are clean images. The rest of the images are distributed over five types of defects in a nonuniform fashion. The defects we are working with are: foreign material on die (48 Images), scratch on die (9 Images), fingerprint on die (8 Images), epoxy on die (69 Images), and cracked die (33 Images). A model was trained to perform background features subtraction and separate dictionaries (one for each feature) were generated by applying K-means clustering with k = 2000. These dictionaries were then used to generate sparse codes corresponding to extracted features. Max pooling was used to combine features to come up with a single representation for the image for each feature type. The resulting multifeature sparse representations are then input to the stacking-based classification stage.

Linear SVM classifiers are used as BL_S and a nonlinear SVM (RBF) is used as a metaclassifier. For each feature type, 100 BL_S are used. Thus, the total number of BL_S is $100 \times N$, where

N is the total number of features. Each time a BL is trained, L samples are sampled per class $(Class\ c\ vs.\ all)$ or $P\%$ of samples are sampled, depending on which option is used. For all experimental results, 70% of the data are used for training and 30% are used for testing. The reported classification results are the average of 30 simulations.

4.6.2 Data Representation

We compare the proposed multifeature sparse-based defect characterization framework with other well-known existing methods, such as linear SPM using sparse coding, locality-constrained linear coding, Dirichlet-based histogram feature transform, image feature by histogram of oriented $p.d.f.$ gradients, and DNN-based features. Different DNN models are used, AlexNet, GoogLeNet, and VGG19, to extract features. For all of these DNN networks, we extract the responses of the last fully connected layer. Table 18 shows the classification accuracy. For each one of these experiments, nonlinear SVM is used as a classifier, except for the last two rows. In the last two rows, we use the proposed classifier with different sets of features. In one case, we consider the classifier with hand engineered features ($SIFT$, HOG, and $SHXT$ descriptors), and in the other case, we consider features extracted by using pre-trained DNN networks. The purpose of this comparison is to show that the data representation framework is an efficient one comparing with the others' systems. To keep the comparison consistent, we use the same dictionary size (2000) for our method, the linear SPM using sparse coding method, and the locality-constrained linear coding method.

Table 18: Comparison of different representation methods in terms of classifications accuracy.

Data Representation	Classifier	Feature	OA	AA
Linear SPM using sparse coding		SIFT	92.2	81.6
Locality-constrained linear coding		HOG	92.9	85.7
Dirichlet-based histogram feature transform		SIFT	88.7	68.8
Histogram of oriented p.d.f gradients	SVM (RBF)	SIFT	89.2	71.8
AlexNet		DNN-Based	91.3	85.6
GoogLeNet		DNN-Based	89.4	86.2
VGG		DNN-Based	90.4	87.7
multi-feature sparse based	SVM (RBF)	SIFT	93.4	87.9
		HOG	93.6	91.18
	Proposed classifier	SIFT, HOG and SHCXT	97.6	93.9
		AlexNet, GoogLeNet and VGG19	94.3	91.5

70% of the data is used for training and 30% is used for testing. The results are the average over 30 simulations. In all cases a nonlinear SVM model is used as the main classifier except for the last two rows, where the proposed stacking based classifier is used (120 linear SVM base learners, three features, nonlinear SVM meta-classifier, L=24 samples)

Table 18 presents the obtained classification performance results. We consider two important measures: the overall accuracy and the average accuracy. The overall accuracy indicates the actual number of images classified correctly out of all images in the testing set. The average accuracy

is obtained by first computing the accuracy for each individual class, and then taking the average over the computed class-based accuracies. This measurement is particularly important when the data are imbalanced with some classes having significantly more samples than others, which can bias the overall accuracy. Maximizing both of these measures is desired. The highest values for these measures are obtained by using the proposed multifeature sparse-based classification method. From Table 18, it can be seen that our proposed method achieves a higher classification performance as compared with others even when only using a single feature type (e.g., SIFT or HOG). We also compare the proposed classification framework with feature-transform-based techniques. The achieved classification accuracy of the proposed method is higher as compared with these methods even when using the SIFT feature only. Recently, DNN has been attracting attention for large scale image classification problems. Training DNN models is very expensive and requires a large number of images to avoid overfitting. Instead of training a DNN model from scratch, which is also not feasible due to the low number of training defect samples in our application, we used three pre-trained models to extract features. Sparse-based data representation framework can lead to higher classification accuracy as compared with the existing methods even when using single feature, such as SIFT or HOG, and even when using a typical nonlinear SVM classifier. The classification accuracy increases further when using the proposed multifeature sparse-based data representation with the proposed stacking-based classifier. In addition, it should be noted that our multifeature sparse-based characterization framework results in a higher classification accuracy as compared with using DNN-based features.

4.6.3 Classification Framework

In this section, we evaluate the proposed stacking-based classifier. We consider four important evaluations. First, we address the effect of using adaptive sampling, ensemble pruning, and metadata synthetic oversampling on the classification accuracy. Second, we evaluate the proposed classifier when using different sampling techniques to deal with the data imbalance problem when training each BL, including the proposed adaptive sampling technique. Third, we compare the classification accuracy of the proposed stacking-based classifier to other well-known ensemble methods that are designed specifically to deal with the data imbalance problem. Fourth, we evaluate the sta-

bility and the scalability of the system when adding more data to the training set and using more features.

Table 19 shows the classification accuracy when using the proposed classifier. Six features are used (SIFT, HOG, SSIM, GB, and two DNN-based features). A total of 600 linear SVMs are used as BL_S (100 per feature). The proposed data representation framework is used to represent the images. Sparse codes are generated by using separate dictionaries of 2000 atoms. In Table 19, the first row shows the classification accuracy when the proposed stacking classifier is tested without applying adaptive sampling, ensemble pruning, or metadata synthetic oversampling. The second row shows the classification accuracy when using the proposed adaptive sampling. The third row shows the classification accuracy when ensemble pruning is enabled, and metadata synthetic oversampling is disabled. The fourth row shows the accuracy when ensemble pruning is disabled, and the metadata synthetic oversampling is enabled. Finally, the last row shows the accuracy when both are enabled. From Table 19, it can be seen that the major improvement in both overall and average accuracy happens when we use adaptive sampling instead of random over-/under- sampling. Introducing ensemble pruning or metadata synthetic oversampling (second and third rows of Table 19) improves the average accuracy, but can affect the overall accuracy negatively. Since the testing set is an imbalanced data set, the average accuracy is more important and a better representative for the system performance. Table 19 shows that the best performance is obtained when the proposed adaptive sampling, ensemble pruning, and metadata synthetic oversampling are enabled (last row of Table 19). The adaptive sampling enhances the quality of each BL. The ensemble pruning eliminates bad base learners. Finally, augmenting ensemble pruning with metadata synthetic oversampling can enhance both of overall and average accuracies. The last column to the right shows the time required to run each experiment. It can be seen that the computational complexity is less when using ensemble pruning, due to the fact that fewer BL_S are used when testing new samples; thus, less time is required to generate the confidence vector for each sample.

In order to evaluate the proposed adaptive sampling, we compare different types of sampling techniques to deal with the data imbalance problem when training BL_S. In the proposed approach, L samples are sampled for each class $(class\ c\ vs.\ class\ all)$ and adaptive sampling is used to deal with the data imbalance problem. For comparison, $P\%$ of the training data are sampled per class and different existing sampling techniques are used to deal with the data imbalance problem. Table

Table 19: Evaluating the proposed classifier when using adaptive sampling, ensemble pruning, and meta-data synthetic oversampling.

Classifier	Adaptive Sampling	Pruning	Meta-data Synthetic OS	OA	AA	training t(s)	testing t(s)
The proposed classifier	OFF	OFF	OFF	96.2	92.74	317	1.31
	ONN	OFF	OFF	98.1	97.01	397	1.42
	ONN	ON	OFF	97.5	98.12	405	0.253
	ONN	OFF	ON	97.6	97.24	415	1.34
	ONN	ON	ON	98.3	98.2	421	0.26

70% of the data used for training and 30% for testing. The results are the average over 30 simulations. In all cases: L=24 samples, 600 base learners (linear SVM) are used, nonlinear SVM (RBF) is used as a meta-classifier. When adaptive sampling is active, z=3 is used. When ensemble pruning is active (1/N) of the total number of base learners are kept, where N is the total number of features. when meta-data synthetic oversampling is active, SMOTE synthetic oversampling is used. 6 features are used SIFT, HOG, SSIM, GB and 2 DNN-based features (Ref Model: FC6 and FC7)

20 shows the comparison of using different existing sampling techniques with the proposed stacking-based classifier. For each class in the training set, 50% of the training samples are used to train each BL. The problem that should be solved by each BL is OVA, where the sampled instances belonging to the target class (one) are grouped together and all other instances belonging to other classes (all) are grouped together to form one class. As indicated earlier, the OVA problem is inherently imbalanced, as the set of all data points from all other classes is likely to outnumber the number of samples of the target class.

In RUS, the majority class among these classes (one and all) is under-sampled by dropping samples randomly until both of them have the same number of instances. In all other cases, synthetic oversampling is used instead of undersampling. Table 20 shows the classification accuracies when using the proposed adaptive sampling method, the original SMOTE algorithm, Adaptive SMOTE, and Border-Line SMOTE. Table 20 shows that the highest classification accuracy can be achieved when using the proposed adaptive sampling technique.

Table 20: Evaluating the proposed classifier with different sampling techniques in terms of classification accuracy.

Classifier	Sampling Method	OA	AA	D5	D17	D24	D13	D9
The Proposed Classifier - Pruning = ON - Meta-Data Synth = ON - Over Sampling = ON	Adaptive Sampling	98.31	98.17	99.1	98.5	93.3	100	100
	RUS	94.01	94.01	97.4	96	90	100	86.6
	SMOTE	95.94	95.86	97.6	95	97	100	90
	AdaSMOTE	96.03	94.33	98.0	96.2	96	100	82
	Border-Line SMOTE	95.77	94.05	97.4	98	98	100	83.3

70% of the data is used for training and 30% is used for testing. The proposed stacking-based classifier is used for all the compared settings. The number of base learners is 600. 6 features are used, SIFT, HOG, SSIM, GB and two DNN-Based Features. 50% (L=24 samples) of the training data is sampled to train each base learner. Linear $SVMs$ are used as base learners and nonlinear SVM (RBF) is used as meta-classifier. Ensemble pruning is enabled and $B\%$ =1/N, where N= 6, of the total base learners are kept. Meta-data synthetic oversampling is enabled, SMOTE synthetic oversampling is used to deal with the data imbalance problem at the meta-data level. when adaptive sampling is used z value is 3.

The next set of experiments was done to compare the classification accuracy of the proposed classification framework with other classification models. In these experiments, we compare with Boosting-based and Bagging-based ensemble methods. These methods are designed to deal with the data imbalance problem. The Boosting-based methods include AdaBoost.M1 and Viola–Jones

classifier; these two methods change the distribution (weights) of the training data adaptively such that as the training is going forward, the classifier will focus more on difficult samples. SMOTE-Boosting and Border-Line SMOTE-Boosting are techniques designed specifically to deal with the data imbalance problem by synthetically oversampling the minor classes in the training subset. The training subset is the subset used to train each BL. The RUS-Boost ensemble technique uses under-sampling to deal with the data imbalance problem. Adaptive RUS-Boost considers the distribution (weights) of the training samples when performing the under-sampling. The algorithm keeps difficult samples by dropping easy samples of small weights. The Bagging-based methods include the bagging technique with under-sampling, SMOTE-Bagging, Border-Line SMOTE-Bagging, and the RBBag.

In bagging with under-sampling, an equal number of samples ($P\%$ of the training instances belonging to the minority class) are sampled per class, with the number of samples sampled per class being equal to the number of samples in the smallest minority class to train each BL. It is important to note that the BL_S used with Boosting and Bagging ensemble methods are multi-class classifiers. In SMOTE-Bagging and Border-Line SMOTE-Bagging, synthetic oversampling is used to deal with the data imbalance. For each class, samples are synthesized, such that the total number of samples in the current training subset belonging to the same class is equal to the number of samples of the majority class in the same subset. The proposed stacking-based classifier is used with ensemble pruning and metadata synthetic oversampling to generate the results shown in Table 21. The proposed multifeature sparse-based defect characterization framework is used for all the compared classifiers. From Table 21, it can be seen that the highest classification accuracy can be achieved when using the proposed confidence-based classifier.

Table 21: Comparison of different classification methods in terms of classification accuracy.

Classifier	OA	AA	D5	D17	D24	D13	D9
The proposed classifier	98.31	98.17	99.1	98.5	93.3	100	100
AdaBoost.M1	95.51	91.68	95.3	98.1	96.6	100	68.3
Viola-Jones	96.55	93.51	96.8	98.3	97.5	100	75.0
SMOTE-Boosting	95.08	90.51	96.8	97.5	95.0	100	63.3
Border-Line SMOTE Boosting	95.6	90.54	98.0	98.1	95.0	100	61.66
RUS Boost	94.05	95.66	95.0	91.3	97.1	100	95.0
Adaptive RUS Boost	91.98	93.46	96.0	85.6	99.2	100	86.66
Bagging-under sampling	87.05	93.0	88.8	78.9	97.2	100	100
SMOTE-Bagging	96.7	94.6	97.4	98.1	95.0	100	82.5
Border-Line SMOTE Bagging	96.8	95.8	97.6	98.3	93.2	100	90.0
RBBag	89.4	94.04	91.1	81.1	98.0	100	100

70% of the data is used for training and 30% is used for testing. The results are the average over 30 simulations. The number of base learners is 600, 6 features are used SIFT, HOG, SSIM, GB, and two DNN-Based Features. 70% (L=24 samples) of the training data is sampled to train each base learner. Linear SVMs are used as base learners. Nonlinear SVM (RBF) is used as meta-classifier for the proposed approach. Ensemble pruning is enabled and ($B\%$=1/N, where N=6) of the total base learners are kept. Meta-Data synthetic oversampling is enabled, SMOTE synthetic oversampling is used to deal with the data imbalance at the meta-data level. Adaptive sampling is used with a z value of 3.

Table 22 shows the incremental classification accuracy when using the proposed model with 100 BL_S per feature when incrementally using more training data and/or more features. From Table 22, it can be seen that using multiple features to perform classification outperforms using any of the features individually. Similarly, increasing the percentage of the training data increases the classification accuracy. The system can achieve high classification accuracy even when using a small number of samples to perform the training. For example, Table 22 shows that the proposed system can achieve a 95.88% average classification accuracy using only 30% of the available data to perform training.

Table 22: Incremental classification accuracy.

Percentage of data used for training	SIFT	+HOG	+SSIM	+GB	+REF: F7, + REF: F6
30%	76.8	78.3	80.05	81.2	95.88
40%	84.8	85.1	86.98	87.77	96.15
50%	92.9	93.0	93.1	93.1	97.62
60%	93.0	93.0	93.0	93.26	97.98
70%	93.9	94.0	94.3	94.4	98.05
80%	94.5	95.0	95.0	97.9	98.5

The most left column shows the percentage of data used for training. The rest of the data used for testing. We tested the proposed classifier by increasing the percentage of training data and the number of used features. the second column shows the classification accuracy when using only SIFT feature, the third column shows the result when using SIFT and HOG, ... etc. 100 x N linear SVM base learners used each time where N is the number of features. The used meta-classifier is a nonlinear (RBF) SVM. Adaptive sampling (z=3), ensemble pruning ($B\%$=1/N) and meta-data synthetic (SMOTE) are enabled. The results are the average over 30 times simulations.

4.7 Additional Experiments: Defects Classification Using the BagStack Classifier

This section presents the classification accuracy of the BagStack classifier implementation as discussed in chapter 3. In this section, we will also consider defects on the epoxy, on the substrate in addition to the defects on the die area. We consider 10 different defect types: Foreign material on die, epoxy on die, scratch on die, finger-print on die, foreign material on epoxy, excess epoxy, missing epoxy, damage components and foreign material / scratch on substrate (Table 23).

The BagStack classifier is used in different settings. In these settings, we vary the number and types of base learners (Table 24), the number and types of features (SIFT and DNN based features), different meta-classifiers (Random Forest (100 with optimization), EBMDTR0 and EBMDTR1, Linear SVM, SVM(RBF), Bagging(DT) and Boosting(DT)) and turning On/Off the imbalance aware cross validation. Coverage percentage is the only constraint that is used to decide the total number of base learners, 90% coverage percentage is used when we turn off the imbalance-aware cross validation and 80% coverage percentage is used when we turn on the imbalance-aware cross

Table 23: Types of defects, descriptions and number of images available for each type.

Defect type	Region	Description	Number of Images
Foreign material on die	Die	Foreign material (hair, dust, etc) on die	60
Fingerprint on die	Die	Fingerprint on die	19
Scratch on die	Die	scratch on die	8
Epoxy on die	Die	Excess epoxy over the die area	40
Cracked die	Die	Cracked/broken die	21
Excess epoxy	Epoxy	Excess Epoxy over the substrate	12
Foreign material ob epoxy	Epoxy	Foreign material on epoxy	44
Missing epoxy	Epoxy	Missing epoxy/ partially missing eoxy	33
Damage component	Component	Foreign material or epoxy touching the component	26
Foreign material on substrate	Substrate	Foreign material/scratch on substrate	89

validation. In the first version EBMDTR0, we concatenate the outputs of all base learners to construct a meta-data instance, in the second version EBMDTR1, to fit a regression tree to predict a specific class c, we use meta-data instances that are constructed by concatenating the outputs of base-learners that have been trained using the class c.

Table 24: Training algorithms used as base learners.

Index	Training Algorithms	Parameters
BL1	Decision Tree	Maximum number of splits = 400
BL2	SVM	SVM, Linear, Z-Score
BL3	Linear Regression	SVM-Regression, Lasso, Sparsa, Lambda=0.1, no optimization
BL4	Linear Regression	LSE-Regression, Lasso, Sparsa, Lambda=0.1, no optimization
BL5	Linear Regression	SVM-Regression, Ridge, Sparsa, Lambda=0.1, no optimization
BL6		LSE-Regression, Ridge, Sparsa, Lambda=0.1, no optimization
BL7	SVM	SVM, RBF, Z-Score
BL8	Random Forest	Random Forest, classification, number of learners=5
BL9	Ensemble regression-bagging	Ensemble regression, bagging, number of learners =10
BL10	Ensemble regression, boosting	Ensemble regression, LSBoost, number of learners=10

Tables 25, 26, 27 and 28 show overall classification accuracy, average accuracy, standard deviation and the average classification accuracy of each class (the 10 defects classes). In each experiment, we use 80% of the data for training and 20% for testing. The reported results are the average of 10 times.

Table 25: Overall accuracy and average accuracy when using two features and two types of base learners.

MC	IACV	Overall		Average		Die					Epoxy			Others	
		OA	STD	AA	STD	CR	EPX	FM	FP	SC	EXS	FM	MIS	SUB	DC
SVML	OFF	0.89	0.03	0.81	0.06	0.97	0.88	0.92	0.2	0.8	0.82	0.9	0.93	0.94	0.72
SVML	ON	0.89	0.02	0.9	0.04	0.97	0.83	0.95	0.9	0.88	0.89	0.83	0.96	0.88	0.94
SVM	OFF	0.86	0.02	0.72	0.05	0.63	0.91	0.85	0.1	0.53	0.82	0.89	0.92	0.94	0.62
SVM	ON	0.87	0.02	0.88	0.05	0.93	0.87	0.94	0.8	0.9	0.93	0.81	0.92	0.85	0.86
RF	OFF	0.91	0.02	0.87	0.03	0.97	0.94	0.94	0.4	0.85	0.89	0.88	0.94	0.93	0.92
RF	ON	0.91	0.01	0.9	0.02	0.97	0.88	0.94	0.6	1	0.92	0.81	0.99	0.92	0.94
Bagging	OFF	0.88	0.02	0.84	0.04	0.97	0.86	0.94	0.3	0.88	0.89	0.81	0.92	0.91	0.88
Bagging	ON	0.86	0.03	0.87	0.02	0.97	0.81	0.87	0.7	0.95	0.85	0.78	0.95	0.88	0.98
RUSBoost	OFF	0.76	0.06	0.77	0.06	0.93	0.79	0.8	0.6	0.83	0.76	0.69	0.71	0.76	0.82
RUSBoost	ON	0.8	0.03	0.83	0.04	0.9	0.72	0.89	0.7	0.88	0.89	0.69	0.9	0.78	0.92
EBMDTR0	OFF	0.86	0.03	0.81	0.03	0.97	0.75	0.97	0.5	0.68	0.77	0.88	0.92	0.94	0.76
EBMDTR0	ON	0.89	0.02	0.89	0.03	0.83	0.83	0.94	0.9	0.85	0.93	0.83	0.94	0.93	0.94
EBMDTR1	OFF	0.9	0.02	0.88	0.04	0.87	0.83	0.96	0.7	0.9	0.89	0.89	0.94	0.89	0.92
EBMDTR1	ON	0.88	0.02	0.9	0.04	0.93	0.86	0.93	0.9	0.93	0.95	0.78	0.96	0.88	0.92

80% of the data is used for training and 20% is used for testing. The results are the average over 10 simulations. Two features are used VGG16 (ImageNet): FC6 and FC7. Two types of base learners are used: Decision Tree (DT) and Linear Support Vector Machine SVML

Table 26: Overall accuracy and average accuracy when using five features and two types of base learners.

MC	IACV	Overall		Average		Die					Epoxy			Others	
		OA	STD	AA	STD	CR	EPX	FM	FP	SC	EXS	FM	MIS	SUB	DC
SVML	OFF	0.89	0.01	0.83	0.03	0.94	0.93	0.99	0.56	0.71	0.89	0.85	0.93	0.97	0.5
SVML	ON	0.93	0.02	0.94	0.01	1	0.9	0.96	1	0.88	0.96	0.88	1	0.93	0.93
SVM	OFF	0.83	0	0.67	0.04	0.39	0.94	0.84	0.22	0.54	0.68	0.9	0.94	0.93	0.33
SVM	ON	0.91	0.02	0.93	0	1	0.91	0.87	1	0.88	0.95	0.88	0.91	0.93	0.97
RF	OFF	0.93	0.02	0.91	0.03	1	0.94	0.94	0.73	0.85	0.92	0.94	0.96	0.94	0.9
RF	ON	0.92	0.02	0.93	0.04	0.9	0.9	0.94	0.93	0.95	0.96	0.87	0.94	0.94	1
Bagging	OFF	0.88	0.03	0.87	0.05	0.94	0.92	0.91	0.78	0.83	0.87	0.86	0.91	0.86	0.87
Bagging	ON	0.89	0.02	0.92	0.01	0.89	0.89	0.95	1	0.88	0.94	0.78	0.93	0.91	1
RUSBoost	OFF	0.79	0.05	0.82	0.03	0.94	0.83	0.85	1	0.92	0.85	0.77	0.86	0.87	0.97
RUSBoost	ON	0.85	0.05	0.89	0.03	1	0.74	0.85	1	0.67	0.73	0.76	0.88	0.78	0.8
EBMDTR0	OFF	0.9	0.01	0.88	0.03	1	0.91	0.97	0.67	0.75	0.87	0.87	0.92	0.91	0.92
EBMDTR0	ON	0.9	0.02	0.92	0.03	1	0.89	0.91	0.93	0.88	0.93	0.83	0.95	0.89	0.98
EBMDTR1	OFF	0.88	0.03	0.86	0.02	0.93	0.83	0.98	0.8	0.65	0.81	0.85	0.93	0.91	0.92
EBMDTR1	ON	0.89	0.02	0.91	0.01	0.89	0.84	0.95	1	0.92	0.97	0.77	0.89	0.91	0.97

80% of the data is used for training and 20% is used for testing. The results are the average over 10 simulations. Five features are used, SIFT, AlexNet (ImageNet): FC6 and FC7, VGG16 (ImageNet): FC6 and FC7. Two types of base learners are used: Decision Tree (DT) and Linear Support Vector Machine SVML

Table 27: Overall accuracy and average accuracy when using two features and all types of base learners.

MC	IACV	Overall		Average		Die					Epoxy			Others	
		OA	STD	AA	STD	CR	EPX	FM	FP	SC	EXS	FM	MIS	SUB	DC
SVML	OFF	0.87	0.009	0.84	0.003	0.85	0.98	0.85	0.66	0.62	0.88	0.81	0.96	0.82	1
SVML	ON	0.9	0.007	0.89	0.002	1	0.84	1	0.68	0.88	0.81	0.88	0.93	0.9	1
SVM	OFF	0.91	0.007	0.87	0.002	0.86	0.89	0.88	0.67	1	0.81	0.88	0.86	0.98	0.85
SVM	ON	0.9	0.004	0.9	0.001	1	0.89	0.88	0.67	1	0.81	0.87	0.86	0.99	1
RF	OFF	0.91	0.008	0.89	0.003	1	0.89	0.93	0.67	0.88	0.82	0.9	0.97	0.89	1
RF	ON	0.91	0.007	0.9	0.002	1	0.89	0.89	0.67	0.88	1	0.81	1	0.9	1
Bagging	OFF	0.89	0.009	0.84	0.003	0.84	0.87	0.88	0.34	0.88	0.81	0.86	1	0.9	1
Bagging	ON	0.89	0.005	0.91	0.002	0.84	0.87	1	1	1	0.93	0.69	0.86	0.98	0.91
RUSBoost	OFF	0.85	0.006	0.89	0.002	0.84	0.84	0.89	1	0.88	0.93	0.66	0.97	0.84	1
RUSBoost	ON	0.9	0.009	0.92	0.003	1	0.84	0.84	1	1	0.81	0.96	0.93	0.9	0.9
EBMDTR0	OFF	0.91	0.005	0.94	0.002	1	0.81	0.96	1	1	0.82	0.86	0.97	0.99	1
EBMDTR0	ON	0.89	0.003	0.93	0.001	1	0.87	0.96	1	1	0.89	0.73	0.9	0.95	1
EBMDTR1	OFF	0.93	0.006	0.93	0.002	1	0.89	0.96	1	0.76	0.97	0.85	0.97	0.97	0.91
EBMDTR1	ON	0.9	0.006	0.93	0.002	1	0.79	1	1	1	0.89	0.86	0.89	0.92	1

80% of the data is used for training and 20% is used for testing. The results are the average over 10 simulations. Two features are used, VGG16 (ImageNet): FC6 and FC7. All types of base learners are used

4.8 Comparison with DNN-based Solutions

Deep Neural Network represents one of the most important advances that got significant attention in computer vision. More specific, Deep Convolution Neural Network CNN has been achieving the best results in computer vision competitions since 2012 when a CNN-based solution (ALEXnet) achieved the best results in the ILSVRC competition. Since then, CNN has been considered as

Table 28: Overall accuracy and average accuracy when using Five features and all types of base learners.

MC	IACV	Overall		Average		Die					Epoxy			Others	
		OA	STD	AA	STD	CR	EPX	FM	FP	SC	EXS	FM	MIS	SUB	DC
SVML	OFF	0.91	0.03	0.94	0.01	1	0.9	0.95	0.97	0.87	0.97	0.89	0.98	0.98	0.9
SVML	ON	0.89	0.03	0.93	0.01	0.99	0.97	0.94	0.87	0.85	0.95	0.84	0.97	0.96	0.98
SVM	OFF	0.51	0.03	0.37	0.008	0.01	1	0.82	0.31	0.01	0.16	0.66	0.45	0.2	0.02
SVM	ON	0.83	0.03	0.72	0.009	0.19	0.98	0.94	0.7	0.65	0.36	0.74	0.9	0.91	0.82
RF	OFF	0.93	0.01	0.86	0.003	1	0.95	0.97	0.7	0.51	0.81	0.95	0.97	0.97	0.81
RF	ON	0.95	0.01	0.95	0.004	1	0.95	1	0.92	1	0.97	0.88	0.86	0.99	0.91
Bagging	OFF	0.94	0.03	0.89	0.01	1	0.9	0.98	0.73	0.9	0.93	0.91	0.85	0.94	0.71
Bagging	ON	0.92	0.02	0.91	0.008	1	0.91	0.97	1	0.77	0.86	0.88	0.84	0.9	1
RUSBoost	ON	0.88	0.03	0.87	0.009	1	0.85	0.91	0.82	0.76	0.9	0.93	0.92	0.69	0.9
RUSBoost	OFF	0.94	0.02	0.93	0.007	1	0.83	0.94	0.92	1	0.9	0.92	0.98	0.92	0.92
EBMDTR0	OFF	0.89	0.02	0.89	0.006	1	0.8	0.84	1	0.76	0.86	0.81	0.91	0.99	0.91
EBMDTR0	ON	0.96	0.02	0.96	0.006	1	0.93	0.93	1	0.9	1	0.96	0.94	0.91	1
EBMDTR1	OFF	0.96	0.02	0.94	0.006	1	0.92	1	0.93	0.78	0.91	0.89	0.94	1	1
EBMDTR1	ON	0.98	0.03	0.95	0.01	1	0.98	0.99	1	0.9	1	0.81	0.97	0.98	0.91

80% of the data is used for training and 20% is used for testing. The results are the average over 10 simulations. Five features are used, SIFT, AlexNet (ImageNet): FC6 and FC7, VGG16 (ImageNet): FC6 and FC7. All types of base learners are used

a very important research topic in machine learning and computer vision. In this section, we will compare the performance of the BagStack classifier (the best results) to different CNN-based solutions. It is well known that training a new CNN model from scratch to solve a specific machine learning problem is very expensive in the context of the need to have large number of images (most of the time millions of images) and the computational complexity. We will compare our results to CNN-based results using three different ways. (1) using pre-trained well-known models as features extractors, and then the extracted features are used to train different classifiers (e.g., SVM, Random Forest, … etc.). (2) Fine-tuning well-known pre-trained models using our data. (3) training our own model using our data.

We used six different CNN models, AlexNet, VGG16, VGG19, GoogleNet, ResNet50 and ResNet152. In the first experiment, we use these models as features extractors. Since these models are trained to predict images from the ImageNet data base, the last layer (softmax) has 1000 outputs, one per each class. Instead of reading the output of the last layer, we read the outputs of the intermediate layers which represent different representations of the input image. These representations can be seen as features that are extracted from the image. For each one of these models, we optimize the overall accuracy and the average per-class accuracy across (1) different features representations (the output of different layers), (2) different classifiers (DT, SVM, SVM(RBF), Random Forest, Bagging, AdaBoost.M2, RUSBoost and Viola-Jones Classifier), and (3) using data-augmentation in training and testing.

In the data augmentation process, each image has been processed using the means of image processing to generate other images. We perform rotation (30, 60, 90, …. 360), flip vertical and horizontal, scale down (0.9, 0.8 and 0.7) and scale up (1.1, 1.2 and 1.3) followed by cropping where we crop the upper left corner, upper right corner, lower left corner, lower right corner and the center.

In total, for each image we generate additional 32 images which results into 33 images including the original image. We tested applying data augmentation in the training process and / or data augmentation in the testing process. When applying data augmentation in the training, it is very important to know that all different versions of the same original sample are bundled together. The sampling is applied at the original samples and all samples belonging to the same original one will move together in the training / testing processes. In the testing process, the final prediction of a sample is calculated as the average of predictions (voting) of the samples belonging to this sample. Figure 28 depicts some of the images that are generated by using data augmentation.

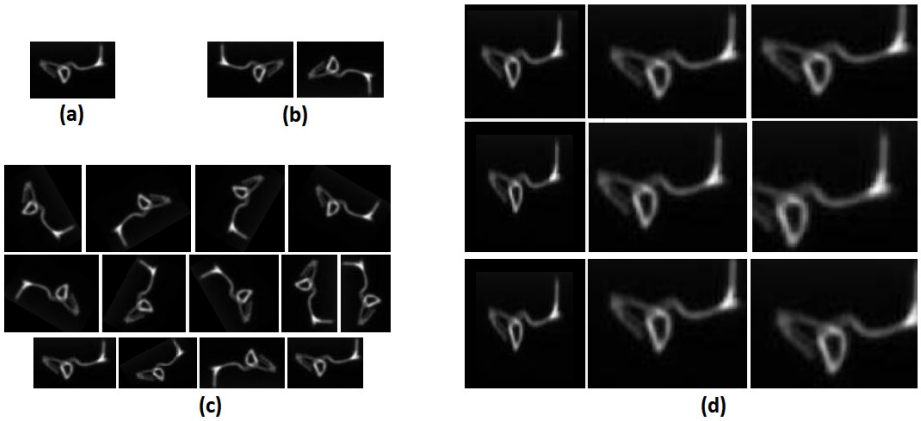

Figure 28: Sample images of data augmentation: (a) The original image, (b) Flip horizontal and vertical, (c) rotating the image and (d) scale down and scale up followed by cropping the four corners and the center.

We also fine-tuned some of these models using fine tuning to solve our problem. Fine-tuning the models to solve new problems necessitates changing the architecture of the network and re-training the model. We change the last layer (softmax), instead of predicting 1000 outputs (the number of classes in the image net) we use a softmax layer to predict 10 outputs (the number of classes in our problem). Data augmentation is used again in the training and the testing processes. We fine-tuned AlexNet, VGG16 and GoogleNet. Stochastic Gradient Descent is used for training starting with (0.001) learning rate. AdaDelta method is used to control the learning rate. The used batch size is 16. The data is divided into 80% learning and 20% testing. The 80% learning is divided into 70% training and 10% validation.

Finally, we train also our own model. Since the available number of images is small, we kept our model very simple to avoid overfitting problems. We used data augmentation in both training

and testing processes. Our network contains 2-convolution layers, each one is followed by linear rectification and max pooling layers, 2-fully connected layers, each one is followed by a drop-out layer and finally a softmax layer. Stochastic Gradient Descent SGD is used for training starting with (0.01) learning rate. AdaDelta method is used to adaptively change the learning rate. The used batch size is 16. The data is divided into 80% learning and 20% testing. The 80% learning is divided into 70% training and 10% validation.

Table 29 depicts the comparison results. The results are the average of 5-times when using the models as a feature extractor, 5-times for the BagStack classifier and 3-times when we fine-tuned the models to our own problem and when we train our own model. It is important to know that five different features are used when training the BagStack classifier, 4 of them are DNN-based features (2 features are extracted from AlexNet and 2 features from VGG16) and SIFT feature. We use all learning algorithms listed in Table 24 as base learners.

Table 29: Comparison of different classification methods in terms of overall and average classification accuracy.

	Specifications		Overall Accuracy		Average Accuracy	
Feature Name	Augmentation	Classifier	OA	STD	AA	STD
REF-19-FC7	Yes	AdaBoostM2(SVM)	94.94%	0.60%	96.46%	0.38%
VGG16-33	Yes	ViolaJones(SVM)	95.64%	0.24%	94.48%	2.37%
VGG19-39	Yes	AdaBoostM2(SVM)	94.51%	1.12%	95.78%	0.88%
GoogleNet-CLS1-FC2	Yes	ViolaJones(SVM)	94.09%	1.69%	94.14%	2.10%
ResNet50-FC1000	Yes	SVMRBF	95.70%	0.35%	93.66%	3.21%
ResNet152-FC1000	Yes	ViolaJones(SVM)	92.97%	0.49%	94.70%	0.50%
REF-ALEX	Yes	softmax	95.60%	1.30%	95.80%	1.40%
VGG16	Yes	softmax	94.90%	3.50%	93.30%	4.00%
GoogleNet	Yes	softmax	95.00%	4.50%	93.20%	3.00%
Training New Model	Yes	softmax	95.20%	5.20%	93.92%	2.30%
BagStack Classifier	No	-	97.50%	3.00%	95.30%	1.00%
BagStack Classifier	Yes	-	98.63%	2.60%	98.02%	0.81%

4.8.1 Conclusion

In this chapter, we present a multifeature sparse-based defects' characterization framework. To the best of our knowledge, this is the first work that uses machine learning to address the problem of defects detection and classification at the level of semiconductor units. The proposed framework consists of an enhanced multifeature sparse-based representation and a stacking-based classifier. The proposed data representation is augmented with background features subtraction to enhance the classification accuracy when very few samples are available. The proposed stacking-based classifier integrates a novel adaptive sampling technique. Ensemble pruning and metadata over-sampling effectively deal with the data imbalance and enhance the quality of the metaclassifier.

We tested the performance of the proposed approach on real images from Intel. The proposed approach was shown to result in a high classification accuracy by using few numbers of images for training comparing with other well-known ensemble techniques that are designed specifically to deal with data imbalance problem. The proposed system is able to achieve a high classification accuracy even when a small number of samples are used for training (30%) and a high level of scalability in the sense that adding more features can only improve the accuracy. The proposed system can achieve 95.88% average classification accuracy when only 30% of the data is used for training and an average accuracy of 98.5% when 80% is used for training (Table 22).

In the last section, we compare the BagStack classifier to different DNN-based classification methods. We consider three different ways of using deep learning in image classification: using pre-trained models to extract features, fine-tuning pre-trained models and training our own model. Due to the fact that the available number of images is very limited, we use data augmentation. For each image another 32 images are extracted. The results in Table 29 shows that BagStack classifier outperforms all other classification techniques.

Chapter 5

LOCALLY ADAPTIVE STATISTICAL BACKGROUND MODELING WITH DEEP LEARNING BASED FALSE POSITIVE REJECTION FOR DEFECTS DETECTION IN SEMICONDUCTOR UNITS

In this chapter, A system for the detection and classification of defects in semiconductor units is presented. The proposed system consists of three stages: proposal generation stage, defect detection stage and refinement stage. In the proposal generation stage, changes on the target unit are detected using a novel change detection approach. In the second stage, a deep neural network is used to classify detected regions into either defective or non-defective regions. Non-defective regions are regions exhibiting allowable changes due to factors such as lighting conditions and subtle differences in manufacturing. The defect detection stage was adopted from [174] and achieves up to 94.3% accuracy. In practice, defects that are smaller than a specified tolerance size are ignored by manufactures. The tolerance size depends on the defect types and is determined based on risk factors. In order to ignore such defects, our approach includes a final refinement stage wherein the detected defects are categorized by a stacking-based ensemble classifier into different classes. Defects smaller than their corresponding tolerance size are ignored. The refined system achieves up to 97.88% overall detection accuracy.

This chapter introduces an automated system for defects detection in semiconductor units including die, epoxy and substrate regions. The presented system is immediately applicable to different types of defects on die, epoxy and substrate. Inputs of the system can be either color or grayscale images.

5.1 Introduction

Automated inspection systems are used in manufacturing to control the quality of products, reduce product waste, and improve efficiency. Inspection is an important process to detect defective products and keep these from reaching customers. Non-contact optical sensing systems are considered that use a camera to monitor the quality of products. Optical inspection systems do not require physical contact with the inspected material and can perform at high speed. However, optical meth-

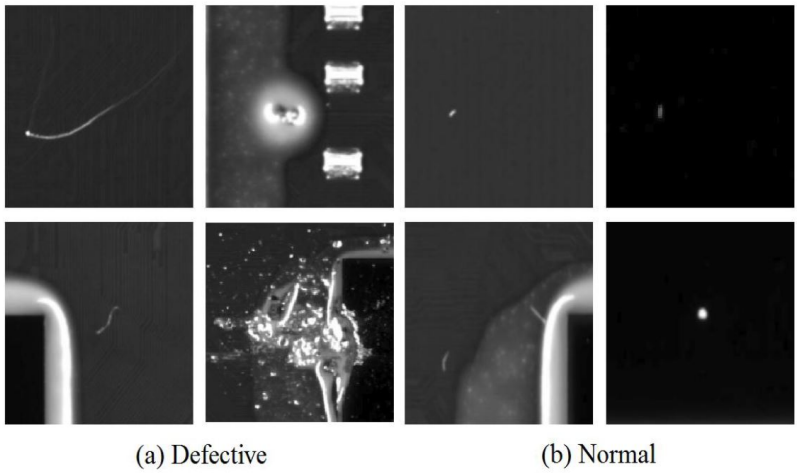

Figure 29: Examples of (a) defective samples and (b) non-defective samples. It can be seen that some of the non-defective regions exhibit small changes due to small foreign materials or epoxy dots. These small changes are small in size and can be ignored.

ods bring additional challenges because of the intrinsic variability in captured images, e.g., lighting conditions and manufacturing differences. State-of- the-art defect detection systems are sensitive to such variability in the sense that they are not able to distinguish allowable non-defective variations from actual defects.

In this chapter, optical-based detection and classification of defects on semiconductor units is considered. There are several unique challenges associated with this problem domain. Firstly, there is a significant amount of intrinsic variability from unit to unit. This variability is most evident in the epoxy region, where it is difficult to control the manufacturing process such that the epoxy is the same for each unit. Secondly, the sizes of defects are often very small in relation to the size of the entire unit. Finally, due to the current progress of the semiconductor manufacturing process, defects are less likely to occur, but it is still important to detect these defects. Figure 29 depicts examples of defective and non-defective regions. The defects exhibit a wide range of variability. Defects can occur because of foreign materials, damaged components, scratches, cracks and epoxy related problems. The proposed system can handle these defects regardless of the location in the unit. In addition to this, there are many small manufacturing variations that would not be considered as defect. Often these variations look like defects but differ in size. In this work, sixteen different products and ten different types of defects are considered.

Semiconductor units consist primarily of four different parts: die(s), substrate, epoxy and com-

ponents. While the die is mostly a uniform region, the substrate and the epoxy are more complex textures. Epoxy material is sensitive to light variations, which creates different types of acceptable variations. The work in this chapter represents an extension to our previous work in the previous chapter [175]. In [175], only defects classification on the die area is considered. This work is concerned with defect detection over all the unit (die, epoxy and substrate). While the detection of defects on the die area is relatively straightforward due to the uniformity of the die region, the detection of defects on the substrate and epoxy regions is more challenging. Our system can detect and recognize defects on die, substrate and epoxy.

A multi-stage system for defect detection and classification is proposed that can handle many of the aforementioned difficulties. In the first stage, change detection is performed wherein potential defects' locations are generated. The proposed change detection process (proposal generation stage) consists of two steps: modeling and subtraction. In the modeling step, a set of reference images (non-defective units) are used to generate a statistical-based locally adaptive abstraction model to describe the perfect units. In the subtraction step, the target unit is compared to the model pixel-by-pixel to generate the change detection mask. In some cases, the detected changes, which are related to regions that are significantly different from the corresponding regions in the non-defective units, are not true defects. For example, some changes due to light and epoxy variations are acceptable changes. For this reason, each detected region is further classified into either non-defective region (false positive) or defective region (true positive) using an adopted DNN-based classifier (defect detection stage) [174]. In the refinement stage, in order to eliminate false positives that are smaller in size than a tolerance size, a stack-based classifier [175] is used to classify defects into different classes. Next, corresponding to different defect types, a type-dependent size threshold is used to filter out tolerated small-size defects.

The contributions of this chapter can be summarized as follows. 1) This work represents the first work to consider defects detection and classification in semiconductor units including die(s), epoxy, substrate and components. To our knowledge, this is the first work to use machine learning to detect and classify defects in semiconductor units. 2) In this chapter, a locally adaptive statistical background modeling for change detection is proposed.

The rest of this chapter is organized as follows. Section 2 discusses previous and related works. An overview of the proposed defect detection and classification system is presented in Section 3. Section 4 presents a detailed description of the change detection process including image repre-

sentation, statistical background modeling and change detection. Section 5 presents briefly the adopted deep neural network-based defect detection stage to distinguish between defective and non-defective regions. This defect detection stage was developed by collaborator Samuel Dodge at the ASU IVU Lab. The refinement stage is presented in Section 6. Performance evaluation results are presented in Section 7 followed by a conclusion in Section 8.

5.2 Problem Formulation and Related Work

Several existing computer vision methods have been presented for defect/change detection [176, 177, 178, 179], object detection [180], and background subtraction [181, 182, 183, 184, 185, 186, 187]. While the problem of interest in this chapter is a defect detection and classification problem, it has its own challenges that make other related methods in the literature not able to achieve acceptable performance. In this section, some of the state-of-the-art methods in this domain will be covered, explain how they are connected to our problem and their limitations.

In this chapter, the problem of defect detection and classification in the semiconductor fabrication process at the unit level is addressed. In the literature, many approaches were proposed to deal with this problem at the wafer level [188, 189, 190, 191, 192] but not at the unit level. While two wafers, belonging to the same product, should be mostly identical, two units belonging to the same product may exhibit some acceptable variations due to subsequent processing steps. Mainly, all subsequent steps after wafer fabrication, cutting, adding components and using epoxy to glue die onto the substrate, may introduce different types of variations. Discriminating between acceptable variations and defects is challenging at the unit level.

The problem of defect classification in the die region of semiconductor units was previously studied in [175]. In [175], no automated defect detection is presented, rather than the inputs are manually cropped from unit images. This ensures that defects are nicely centered in the input images. By contrast, our proposed system performs defect detection and classification, which is a more difficult problem. In [175], the authors addressed the defect classification problem on the die while our work addresses the detection and classification problem on the die, epoxy and substrate. In our system, the only human-provided input is the training dataset consisting of ground-truth defective and non-defective samples. After training, the system automatically detects, crops and classifies defects. Our approach is similar to the RCNN [180] paradigm for detecting objects in natural im-

ages. In this work, instead of objects we are looking for defects, and instead of natural images, semiconductor units are being considered. In RCNN, for object detection, selective search [193] is used to generate candidate boxes that could contain objects. In our case, such selective search fails to find meaningful objects because defects occupy a very small region of the image, and do not exhibit characteristics like objects in a natural scene. When we attempted to use selective search [193], we found that most of the generated boxes centered around the components of the unit rather than the defects. Instead of selective search, we propose in this work a statistical-based change detection method based on statistical background modeling as described later in Section 4. After detection, RCNN uses a deep neural network (DNN) fine-tuned from a pre-trained neural network to perform classification. A DNN is also used here; however, the proposed DNN is a unique architecture and has been trained from scratch due to major differences between natural images and semiconductor images. Furthermore, the proposed DNN architecture is not applied to the semiconductor input image directly but rather it is applied to regions in the image where significant changes from a reference normal (non-defective) unit model are detected. These candidate defect regions are generated by our novel proposed statistical-based change detection stage.

In change detection algorithms, the goal is to find the differences between two or more images. However, depending on the application domain, the algorithm can be quite different. Many applications try to separate the foreground from background in video. This class of approaches is called background subtraction. In [176], the authors present a review of state-of-the art methods for background subtraction. Another application domain is to detect changes between two images [177, 178] (e.g., remote sensing images [179]). In this later application, the goal is to detect changes in a pair of images. However, our problem contains certain intrinsic variabilities (such as epoxy changes) that give unsatisfactory results when using existing or generic “off-the shelf” change-detection algorithms.

Background subtraction methods learn a model of the background for different locations in the image, and then compare a pixel to the learned background model to determine whether it is foreground or background. This background model should be flexible enough to capture all acceptable and expected variations yet able to adapt to new variations. The model can be learned at the block level [181] or at the pixel level [182]. The pixel level provides a more accurate result; however, the computation and storage requirements are much greater. In existing statistical-based background modeling, the background model can be a parametric model, such as a single Gaussian [183] or a

mixture of Gaussians [184], or a non-parametric model such as using a codebook to represent the background [185]. Background subtraction methods can be applied directly to the raw images using pixel intensities to learn a background model [184] or to other image representations, e.g., texture features. In [181], local binary patterns (LBP) [186] are used as texture features to capture background statistics. LBP features are robust to changes in illumination. Going a step further, intensity information and texture information can be combined [187]. The authors of [187] proposed a multi-layer background subtraction technique which combines LBP features with color measurements in the RGB color space.

Another related field is change detection in remote sensing images [179]. In remote sensing, there are often only two images, corresponding to two different time instances, to compare. With a lack of more images, using background subtraction related methods cannot accurately model the statistics of pixels. To compensate for the lack of temporal information, remote sensing algorithms often make use of spatial information. The first step in change detection algorithms is to produce a difference image that gives the likelihood of a change. Among the earliest methods to find a difference image is to use a simple subtraction operation [179]. An alternative to subtraction is the ratio method which is less susceptible to illumination and other small changes in the images [194], [195]. Change Vector Analysis (CVA) [177] uses spectral information to map the change magnitude and direction. Principal component analysis was also adopted to produce an initial change detection mask [195]. The methods proposed in [196] and [197] use a mixture of Gaussians to model the local statistics around a pixel and the Kullback-Leibler divergence is used to compute the difference. Once a difference image has been obtained, then the problem of change detection reduces to an image segmentation problem. This segmentation can be performed using thresholding [198, 199] or clustering methods [200, 201, 202, 203].

In this chapter, a change detection algorithm is proposed designed specifically for defects detection in semiconductor units and which integrates both image intensity and local binary patterns to describe the local image characteristics. Describing the image features in terms of a simple parametric model as in [183] cannot accurately describe the statistics. A non-parametric model based on a set of histograms learned from the data using the K-means clustering algorithm is adopted.

The refinement stage adopts the stacking-based classifier proposed in our previous work [175]. The lack of defective images and the data imbalance problem represent the main challenges that the stacking-based classifier [175] is designed to deal with. The classifier integrates a stacking-

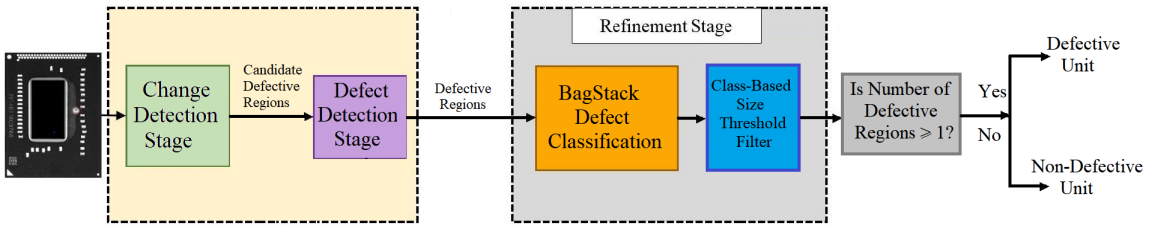

Figure 30: Workflow of the proposed system.

based ensemble method with adaptive sampling technique to achieve high accuracy under these circumstances. By contrast, under these same circumstances, it is difficult for a deep neural network to learn a function that does not overfit the training samples or that can perform well on the unseen defective samples that are not part of the training set.

5.3 System Overview

The proposed system consists of three stages: change detection stage, defect detection stage and refinement stage. Figure 30 depicts an overview of the proposed system. The proposed system uses statistical background modeling and data mining to perform defect detection and classification. The change detection process consists of two main steps: background modeling, wherein a locally adaptive statistical background modeling is proposed to model the background (i.e., the non-defective unit), and background subtraction, wherein the target image is processed for change detection. The model describes non-defective imaged units and captures different possible variations. A statistical model of images of the non-defective units at each pixel is learned. Detecting changes is done by comparing target images to the learned background model pixel-by-pixel.

The change detection mask is used as input to the defect detection stage. The defect detection stage uses a deep neural network (DNN) to determine whether detected changes are defects or ignorable ones. The adopted DNN architecture is a multi-column architecture that has several input slices (consisting of the target unit's image as the first slice followed by the images of non-defective units). The network learns to distinguish between acceptable changes and defective regions. Each candidate defective region is processed by the defect detection stage (one-by-one) and classified into either defective or non-defective region. The network has been trained using labeled detected regions. In some cases, multiple neighboring regions are detected for the same defect (for example,

this is common in the case of the fingerprint on die defect). These latter regions, corresponding to the same defect, are considered defective regions.

In the refinement stage, a stacking-based ensemble classifier is used to classify defects into different classes. Based on a defect's type and size, these defects are considered as either acceptable defects or not. Using a single size threshold for all the different defect types may lead to many false positive alarms. However, using a different class-specific size threshold for each type of defect can effectively reduce false positive alarms. More details about the proposed methods are presented in Sections 4 to 6.

5.4 Change Detection Stage

The change detection process is the first step in the proposed framework. It consists of two major sub-steps: offline modeling and online detection. In the modeling process, a set of images corresponding to non-defective units from the same product line is used to build an abstract representation of imaged non-defective units. The modeling process results in a background model consisting of $W \times H$ sub-models, one at each pixel location, where W and H are the width and height of the reference images. Each pixel has a corresponding non-defective "background" model that captures all possible variations at this pixel location across all reference images. In the detection process, target images are compared to the derived background model pixel-by-pixel using a proposed change detection procedure. The proposed procedure is a nonparametric method that uses both temporal and spatial information to improve the detection accuracy. The change detection procedure measures the deviation of the target pixel from the corresponding sub-model at the same pixel location.

5.4.1 Image Representation

Designing a robust change detection solution starts with the image representation that is used through the modeling and detection processes. While the simplest solution is to use intensity values directly, this solution may not be able to cope with illumination and noise variations, which lead to false positives. Instead, both normalized intensity values and texture features based on local binary patterns (LBP) are used. First, change detection using these two representations separately is

applied, and then a spatial-based merging algorithm is employed to combine the detected changes resulting from each representation.

When using normalized intensity values, each pixel is normalized by subtracting the mean computed in a $W_{norm} \times W_{norm}$ window around it. This normalization is necessary to handle local acceptable variations between different units. Selecting an optimal W_{norm} depends on the input image resolution and the defect type. Selecting a large normalization window results in losing the ability to adapt to local image statistics. Selecting a small normalization window results in a high correlation between pixels, in which case normalization will result in a representation with values very close to zero. The values of the normalized pixels depend on the pixel's location and neighboring image characteristics. For example, edge pixels or pixels that are close to edge pixels have relatively large absolute values pixels on opposite sides of an edge will have different signs (+ve, -ve), and pixels belonging to uniform regions (e.g., die) will have values close to zero. Figures 31 (a) and 31 (b) illustrate the normalization process.

Using intensity normalization is very robust against light variations and thus results in a smaller number of false positive alarms compared to using raw intensity values. However, some defects have intensity values close to the background (e.g., scratch on substrate). The detection of such defects in the normalized intensity domain is challenging. Local Binary Patterns (LBP) [186] are widely used to represent local image characteristics for a variety of applications including background subtraction because of their computational efficiency, and effectiveness in representing local image variations. The LBP representation, in its simplest form, is a texture descriptor that encodes the relationship between a central pixel and neighboring pixels within a pre-defined neighborhood, into a sequence of binary values (a binary pattern). If the difference between pixels is greater than or equal to zero, the binary value is set to be one and if the difference is less than zero, the value is set to be zero. A modified version of the basic LBP called threshold-based LBP is used. Threshold-based LBP requires the difference to be higher than a threshold t_{lbp} to set the value to zero. This makes the LBP robust against small variations due to noise. To compute local binary patterns, a $W_{lbp} \times W_{lbp}$ pixel square pattern with eight neighbor pixels is utilized. LBP works very well in detecting small defects with sharp edges that introduce texture changes. In choosing W_{lbp} and t_{lbp}, we optimize for better detection rate, at the expense of more false positives. These false-positives are filtered out in the subsequent steps.

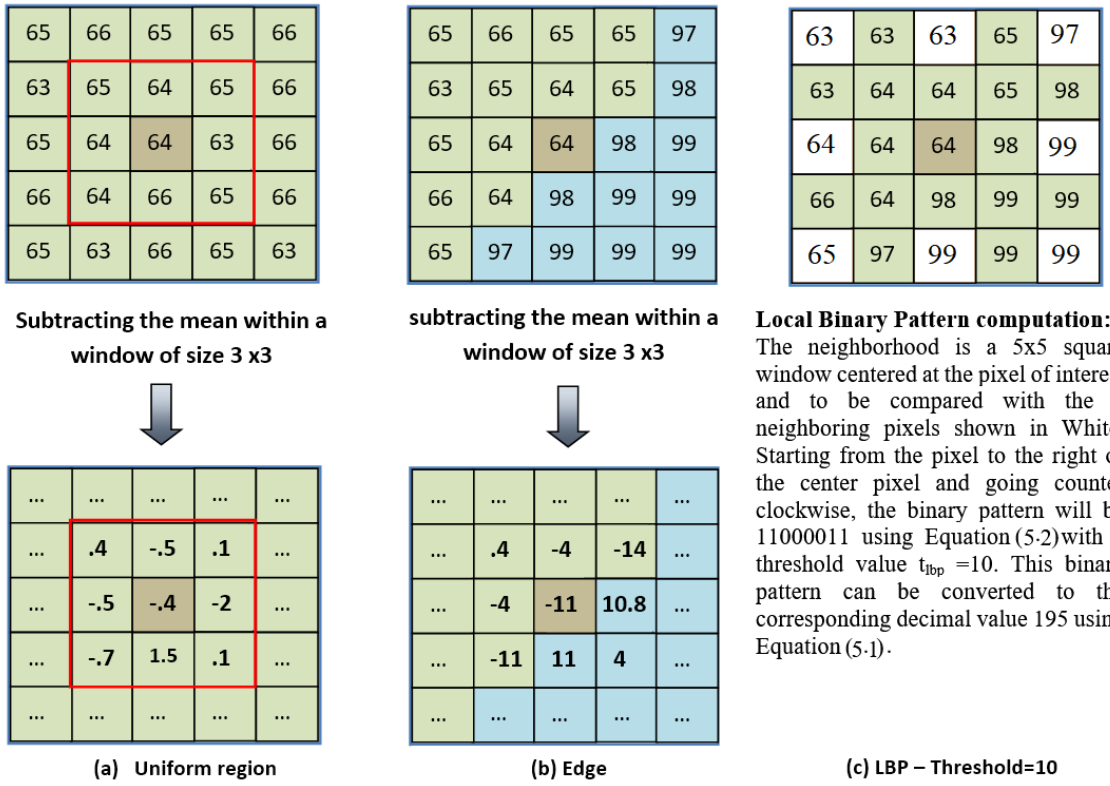

Figure 31: Intensity-based and LBP-based image representations. Behavior of the intensity-normalized region representation in slow varying regions (a) and near edges (b). For uniform regions, the output values are very small. For edge regions, the output is very large along the edge. (c) Computation of the LBP features.

Let $P_{i,j}$ be the intensity of a pixel at location (i, j) and let P be the total number of its considered neighboring pixels $(e.g. P = 8)$ in Figure 31.c. The value of the LBP code of the pixel $p_{i,j}$ is given by:

$$LBP_{i,j} = \sum_{z=0}^{P-1} S(p_z - p_{i,j}) \times 2^z \tag{5.1}$$

Where p_z is the intensity of the z^th neighbor pixel and

$$S(p_z - p_{i,j}) = \begin{cases} 1, & |p_z - p_{i,j}| \geqslant t_{lbp} \\ 0, & |p_z - p_{i,j}| < t_{lbp} \end{cases} \tag{5.2}$$

In (1), the P neighboring pixels P_z are scanned in a counterclockwise fashion with $z = 0$ corresponding to the starting pixel to the right of the center pixel $p_{i,j}$ and $z = P - 1$ corresponding to the last scanned pixel. Figure 31 (c) illustrates the process of LBP feature extraction.

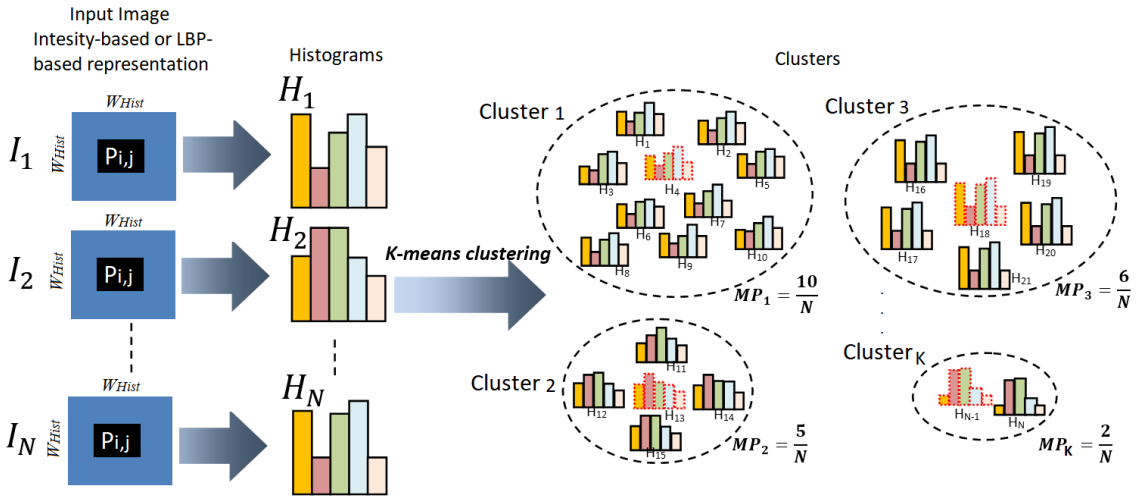

Figure 32: Background Modeling. Background modeling consists of three main steps. 1) Histogram construction; 2) Applying K-means on the histograms; 3) Finding the mixing percentage of each cluster. K is the number of clusters; $I_1, I_2, ..., I_N$ are the input-reference images; $P_{i,j}$ is the pixel location (i, j) for which N histograms are computed, one for each image $I_n, n = 1, ..., N$, using spatial neighborhood $W_{Hist} \times W_{Hist}$. For each cluster i, the centroid i and the mixing percentage MP_i are calculated.

The threshold-based LBP is not invariant to rotations of the input image. Image registration may result in small rotations. As the LBP patterns are obtained by circularly sampling around the center pixel, rotation of the input image results in significantly different $LBPs$. Since image registration represents a pre-processing step in our framework (Section 4) a rotation-invariant threshold-based LBP is needed. For this purpose, a uniformity measure U is used as in [186]. U is the number of bitwise transitions from 0 to 1 or vice versa when the bit pattern is considered circular. A local binary pattern is called uniform if its uniformity measure is at most $U_{th} = 2$. The value of the uniform LBP code of a $pixel$ $P_{i,j}$ is given by:

$$LBP_{i,j}^{U_{th}} = \begin{cases} \sum_{z=0}^{P-1} S(p_z - p_{i,j}), & U(LBP_{i,j}) \leq U_{th} \\ P + 1, & Otherwise \end{cases} \tag{5.3}$$

where:

$$U(LBP_{i,j}) = \sum_{z=1}^{P} |S(p_z - p_{i,j}) - S(p_{z-1} - p_{i,j})| \tag{5.4}$$

Such uniform-based LBP descriptor is robust to noise and adds an extra level of rotation and grayscale invariance.

Using each of the intensity-based and LBP-based representations, a binary change detection mask can be generated for each representation as described in Section 4.B where a value of 1 at

a pixel location $(i,\ j)$ denotes that a change is detected at that location. The final change detection mask is generated by combining the detection masks corresponding to each representation as described in Section 4.C.

5.4.2 Learning Models and Change Detection

Figure 32 illustrates the proposed framework for learning a background ("non-defective" unit) model. The learned model is subsequently used as part of the change detection stage. The input images, consisting of N reference images (i.e., images of non-defective units) are registered first to ensure alignment. Alignment is necessary to improve the quality of the background model and reduce the false positives. The Enhanced Correlation Coefficient maximization algorithm (ECC) [204] from the MATLAB Image Alignment Toolbox [205] is used to perform registration. The ECC algorithm is an extension to the Lucas-Kanade [206] algorithm but has the desirable characteristic of being invariant with respect to photometric distortions. One of the reference images is randomly selected to be the reference for the alignment.

A general formulation for change detection is learning a background model M wherein M consists of $W \times H$ sub-models $M_{i,j}$, one for each pixel location (i,j), and determining how well a pixel representation $x^{'}_{i,j}$ at a location (i,j) fits the sub-model $M_{i,j}$. $x^{'}_{i,j}$ could be a normalized intensity or LBP representation. The result of the change detection process is a binary image Z (detection mask) given by:

$$Z_{i,j} = \begin{cases} 1, & f(x^{'}_{i,j}|M_{i,j}) > \lambda \\ 0, & f(x^{'}_{i,j}|M_{i,j}) \leq \lambda \end{cases} \tag{5.5}$$

where $f(x^{'}_{i,j}|M_{i,j})$ is a function that measures the deviation of the pixel $x^{'}_{i,j}$ from the corresponding sub-model $M_{i,j}$ at the corresponding pixel location (i,j). The sub-model $M_{i,j}$ is learned using samples $x^{'}_{i,j}{}^{1\cdots N}$ from all reference images $(I_1\ ...\ I_N)$ at the same location (i,j). N is the total number of reference images used to train the model. Using N reference images in this manner is similar to the concept of using temporal information across a sequence of video frames.

Motivated by remote sensing algorithms [194, 195, 197, 203], where only two images (reference and target) are available, spatial information is also being used within each image to increase the robustness against noise, light variations and registration error. For each pixel $x^{'}_{i,j}$ in an image, we use all pixels in the same image and within a window $W_{hist} \times W_{hist}$ centered at the location (i,j)

to construct a histogram. One can think about this histogram as a feature that describes the pixel (i, j) in the considered image.

Histograms of pixels (intensity-based or LBP-based representations) can be constructed by either using a uniform or non-uniform quantization. In uniform quantization, the distance between bins are equal while in the non-uniform quantization distances can be different. In the proposed approach, two different types of quantization are used: a non-uniform quantization for the normalized intensity values and a uniform quantization for the LBP representation. Figure 33 shows both uniform and non-uniform histogram bins. Due to spatial correlation, the majority of the normalized intensity pixels have values close to zero. For this reason, the number of bins assigned to values close to zero is higher in the histogram to capture small intensity variations. This improves the discrimination capabilities of the histogram descriptor.

The background sub-model $M_{i,j}$ at pixel location (i, j) is learned using the histograms $H_{i,j}^{1,..N}$. Let $H_{i,j}^{n}$ be the constructed histogram at location (i, j), where n, $1 \leq n \leq N$, is the index of the reference image and (i, j) is the spatial pixel location. For example, the model $M_{i,j}$ could be taken as the average of the histograms at location (i, j); however averaging histograms can lead to a less discriminative smooth histogram. An alternative is to model the histograms using a mixture of Gaussian (MOG) distributions.

In this thesis, a nonparametric modeling technique is proposed that uses the K-means clustering algorithm. $K-means$ clustering is performed on the histograms $H_{i,j}^{1,..N}$ to generate a set of $K << N$ histograms $\widehat{H}_{i,j}^{1}, \widehat{H}_{i,j}^{2}, ..., \widehat{H}_{i,j}^{K}$, where K is the number of clusters. These histograms are obtained by computing the centroid of each cluster. These histograms capture all significant variations at a pixel location (i, j).

Typically, the $K - means$ algorithm uses Euclidean distance to measure distances between different data points. In the proposed approach, the data points are histograms. The distance between histograms can be measured using the symmetric form of the Kullback-Leibler Divergence (KLD), given by:

$$D_{SKLD}(Q_1 \| Q_2) = \frac{D_{KLD}(Q_1 \| Q_2) + D_{KLD}(Q_2 \| Q_1)}{2} \tag{5.6}$$

where Q_1 and Q_2 are two probability distributions (normalized histograms), and D_{KLD} is the KLD. Figure 32 depicts the process of using N reference images to fit a model corresponding to the location (i, j). Compared with averaging or mixture of Gaussians, our method can more accurately model the data (Figure 34).

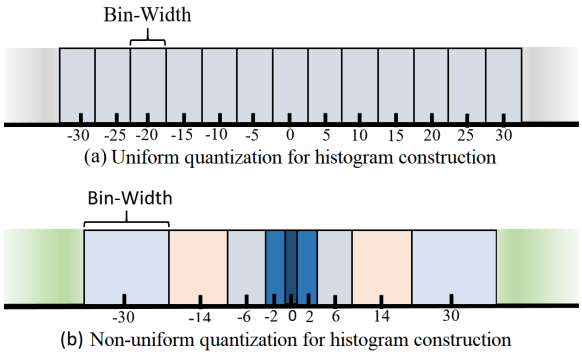

Figure 33: (a) Uniform and (b) Non-uniform histogram bins.

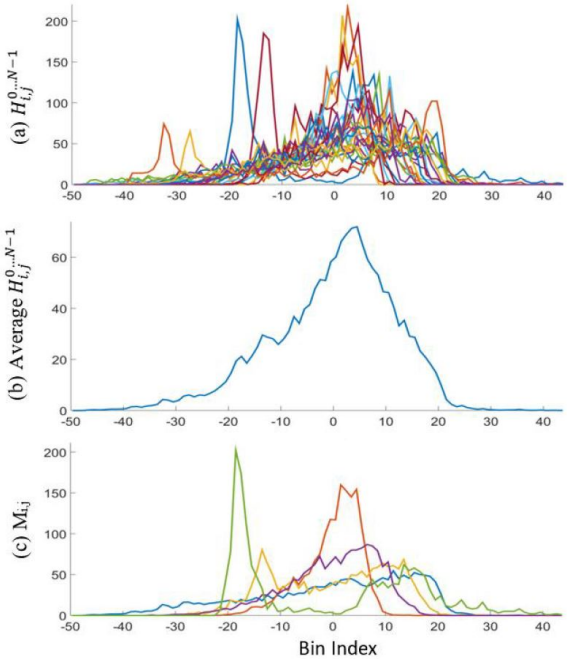

Figure 34: Model generation using K-means clustering. (a) Histograms from 25 patches drawn from the same location in $N = 25$ reference images; (b) average histogram; (c) five cluster centroids "histograms" learned from the data using the proposed method. Compared with averaging, the cluster centroids can better represent the local data characteristics.

The similarity between reference images typically result in relevant histograms being grouped into a relatively small number of clusters after applying the SKLD-based k-means algorithm. Other clusters would contain the less relevant noisy histograms. To avoid using clusters that capture noise related variations, the mixing percentage MP_k is defined as:

$$MP_K = \frac{N_K}{N} \tag{5.7}$$

where MP_k is the mixing percentage of the cluster k, N_k is the number of histograms belonging

to cluster k and N is the total number of histograms (which is equal to the number of reference images). MP_k is used as a measure of confidence of how much cluster k represents background or acceptable variations.

Given a target image T, to perform change detection at pixel (i,j), first the histogram $H^T_{i,j}$ at pixel (i,j) is first computed in target image T using all pixels within a window $W_{hist} \times W_{hist}$ centered at pixel (i,j). Then, the $SKLD$ distance between $H^T_{i,j}$ and each of the centroid histograms $\widehat{H}^k_{i,j} \in M_{i,j}\ s.t\ MP_k > \alpha$ is computed, and determine the minimum computed distance (f). The minimum distance (f) measures how different the target patch is from the reference patches. This measure f is used to compute a binary change mask Z as follows:

$$Z_{i,j} = \begin{cases} 1, & f(H^T_{i,j}|M_{i,j}) > \lambda \\ 0, & f(H^T_{i,j}|M_{i,j}) \leq \lambda \end{cases} \tag{5.8}$$

where

$$f(H^T_{i,j}|M_{i,j}) = \underset{\forall\ \widehat{H}^k_{i,j} \in M_{i,j}\ s.t.\ MP_k > \alpha}{min} D_{SKLD}(H^T_{i,j} \| \widehat{H}^k_{i,j}) \tag{5.9}$$

where $MP_k = \frac{N_k}{N}$ is the mixing percentage of each cluster k, $\widehat{H}^k_{i,j}$ is the centroid of cluster k in model $M_{i,j}$, $H^T_{i,j}$ is the computed histogram in the target image at the location (i,j), N_k is the number of histograms belonging to cluster k, N is the total number of reference images used to fit the model, and α is a parameter that is used to ignore clusters that contain a small number of reference histograms.

Our model is a hybrid of traditional background subtraction models and change detection models used for remote sensing. Background subtraction models consider only $x^{1\ldots N}_{i,j}$ for fitting the model. $x^{1\ldots N}_{i,j}$ represents the temporal information at the pixel (i,j). On the other hand, the change detection models, which typically consider two images (reference and target), incorporate spatial information. Consequently our model uses both spatial and "temporal" information because the model $M_{i,j}$ is determined using the spatial information (histograms) from all reference images: $H^{1\ldots N}_{i,j}$. These reference images provide information that is similar to the temporal information provided in video-based background subtraction.

5.4.3 Mask Fusion

From the previous step, two change detection masks are obtained: one based on normalized intensity features (Z_I) and one based on LBP features (Z_{LBP}). The two change detection masks

are complementary: the LBP feature-based mask helps in detecting potential defective regions (e.g., defective regions with smooth edges) that the normalized intensity-based mask misses, and vice versa. If a simple logical OR operation is used to merge the two masks, the false positives of the two methods will be combined. Similarly, if a logical AND operation is used, any region that is only detected by a single method is removed.

A heuristic merging algorithm is developed based on two observations: 1) the LBP-based detection algorithm performs better in detecting defects that are close to the edges in the unit; and 2) the LBP-based detection algorithm can detect all defective regions but results in many small false positive regions (due to the complex texture of the substrate). These false positive alarms are filtered out by using the intensity-based change detection mask. The edge mask (E) of the unit is generated using an edge detection algorithm. Changes, that are detected using the LBP-based representation and are close to the edges in the edge mask (E), are added to the final mask Z_f.

In order to detect changes that are close to edges, a dilated edge detection mask is generated by applying a morphological dilation operation to the edge mask E. Then, pixel locations with a corresponding value of 1 in Z_{LBP} and which also overlap with the dilated edge mask are added to the final mask Z_f. A size threshold t parameter is subsequently used to decide how to proceed with the other remaining regions where change is detected by using the intensity-based representation and/or the LBP-based representation. Regions smaller than a threshold t are added to the final mask Z_f if a change is marked as detected in this region in both Z_I and Z_{LBP}. Regions of size greater than the threshold t are added to the final change detection mask if a change in such regions is detected by Z_{LBP}. The reason why only the LBP-based detection results is considered in the last case, is that some defects can be only detected by using the LBP-based representation (e.g., scratch on substrate) because of the similarity between the defect intensity value and the background intensity value. Figure 35 illustrates the change detection algorithm. Algorithm 1 describes the steps of the mask fusion process.

5.4.4 Efficient Implementation

The potential downside of our algorithm is that it is computationally intensive to compute the symmetric KLD between the histogram of each patch $(H^T_{i,j})$ in the target image and the corre-

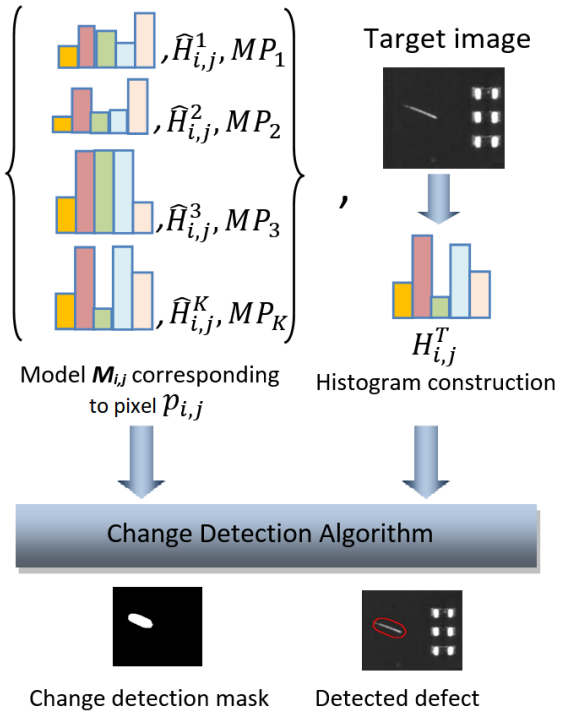

Figure 35: Change Detection. The change detection is a subtraction algorithm, where the symmetric KLD distance is calculated between the target histogram and each centroid histogram in the considered learned model. If the distance is less than a threshold λ, the pixel is considered to be a background pixel with no change, otherwise a change is detected at that pixel location.

sponding model histograms $\widehat{H}^1_{i,j}, \widehat{H}^2_{i,j}, ..., \widehat{H}^K_{i,j}$. To reduce the computational complexity, a greedy subsampling-based algorithm is proposed. In the proposed algorithm, the image id divided into a square grid. Each cell in the grid has a $W_{sub} \times W_{sub}$ pixels. The algorithm begins by only processing one pixel in each cell (e.g., the pixel at the center). In the second iteration, only the pixels that are labeled as changes in the first iteration are considered. For each one of these pixels we process the 8-connected pixels. The algorithm continues by processing the 8-connected pixels of each pixel predicted as change in the previous iteration. Pixels that have already been processed in the previous iterations will not be processed again. The algorithm terminates when there are no new pixels to process.

Because the algorithm ignores the 8-connected pixels of each pixel that is predicted as background in the previous iterations, many of the pixels will not be processed. This reduces the execution time. Although this implementation does not guarantee producing the same result as the original algorithm, in practice, it achieves almost the same performance. The proposed greedy algorithm potentially can miss some pixels that would be labeled as changes by the original algorithm.

Algorithm 1: Mask Fusion Algorithm

$\boldsymbol{Z_I}$: the intensity-based binary change detection mask.
$\boldsymbol{Z_{lbp}}$: the LBP-based binary change detection mask.
$\boldsymbol{r_i}$: is the i[th] region of connected pixels of value of 1 in the Z_{lbp} change detection mask.
$\boldsymbol{r_{i,intensity}}$: is the region, in the intensity mask Z_I, that is co-located with r_i
$\boldsymbol{Z_f}$: is the fused binary change detection mask.
$\boldsymbol{size(r)}$: is the size of a region r in pixels.
$\boldsymbol{r_i \Lambda r_{i,intensity}}$: is the AND operation between two regions, r_i in Z_{lbp} and its co-located region in Z_I. It returns 1 if at least two pixels corresponding to the same location (i, j) are 1.
$\boldsymbol{E}$: is the edge detection mask.
$\boldsymbol{r_i \Lambda E}$: is the AND operation between the co-located pixels in region r_i and the edge detection mask. It returns 1, if at least one AND operation between co-located pixels in r_i and the edge detection mask returns 1.

Procedure MERGE (Z_I, Z_{lbp})
1: Set all the pixel values in Z_f to 0.
2: Let m be the number of regions in the LBP-based change detection mask Z_{LBP}.
3: **for all regions** $\boldsymbol{r_i}$, i: 1 *to* m **do**
4: **if** $\boldsymbol{r_i \Lambda E}$ == *1* **then**
5: *Add* r_i *to* Z_f
6: **else**
7: **if** $size(r_i) <$ t **then**
8: **if** $(r_i \Lambda r_{i,intensity})$ ==1
8: *Add* r_i *to* Z_f
9: **end**
10: **else**
10: *Add* r_i *to* Z_f
Return: The fused mask Z_f

Figure 36: Merging algorithm.

However, these pixels tend to be part of a region smaller than $W_{sub} \times W_{sub}$ pixels. By choosing W_{sub} to be sufficiently small, such small regions are very unlikely to be part of a true defect, and most often are due to noise.

Figure 37 depicts an example of using the iterative change detection algorithm. For this specific example, it is possible to mark the image as defective as early as the second iteration, since the largest connected change detection region is large enough to consider the unit as defective. After the first few iterations, changes in the change detection mask become insignificant. For this specific example, the last iteration was iteration 7. On average, based on experimental results, the total number of iterations is 3 to 4 iterations for defective images and 2 to 3 iterations for non-defective ones.

Let $C_{initial}$ be the computational complexity of processing all the pixels in the considered in-

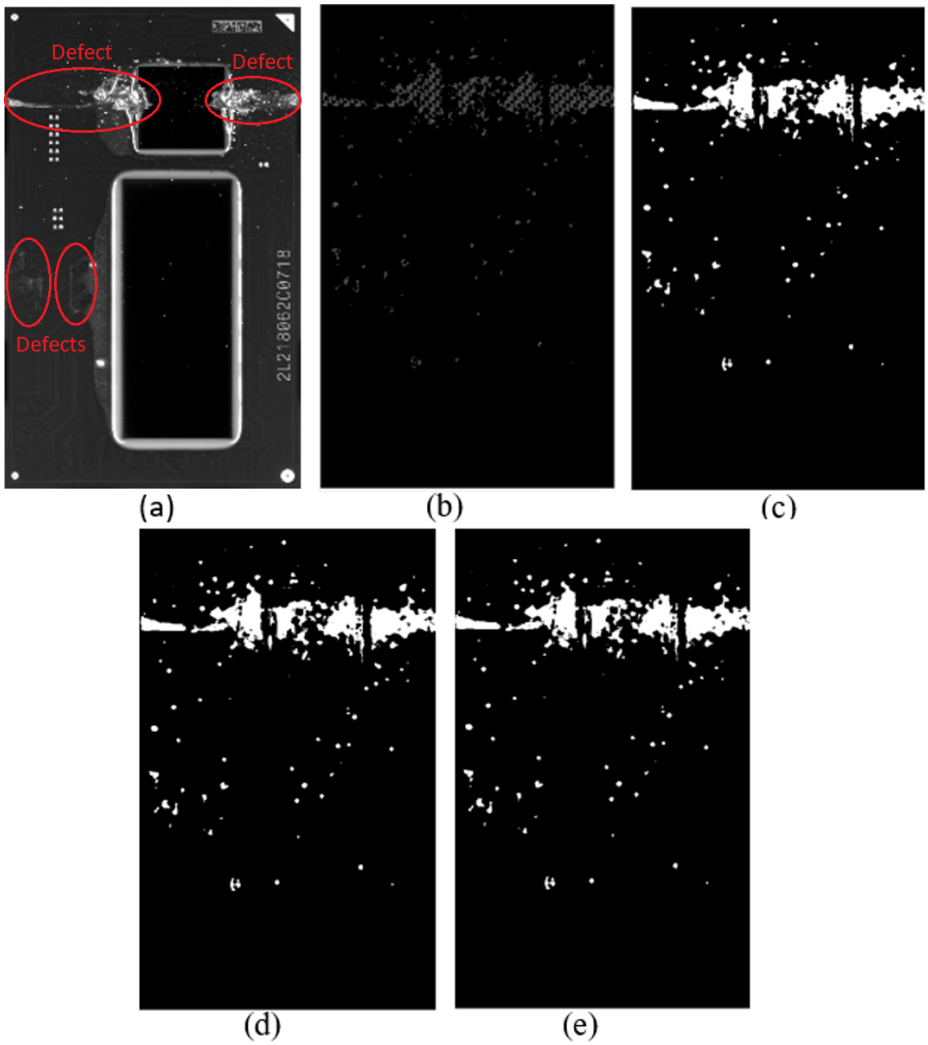

Figure 37: Iterative Change Detection. (a) Defective image; (b) change detection mask after the first iteration; (c) second iteration; (d) third iteration and (e) seventh iteration (the last iteration).

put image. Let C_{greedy} be the computational complexity of the iterative version to finish all iterations. Depending on the size of the cell $W_{sub} \times W_{sub}$, the computational complexity (CT_{greedy}) of processing an image, which can be defined as the number of pixels being processed in the input image multiplied by a constant factor (the time needed to process one pixel), is bounded by $CT_{original}/(W_{sub} \times W_{sub})$ and $CT_{original}$, such that $CT_{original}/(W_{sub} \times W_{sub}) \leq CT_{greedy} \leq CT_{original}$.

5.5 Defect Detection Stage

Regions detected by the change detection process represent regions of change in the unit. Detected regions should contain any defects in the image (true positives), but also may contain changes in the image due to other acceptable variations (e.g., lighting differences and small manufacturing differences) between units (false positives). The goal of the Defect Detection Stage is to classify changes into either defective regions (true positives) or good regions (false positives). Without any a-priori knowledge of the manufacturing process, it is difficult to determine which changes are defects and which changes can be ignored. We propose to learn the differences using a data-driven deep learning framework. In different applications, the tolerance of what constitutes a defect may be different. For instance, in some applications a certain defect may be acceptable, whereas in another application the defect would be unacceptable. The provided training data implicitly defines this tolerance for learning.

We choose to incorporate a deep neural network to learn this information. The motivation behind choosing a deep neural network instead of another classifier is two-fold: 1) the deep network can learn features that may be more appropriate to the problem domain (semiconductor units) compared with hand crafted features designed to work well on many problem domains; 2) the deep neural network can be easily finetuned in the future if new defects need to be detected or can be easily adapted to similar future problems. It is possible to use a pretrained network and adapt it to our problem domain (as in [207]). However, the common pretrained networks are trained on natural image datasets. Our data is very different than natural images. Training a network from scratch can result in a much smaller network than pretrained networks on natural images, which results in computation and memory savings.

The Defect detection stage was designed and developed by Samuel Dodge of the IVU Lab. More details about this stage can be found in [174].

5.6 Refinement Stage

The refinement stage consists of two steps: classifying defective regions into different classes and using defect-type dependent size threshold to filter out the false positives (small ignorable defects). Classifying defects into different classes serves the objective of understanding the types

and distributions of defects. This is very beneficial in applying root cause analysis to understand sources of different defects and taking correction actions. Filtering out defects based on their sizes improves the accuracy of the whole system and reduces the false positive alarms. Different defects impact the functionality of the unit differently. In fact, the size of the defect plays a major role in deciding if a defect is a fatal one or not. This is the main reason why size-based filtering is applied after classifying the defects into different classes. To classify the detected defects, the multi-feature stacking-based classifier (BagStack) proposed in [175] is used. The BagStack classifier has been designed to deal with the data imbalance problem and with small number of instances. The main idea behind the BagStack classifier is to use bagging to deal with the data imbalance problem at the base classifiers level and stacking to combine different training algorithms. Diversity is achieved at the base level by using different features. Six different features are used: SIFT, HOG, SSIM, GB and DNN-based features. Linear support vector machine SVM is used for each of the base classifiers and non-linear support vector machine (RBF) for the meta-classifier. More information about the BagStack classification can be found in [175].

5.7 Evaluation

In this section, the evaluation of the proposed approach is presented. First, a description of the dataset that is used in our experiments is presented. Then, performance results are presented for each stage of the proposed system starting with results for the change detection stage followed by results for the defect detection and refinement stages.

5.7.1 Dataset

A dataset of semiconductor units provided by Intel Corporation is used. The dataset consists of 16 different types of products. In total, we have 261 defective units and 2551 non-defective units. 400 of the non-defective units are used as reference images to learn the background (25 reference images per product). The defects have a nonuniform distribution; there are many more instances of some defect types than others. Grayscale images of the units are taken by using high resolution line scan cameras (16 µm/pixel). For each product, the number of defective units is much smaller than the number of non-defective units.

Table 30: Parameters of detection stage.

Parameter	Description	Value
W_{norm}	Normalization window size	21
W_{hist}	Histogram window size	16
W_{lbp}	LBP window size	11
λ_{lbp}	LBP background threshold	4.0
$\lambda_{intensity}$	Intensity background threshold	1.8
α_{lbp}	LBP mixing percentage threshold	0.3
$\alpha_{intensity}$	Intensity mixing percentage threshold	0.3

We randomly split the 261 defective units and the 2151 non-defective images (after excluding the 400 reference images) into 60% training, 20% validation and 20% testing. In the following experiments, we repeat the random training split 5 times and report average results.

5.7.2 Change Detection

The behavior of the proposed change detection approach is controlled by the set of parameters shown in Table 30. In general, changing these parameters can affect the true positive rate (TPR) and the false positive rate (FPR). Increasing the true positive rate is important to enhance the ability of the system to capture true defects while reducing the false positive rate is important to reduce the non-defective unit's rejection rate.

Values for the parameters listed in table 30 are chosen to give a high sensitivity for the change detection stage, at the expense of a higher number of false alarms. These false alarms will be corrected in the subsequent stages (defect detection stage and refinement stage). In our approach, we set the parameter values with the goal of optimizing the defect detection capability. We chose $\lambda_{lbp} = 1.8$ and $\lambda_{intensity} = 4.0$ to maximize the true positive rate, such that we can detect all the defects while reducing the false positive rate.

Figure 38 depicts the change detection stage results for several defects. The first defect (top row) can mainly be detected by using the LBP-based change detection. The other defects can be detected by fusing both the intensity-based and LBP-based change detection masks. In some images, the intensity-based change detection mask can be used to reduce the false positive alarms that are present in the LBP-based change detection mask.

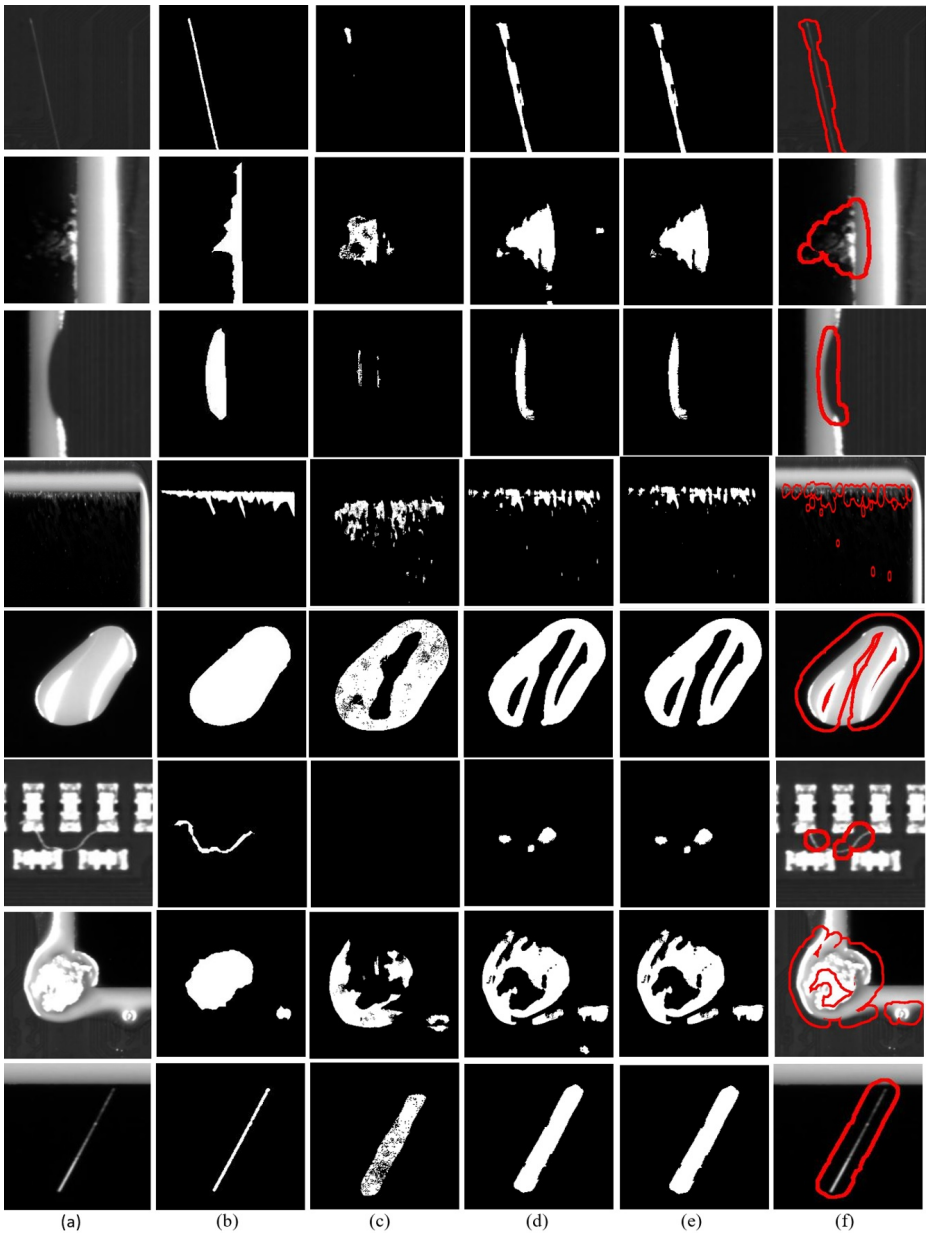

Figure 38: Comparison of change detection results: (a) Original image; (b) Ground truth mask; (c) Intensity-based change detection mask Z_I; (d) LBP-based change detection mask Z_{LBP}; (e) fused mask Z_f; (f) detected changes overlayed on original image.

5.7.3 Defect Detection Stage

True positive rates and false positive rates can be measured at the pixel level, the region level or the unit level. In our application, what really matters is detecting if a unit is defective or not. Since different defects can consist of several regions and can have different expected sizes, looking at

the region level or the pixel level may lead to misleading results. For example, fingerprint on die is usually a large defect (large number of pixels) containing a large number of regions, while foreign material on die is usually a small defect containing only one region.

As described in [174], the deep neural network was implemented using the Theano Python library [208]. Data augmentation is performed during the testing to improve the stability of the results. For each detected region, the original box as well as 4 other boxes are considered. Each of the 4 other boxes is shifted 5 pixels from the original box. The final output is made by averaging the outputs from each box. A unit is considered defective if at least one of the regions is labeled as defective in the unit. A unit labeled as non-defective will have no defective regions. More details about the implementation and performance analysis of the defect detection stage can be found in [174].

5.7.4 Refinement Stage

The refinement stage represents a vital stage in the proposed approach. In this stage, we are interested in two performance measurements: precision and recall. Precision is defined as the number of units that are correctly predicted as non-defective divided by the total number of units predicted as non-defective. Recall is defined as the number of units that are correctly predicted as non-defective divided by the total number of ground-truth non-defective units. Both recall and precision can be calculated by using the confusion matrix.

The used classifier [175] is a multi-feature sparse based classifier. Twenty base learners are used for each feature-problem combination. Six features (SIFT, HOG, GB, SSIM and DNN-Based features) [175] are used. The classifier uses a stacking-based technique to combine the predictions of several base classifiers. Table V shows the number of imaged units predicted as good (i.e., non-defective) and the number of imaged units predicted as defective at the output of the detection stage and the refinement stage (confusion matrices). It is very important to know that only regions detected and marked as defective regions in the second stage will be processed in the refinement stage. For this reason, any real defective region that is missed in the detection stage will not be further detected, while good regions that are flagged by the defect detection stage as defective can be filtered out by using the refinement stage. In practice, it is expensive to miss-classify defective units as good units. Thus, it is better to select the operation point in the sensitivity-specificity curve such that we maximize the true positive rate (recall) while minimizing the false negative rate, even

Table 31: The confusion matrices of both defect detection stage and refinement stage

Ground Truth		Detection Stage		Refinement Stage	
		G	D	G	D
G	430	410.3	19.7	427.6	2.4
D	52	7.6	44.4	7.8	44.2

though that point may increase the number of false positive alarms; the refinement stage can be used to filter out these false positive regions.

Using the confusion matrices in Table 31, we can compute the precision and the recall values. For the defect detection stage, the $precision = (410.3)/(417.9) = 98.18$ and the $recall = (410.3)/(430) = 95.42$. For the refinement stage, the $precision = (427.6)/(435.4) = 98.2$ and the $recall = (427.6)/(430) = 99.44$. The main improvement of using the refinement stage after the defect detection stage is to increase the recall percentage by accepting more good units.

5.8 Discussion and Conclusion

This chapter presents a defect detection and classification framework. To the best of our knowledge, this is the first work that uses machine learning to address the problem of defects detection and classification at the level of semiconductor units, wherein defect detection is performed in an automatic way and all regions of the unit are included. The proposed framework consists of three main stages: change detection, defect detection and refinement stage. The contributions of this thesis include the change detection stage and the refinement stage. The change detection is a locally adaptive change detection algorithm. The defect detection algorithm makes use of a novel deep neural network architecture designed and implemented by Samuel Dodge of the IVU Lab at ASU. The refinement stage is a stack-based classifier. We tested the performance of the proposed approach on images of real semiconductor units. The proposed approach was shown to result in a high detection and classification accuracy. The proposed system was able to achieve a 98.2% precision and a 99.44% recall values.

The proposed method can handle different types of acceptable variations (e.g., light variations, epoxy variations). Our method can ignore these differences while still maintaining the ability to detect the actual defects. Although our change detection system is designed specifically for semi-

conductor units, it can also be applied to other problems (e.g., surface defect detections) that have similar types of nuisance variations (e.g., lighting conditions and material physical properties).

Chapter 6

STEEL SURFACE DEFECT DETECTION AND CLASSIFICATION

This chapter addresses another real application from industry to prove the efficiency and high performance of the proposed BagStack classifier. Surface defect detection in steel products is a hot research topic due to the large number of challenges in this application and the demand to have accurate algorithms that can satisfy the real time requirements. This chapter is organized as follow. In Section 6.1, an introduction to steel surface defect detection problem is presented. In Section 6.2, the main challenges associated with this problem are introduced. Section 6.3 presents the literature review of steel surface defect detection and classification. Section 6.4 presents the classification results using the BagStack classifier and compares its performance to other recent methods in the literature.

6.1 Introduction

Steel is one of the most important metals on the planet. Its importance emerges from the fact that steel is the most common metal on earth and it is used in diverse and a large number of industrial applications. The variation of products in the iron and steel industry necessitates agile and reconfigurable production systems that adapt to various products. This is important to reduce product development time and shorten the product lifecycles. The main two products of steel are the slabs $(1,600 \times 250 \times 12,000 mm^3)$ and billets $(150 \times 150 \times 12,000 mm^3)$. Slabs are subsequently rolled into hot strips and then into cold strips. Billets are rolled into structures of various dimensions. The steel strip is one of the main products of iron and steel industry, and its quality directly affects the quality and performance of the final product [209, 210].

Strength, toughness, ductility, weldability, durability and surface quality are some of the properties that define the quality of the steel. While some of these properties result from the steel's chemical compositions, others result from the manufacturing process. Surface quality represents the most important quality parameter, particularly for flat-rolled (hot or cold) steel products. The presence of defects on the surface of a steel strip has serious implications and limits its use significantly. Surface defects (the most common defects) do not only affect the appearance of the

product, but also affect negatively other favorable properties, e.g., wear resistance, high temperature resistance, corrosion resistance, abrasion resistance and fatigue strength of the steel strip [209, 211].

The quality of the raw materials, the rolling process, the system control, the failures in production machineries and many other factors contribute to different surface defects. Different defects have different adverse effects on steel surface quality. Surface defects consist of many different types of defects, including scratches, surface crazing, rolled-in scale, scars, insect prints, inclusions, bright prints, burrs, seams, black burn, iron scales, pollution, hole, bruise, blisters, edge crack, water droplets (pseudo defect), … etc. Figure 39 depicts samples of six kinds of typical surface defects from the NEU surface defect database [209]. Each row shows one example image from each of 300 samples of a class. In the market, steel products have different tolerance standards for different defects. So, it is not only important to detect the defects but also to identify each one of them [209, 210, 211].

Quality control is a big challenge for steel production; if not performed adequately, quality control can affect greatly the quality of steel and consequently, can cause significant economic loss. Visual surface inspection is playing a critical role in the industrial production of steel strip and such a system can be used to control the product quality. Traditional manual surface inspection procedures are not sufficient and awfully inadequate to ensure high quality and a defect-free surface. Due to high line speed, operators' fatigue, highly subjective nature, being time consuming and other adverse factors, the manual inspection process is hardly satisfactory and cannot meet the requirement of real-time online detection [209, 211, 212].

The importance of surface quality requires that effective and efficient methods be implemented to replace traditional manual visual inspection. To overcome the limitations of human inspection, automated surface inspection systems (ASIS) have become essential to the production system to assist or replace human decisions. In recent years, the visual-based inspection technology, as a kind of non-contact inspection method, has become a research hot-spot in the field of surface defect inspection. This method integrates many advanced technologies including image processing, optics, pattern recognition, artificial intelligence, and has obtained real-time, and relatively high detection accuracy and reliability. A good defect recognition algorithm should be able, in addition to detecting and localizing the defects, to classify them into different categories with high accuracy

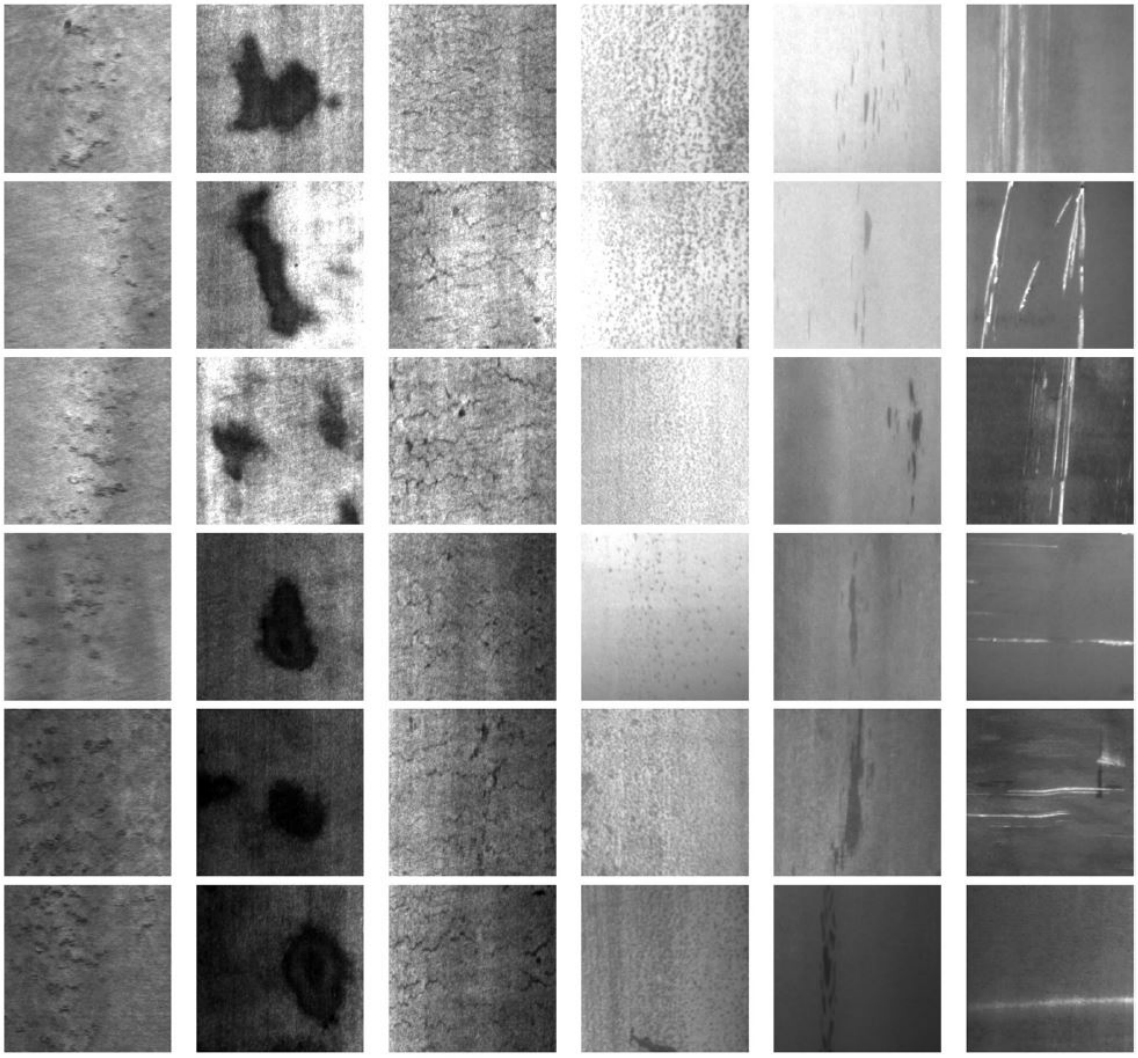

Figure 39: Six kinds of typical surface defects from the NEU surface defect database.From left to right: rolled-in scale, patches, crazing, pitted surface, inclusion, scratches.

and to meet the real time requirements. Defect classification is the task to assign a defect to one class or category.

6.2 Challenges in Steel Surface Defect Recognition

Despite all the advantages (speed, accuracy and upgradability) that come with the visual-based, non-contact surface inspection systems in the steel industry, they face several challenges. The production environment where the inspection equipment (lighting systems, cameras and process-

ing units) resides, is hazardous. The presence of high temperature, dust, oil, water droplet and vapor are very common. Even though the production environment is considered controllable with respect to steel and iron production standards, this kind of environment with all these variations is considered an uncontrolled environment for visual-based inspection systems. Both lighting systems and cameras are vulnerable in the presence of shock and vibration, which introduces additional requirements at the hardware and the algorithm levels to increase the robustness of the system. The operating speed of steel production lines is very high. In general, the speed for flat steel products at the end of the rolling where the inspection happens can reach up to 20 m/s. Operating at such high speeds requires special image processing techniques and equipment that can provide high inspection rates and a short execution time. High variation of surface defects is reported. These defects do not follow specific standards and they also vary also from mill to mill, from production line to another and from product to another. A large number of cameras is required to cover the product, e.g., in flat steel products, two sets of cameras are required to cover the top and the bottom of the product. Each set may contain 3 to 4 cameras to cover the entire width of the strip. Changes in illumination (uneven illumination) can easily affect the stability of the algorithm and the performance of the detection algorithm [209, 210, 211, 212].

Steel surface defects experience high intra-class variations and high inter-class similarities. In fact, images of defects belonging to the same category can exhibit different types of variations; e.g., appearance, texture and color. On the other side, images of defects belonging to different categories may exhibit different types of similarities. Data drift is defined as changes to data over time. In a more technical way, it can be defined as the change that happens to the true data generating distribution over time. Data drift is considered as one of the challenges in online machine learning. Training models on data and testing models on different data that has been generated by using a different generating distribution is problematic and can result in erroneous inference. In the steel industry, the data drift can happen due to different reasons: using a single production line to produce different products, using different production lines to produce the same product; with the passage of time, the physical conditions of the equipment and of the detection device will change. These kinds of changes are negligible on the production environment but can affect the performance of the models that are used in the inspection systems. Training a separate model for each product may not be a feasible option. In fact, collecting defective images for each product may take a long time, and in many cases it is even longer than the product life time. This means that the surface

inspection algorithms should be able to carry knowledge from one product (several products) to new products in an efficient way that can improve the accuracy of the system on the new products [209, 211, 212].

6.3 System Overview

Automated surface inspection systems mainly include two major components: defect detection and defect classification. In the defect detection phase, images of the target product are acquired, these images are scanned and processed to localize and segment potential defects. In many cases, distinguishing between real-defects and pseudo-defects is considered as part of the defect detection process. In the defect classification phase, the detected defects are classified into different categories. While these two stages are considered to be complementary to each other, they still can be evaluated and improved separately [210, 211, 212]. The defect detection phase consists of two main steps: modeling the background to build a template that is used in the second step. In the second step, the input image is compared to the background template to detect potential defects. The defects classification process consists of three main steps: feature extraction, feature aggregation and defect classification. Feature extraction and aggregation is the step of finding a new representation for the image. A representation that can be used as input for the classification algorithm. It is worth mentioning that the ASIS pipeline changed over the time from a simple pipeline, that depends on the traditional image processing techniques like using structural methods (e.g., edge, skeleton, morphological operations), thresholding methods (e.g., Otsu), spectral methods (e.g., Fourier transform, wavelet transform) and model-based methods (e.g., gaussian mixture, low-rank matrix model) to a more advanced pipeline, that depends on machine learning based technology which includes features extraction (e.g., HOG, SIFT, LBP, CLBP, SCXHT…etc.) [209, 212, 213] and pattern classification (e.g., SVM, KNN, DNN, … etc.) [209, 211, 212, 214].

6.4 Related Work

The automated surface inspection started in the 1980s as a result of the high demand for steel products. The vision-based automatic inspection systems became a succedaneum for human-inspection gradually. This section briefly presents the literature review for steel surface defect

detection methods, using traditional machine learning techniques to classify detected defects and using deep learning-based techniques to detect and classify steel surface defects. For more details, Neogi et al. provides a relatively comprehensive review of state-of-the-art steel surface defect detection and classification algorithms using vision-based techniques [209].

To classify defects using traditional methods of machine learning, features are extracted from defective images, these features are then used to find a more compact and rich representation of the image using, for example, bag of visual words or sparse coding techniques. The new representation is then used to train and test a classifier, (e.g., SVM, KNN, NN, DT, … etc).

In [210], a new feature is proposed based on Complete Local Binary Pattern (CLBP). An adjacent evaluation window which is around the neighbor is constructed to diminish the effect of noise interference on the system's performance. The new feature is tested when adding different levels of Gaussian noise using two main classifiers KNN and SVM. The new feature is compared to other features using the same classifiers. The authors of [215] present a multi-feature-based defect classification approach. In this approach, multiple features are extracted, ULBP, GLCM, HOG, Gabor filter-based features and gray level histogram features. Then these features are combined and SVM classifiers are trained with a random subspace of the features. Finally, a Bayes classifier was trained as an evolutionary kernel fused with the results from the sub-SVM to form an integrated classifier. A defect classification framework is presented in [216]. The proposed framework is a multi-feature based one. It uses three types of image features, including geometric feature, grayscale feature and shape feature. These features are extracted and combined. The classification model is based on support vector machine. Cross-validation is used to optimize the classification model. A One-versus-one method is used to solve the multi-class classification problem. A new feature is proposed in [217]. The paper proposed the generalized completed local binary patterns (GCLBP) framework. the new feature is built on top of the Improved completed local binary patterns (ICLBP) and the improved completed noise-invariant local-structure patterns (ICNLP). [218] presents a new approach for defect classification in strip steel using multiple hyper-spheres support vector machine with additional information (MHSVMþ). [219] performs automatic detection of surface defects on rolled steel using computer vision and artificial neural networks. [220] enhances the performance of surface defect classification based on an improved BP algorithm. In [221], the authors described an automated visual inspection system with discrete wavelet transform (DWT) features and a support vector machine (SVM).

In recent years, deep learning (DL) methods have been achieving good performance on image related tasks such as object recognition. Despite the challenges of applying deep learning in the ASI field due to the small number of images, many methods [213] that use deep learning reported competitive results on both defect detection and classification problems [213]. Deep convolutional neural network (CNN) is trained purely on raw defective images. The network consists of two parts, the first few layers (convolution layers) represent the features extraction process and the last few layers (fully connected layers) represent the classification process. These two parts are combined and trained in a single process. In CNN, we not only learn a classifier but also the best feature extraction. This is considered a major advance in machine learning-based computer vision, where we replace using the hand engineered features with features that are learned based on the training data. The lack of data represents a challenge when training CNNs since, due to the large number of parameters, the risk of overfitting becomes higher. Data augmentation and transfer learning are used to alleviate the overfitting problem.

A generic DL-based ASI method is presented in [211]. This method includes feature transferring from a pre-trained DL network and convolution of patch classifier over input image. The input image is divided into cells and each cell is processed to generate a probability that a defect exist. Then, these predictions are combined to generate a heat map. Finally, the heat map is thresholded to detect and classify defects. [212] is another proposed method that makes use of deep learning. Three different convolution neural networks are trained first. Then, the three models are combined by using averaging. Thus, this method represents an ensemble-based approach that uses DCNN models instead of weak learners. In [213], an end-to-end framework is proposed for defect detection and classification. For detection, the symmetric surround saliency map is used. For classification, deep convolutional neural networks (CNNs) which directly use the defect image as input and provide the defect category as output. In [214], a new CNN-based architecture is presented to perform both defect detection and recognition for a metallic surface against complex industrial scenarios. The method consists mainly of two separate steps, defect segmentation and classification. The CASAE module [214] can transform a defect image to a pixel-wise prediction mask that contains only defective pixels and background pixels. To quickly obtain the defect category in real inspection environments, a compact CNN is also presented. [222] is another approach that uses deep learning to perform steel surface defect classification. In this paper, a convolutional neural network is proposed to learn multiple useful feature representations for classification from low level (raw

pixels) to high level (object). Convolutional kernels are initialized by the learned filter kernels that come from sparse auto-encoders. Recently, the You Only Look Once (YOLO) framework has been presented for the purpose of object detection [223]. In [223], a new framework was proposed based on YOLO to detect defects on the steel surface. In [224], another approach that uses deep learning is proposed. A deep convolution neural network is trained to classify steel surface defects. However, the results of the CNN with two different structures still could not reach the requirement of high precision, and no benchmark defect database is used to evaluate its performance. [225] proposed a flexible multi-layered deep feature extraction framework based on CNN via transfer learning to detect anomalies in anomaly datasets. A majority voting mechanism is also designed to overcome the problems of overfitting by combining deep features with linear support vector machine (SVM) classifiers. The deep network structures designed by the above two methods are primarily aimed at the classification task of an imaged defect; and the position of the defect is not localized.

6.5 Results and Analysis

In this section, we present the results of the BagStack classifier when applied to the steel surface defect classification problem. The NEU steel surface defect database [210] is used. The NEU dataset contains six different defects. Figure 39 depicts the different defects in the NEU dataset. Each class contains 300 (200 x 200) images. In all these experiments, we use $50\%\ (150)$ images for training and $50\%\ (150)$ for testing.

Table 32 shows the results of testing different classifiers, using different features. The table shows the overall accuracy and the standard deviation when available and as mentioned in the reference papers. The DL-Based flag indicates whether or not the method uses a deep learning based classifier, mainly a convolution neural network. The Ensemble flag indicates whether or not the method is an ensemble based method. In some cases, the method is both DL-based and Ensemble-based, which means that multiple DL-based methods are used and the predictions of these methods are combined together using bagging for example. If the method is an ensemble method, the column shows the main models used in this ensemble.

In this experiment, we tested different configurations of the BagStack classifier, basically changing the meta-classifier. We use 5 different features, SIFT, and 4 DL-based features (REF(FC6), REF(FC7) [149], VGG16(FC6) and VGG16(FC7) [150]). The five features are used together with

Table 32: Comparison of different methods in terms of overall classification accuracy using the NEU dataset.

Classifier	Feature(s)	(Average) OA	STD	DL Based	Ensemble	Ensemble Models	REF
NNC	LBP	95.07	0.71	No	No		[210]
SVM	LBP	97.93	0.66	No	No		[210]
NNC	LTP	95.93	0.39	No	No		[210]
SVM	LTP	98.22	0.52	No	No		[210]
NNC	CLBP	96.91	0.24	No	No		[210]
SVM	CLBP	98.28	0.51	No	No		[210]
NNC	AECLBP	97.93	0.21	No	No		[210]
SVM	AECLBP	98.93	0.63	No	No		[210]
SVM	GLCM	90.01		No	No		[211]
GBC	GLCM	94.2		No	No		[211]
MLR	MLBP	97.3		No	No		[211]
NNC	SCN	97.3		Yes	No		[211]
SVM	MLBP	97.8		No	No		[211]
GBC	MLBP	97.8		No	No		[211]
SVM	SCN	98.5		Yes	No		[211]
MLR	GLCM	98.61		No	No		[211]
MLR	DeCAF	**99.27**		Yes	No		[211]
Resnet-32		**99.07**	0.27	Yes	No		[212]
WRN-10		**99.69**	0.21	Yes	No		[212]
WRN-20		**99.72**	0.166	Yes	No		[212]
Bagging		**99.89**	0	Yes	Yes	Resnet-32, WRN-10 WRN-20	[212]
CNN		**99.05**		Yes	No		[213]
BYEC-Stacking	ULBP, GLCM, HOG,Gabor, Gary level Hist	96.3		No	Yes	SVM	[215]
BagStack-RF		**99.822**	0.001	No	Stacking	DT, SVM (RBF)	BagStack Classifier
BagStack-EBMDTR0	SIFT	**99.855**	0.005	No	Stacking	LR (LSE, Lasso)	
BagStack-EBMDTR1	Ref(FC6)	**99.912**	0.007	No	Stacking	LR (SVM, Lasso)	
BagStack-SVML	Ref(FC7)	**99.955**	0.001	No	Stacking	LR (LSE, ridge)	
BagStack-SVM	VGG16(FC6)	92.33	0.214	No	Stacking	LR (SVM, ridge)	
BagStack-Bagging	VGG16(FC7)	**99.56**	0.0036	No	Stacking	Bagging	
BagStack-Boosting (RUS)		96.125	0.179	No	Stacking	Boosting, RF	

each configuration. For each configuration several features are used to train base learners. Nine different base learners are used, decision tree, linear SVM, linear regressions ((LSE, Lasso), (LSE, Ridge), (SVM, Lasso) and (SVM, Ridge)), Bagging (DT), Boosting (DT) and Random Forest. Seven BagStack configurations are implemented and tested. The main difference between these configurations is the used meta-data classifier. Seven different meta-classifiers are used including, Random Forest RF, Ensemble-based multiple decision tree regression (EBMDTR0/1), Linear Support Vector Machine, Bagging of decision tree, and random undersampling boosting using decision trees. The main difference between EBMDTR0 and 1 is that the first one uses all the outputs from the base learners and the latter uses only a subset of the outputs when training each regression function. In 5 out of the 7 configurations, the BagStack classifier is able to achieve a classification accuracy higher than 99.0%. The reported average overall accuracy is the average of 10 times.

Chapter 7

CONCLUSION

Ensemble classifiers have attracted significant attention during the last two decades. An ensemble classifier combines multiple base learners (weak) to build a stronger one. The three most known ensemble classifiers are Bagging, Boosting and Stacking. In this work, a new classifier (the BagStack classifier) is proposed. The new classifier uses the concepts of bagging and stacking to deal with the data imbalance problem under the assumptions of having few number of samples, within class variations and similarities between classes. The classifier can combine multiple training algorithms and data representations that have non-zero generalization error (bias and variance) to achieve better accuracy. This work also addresses the problem of defect detection and classification in semiconductor units. A locally adaptive statistical background modeling approach for defect detection is proposed. To prove the effectiveness of the proposed BagStack classifier, defect classification on steel surfaces is also considered. This chapter summarizes the main contributions of this work and suggests possible directions for future research.

7.1 Contributions

The main contributions of this work are as follows:

- BagStack classifier is proposed to handle the data imbalance problem when the number of available samples from one class is relatively small as compared to the other class(es).
- This work adopted the method of solving the multi-class classification problem by first converting it to a set of binary classification problems and then training several base learners to solve each one of these binary problems (bagging). A novel method was proposed to specify the number of base learners and the number of samples to sample from the majority and the minority classes adaptively. The main objective is to maximize the usage of the majority class under the constraint of using a specific number of base learners for all problems.
- Variation-based, Adaptive, Synthetic Minority Oversampling Technique (VA-SMOTE) is proposed to oversample the minority class and undersample the majority class. VA-SMOTE

considers the variation within the minority class and the number of samples belonging to the minority class to control the behaviour of the synthetic process.

- This work proposes the imbalance-aware cross-validation to generate balanced meta-data. Using the standard V-fold cross validation to generate the meta-data and train the BagStack classifier results in imbalanced meta-data and a biased meta-classifier. The proposed approach adaptively increases the number of folds (K) to handle the data imbalance problem at the meta-level.
- A multi-stage defect detection and classification framework is proposed to detect defects in semiconductor units. The proposed framework consists of three stages: proposal generation, defect detection and refinement stage. A locally adaptive statistical background modeling is proposed for proposal generation and change detection.

7.2 Future Research Directions

There are several directions that can be explored in future work:

- The proposed defect detection and classification framework can be applied to other problem domains wherein data imbalance is a problem and multiple features are needed to achieve high accuracy. In this work, defect detection and classification are considered in semiconductor units and steel surfaces.
- The proposed framework can be combined with other recent advances in image synthesis methods (e.g., Generative Adversarial Network) to synthesize images that can be used for training.
- Pruning techniques can be applied at the base-learners level to eliminate redundant classifiers which serves both complexity and performance. Reducing the number of base learners can reduce the tendency of the meta-classifier to overfit the meta-data.
- The BagStack classifier is a general classifier that can be applied to different classification problem. Providing the mathematical insights to identify problems wherein BagStack classifier can overcome other well-known ensemble classifiers is beneficial.

REFERENCES

[1] Sebastiano Battiato Roberto Cipolla and Giovanni Maria Farinella. *Machine Learning for Computer Vision*. Springer, 2013.

[2] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[3] Xuezhi Yang, G Pang, and N Yung. Robust fabric defect detection and classification using multiple adaptive wavelets. *IEE Proc. Vision Image Processing*, 152(6):15–723, 2005.

[4] Robert E Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[5] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[6] Bee Wah Yap, Khatijahhusna Abd Rani, Hezlin Aryani Abd Rahman, Simon Fong, Zuraida Khairudin, and Nik Nik Abdullah. An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. In *Proceedings of the First International Conference on Advanced Data and Information Engineering*, pages 13–22, 2014.

[7] V. Ganganwar. An overview of classification algorithms for imbalanced datasets. *International Journal Emerging Technology and Advanced Engineering*, 2(4):42–47, 2012.

[8] Wei Fan, Salvatore J. Stolfo, Junxin Zhang, and Philip K. Chan. Adacost: Misclassification cost-sensitive boosting. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 97–105, 1999.

[9] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 2005.

[10] Szu-Hao Huang and Ying-Cheng Pan. Automated visual inspection in the semiconductor industry: a survey. *Computers in Industry*, 6:1–10, 2015.

[11] Carlos Mera, Mauricio Orozco-Alzate, John Branch, and Domingo Mery. Automatic visual inspection: An approach with multi-instance learning. *Computers in Industry*, 83:46–54, 2016.

[12] Wei Chen, Li-Ming Wu, and Shan-Ling Cui. A high speed image pre-processing method for ic wafer inspection. In *International Conference on Electronic Packaging Technology and High Density Packaging*, pages 103–106, 2008.

[13] Louis Breaux and Baljit Singh. Automatic defect classification system for patterned semiconductor wafers. In *Proceedings of International Symposium on Semiconductor Manufacturing*, pages 68–73, 1995.

[14] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16(1):321–357, 2002.

[15] Giorgio Fumera, Fabio Roli, and Alessandra Serrau. Dynamics of variance reduction in bagging and other techniques based on randomization. *Multiple Classifier Systems*, 3541, 2005.

[16] Leo Breiman. Bias, variance, and arcing classifiers. Technical report, University of California at Berkeley, 1996.

[17] Peter Bühlmann, Bin Yu, et al. Analyzing bagging. *The Annals of Statistics*, 30(4):927–961, 2002.

[18] Andreas Buja and Werner Stuetzle. Observations on bagging. *Statistica Sinica*, 16(2):323–351, 2006.

[19] Jerome H Friedman and Peter Hall. On bagging and nonlinear estimation. *Journal of Statistical Planning and Inference*, 137(3):669–683, 2007.

[20] Peter Bühlmann. Bagging, boosting and ensemble methods. *Handbook of Computational Statistics*, pages 877–907, 2004.

[21] Sotiris B Kotsiantis. Bagging and boosting variants for handling classifications problems: a survey. *The Knowledge Engineering Review*, 1:1–23, 2013.

[22] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall-CRC, 2012.

[23] Artur FerreiraEmail and Mario Figueiredo. Boosting algorithms: A review of methods theory and applications. *Ensemble Machine Learning*, pages 35–85, 2012.

[24] Paula Branco, Luis Torgo, and Rita Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys*, 49(2):31:1–31:50, 2016.

[25] Nazia Tabassum and Tanvir Ahmed. A theoretical study on classifier ensemble methods and its applications. In *Proceedings of the 3rd International Conference on Computing for Sustainable Global Development*, pages 67–78, 2016.

[26] Krystyna Napierala and Jerzy Stefanowski. Types of minority class examples and their influence on learning classifiers from imbalanced data. *Intelligent Information Systems*, 46(3):563–597, 2016.

[27] Taghi Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Comparing boosting and bagging techniques with noisy and imbalanced data. *IEEE Transactions on Systems Man and Cybernetics: Part A: Systems and Humans*, 41(3):552–568, 2011.

[28] Robert E Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[29] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Computational Learning Theory:*, 55(1):119–139, 1997.

[30] Ludmila Kuncheva, Marina Skurichina, and Robert P. W. Duin. An experimental study on diversity for bagging and boosting with linear classifiers. *Information Fusion*, 3(4):245–258, 2002.

[31] David W. Opitz and Richard F. Maclin. An empirical evaluation of bagging and boosting for artificial neural networks. In *Proceedings of International Conference on Neural Networks*, June 1997.

[32] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods, 1997.

[33] Pedro Domingos. A unified bias–variance decomposition for zero–one and squared loss. In *In Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 564–569, 2000.

[34] Thomas G. Dietterich. Ensemble methods in machine learning. *Multiple Classifier Systems*, 1857:1–15, 2001.

[35] Rocco A. Servedio. Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, 4:633–648, 2003.

[36] Alexander Vezhnevets and Olga Barinova. Avoiding boosting overfitting by removing confusing samples. In *Machine Learning*, pages 430–441, 2007.

[37] Yunlong Gao, Feng Gao, and Xiaohong Guan. Improved boosting algorithm with adaptive filtration. In *8th World Congress on Intelligent Control and Automation*, pages 3173–3178, 2010.

[38] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[39] Richard Nocka and Frank Nielsen. A real generalization of discrete adaboost. *Artificial Intelligence*, 171(1):25–41, 2007.

[40] Jerome H. Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

[41] Jerome H. Friedman. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[42] Alexander Vezhnevets and Vladimir Vezhnevets. Modest adaboost – teaching adaboost to generalize better. In *Computer Graphics and Applications*, 2005.

[43] Stan Li and ZhenQiu Zhang. Float boosting learning and statistical face detection. *IEEE Transactions on Pattern Analysis and Machine Learning*, 26(9):1112–1123, 2004.

[44] Peter Buhlmann and Torsten Hothorn. Boosting algorithms: regularization prediction and model fitting. *Statistical Science*, 22(4):477–505, 2007.

[45] Juan J. Rodriguez and Jesus Maudes. Boosting recombined weak classifiers. *Pattern Recognition Letters*, 29(8):1049–1059, 2007.

[46] Andreas Toscher, Michael Jahrer, and Robert M. Bell. The bigchaos solution to the netflix grand prize, 2009.

[47] Joseph Sill, Gabor Takacs, Lester Mackey, and David Lin. Feature-weighted linear stacking. *ArXiv e-prints*, 2009.

[48] Yehuda Koren. The bellkor solution to the netflix grand prize, 2009.

[49] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–260, 1992.

[50] Mark J. Van der Laan, Eric C. Polley, and Alan E. Hubbard. Super learner. *Statistical Applications in Genetics and Molecular Biology*, 6(1):25, 2007.

[51] P. Shunmugapriya and S. Kanmani. Optimization of stacking ensemble configurations through artificial bee colony algorithm. *Swarm and Evolutionary Computation*, 12:24 – 32, 2013.

[52] M. Paz Sesmero, Agapito I. Ledezma, and Araceli Sanchis. Generating ensembles of heterogeneous classifiers using stacked generalization. *WIREs Data Mining Knowledge Discovery*, 5(1):21–34, 2015.

[53] Leo Breiman. Stacked regressions. *Machine Learning*, 24(1):49–64, 1996.

[54] David Bingham Skalak. Prototype selection for composite nearest neighbor classifiers. Technical report, University of Massachusetts Amherst, 1997.

[55] Wei Fan, Salvatore J. Stolfo, and Philip K. Chan. Using conflicts among multiple base classifiers to measure the performance of stacking. In *Proceedings of the ICML Workshop on Recent Advances in Meta-Learning and Future Work*, pages 10–15, 1999.

[56] Kai Ming Ting and Ian H. Witten. Stacked generalization: when does it work? In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 866–871, 1997.

[57] Kai Ming Ting and Ian H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[58] Alexander K. Seewald. How to make stacking better and faster while also taking care of an unknown weakness. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 554–561, 2002.

[59] Kai Ming Ting and Ian H. Witten. Stacking bagged and dagged models. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 367–375, 1997.

[60] Christopher J. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 36(1):33–58, 1999.

[61] Saso Dzeroski and Bernard Zenko. Is combining classifiers with stacking better than selecting the best one? *Machine Learning*, 54(3):255–273, 2004.

[62] Saso Dzeroski and Bernard Zenko. Stacking with multi-response model trees. In *Multiple Classifier Systems*, pages 201–211, 2002.

[63] Bernard Zenko and Saso Dzeroski. Stacking with an extended set of meta-level attributes and mlr. In *Machine Learning*, pages 493–504, 2002.

[64] Bernard Zenko, Ljupco Todorovski, and Saso Dzeroski. A comparison of stacking with meta decision trees to bagging, boosting, and stacking with other methods. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, 2001.

[65] Alexander K. Seewald and Johannes Fuernkranz. An evaluation of grading classifiers. In *Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis*, pages 115–124, 2001.

[66] Anna Jurek, Yaxin Bi, Shengli Wu, and Chris Nugent. A survey of commonly used ensemble-based classification techniques. *Knowledge Engineering Review*, 29(5):551–581, 2014.

[67] Sam Reid and Greg Grudic. Regularized linear models in stacked generalization. In *Multiple Classifier Systems*, pages 112–121, 2009.

[68] Xinlong Bao, Lawrence Bergman, and Rich Thompson. Stacking recommendation engines with additional meta-features. In *Proceedings of the 3rd ACM Conference on Recommender Systems*, pages 109–116, 2009.

[69] Buzhou Tang, Qingcai Chen, Xuan Wang, and Xiaolong Wang. Reranking for stacking ensemble learning. In *Proceedings of the 17th International Conference on Neural Information Processing*, pages 575–584, 2010.

[70] Eitan Menahem, Lior Rokach, and Yuval Elovici. Troika–an improved stacking schema for classification tasks. *Information Science*, 179(24):4097–4122, 2009.

[71] Agapito Ledezma, Ricardo Aler, and Daniel Borrajo. *Heuristics and optimization for knowledge discovery*, chapter Heuristic search based stacking of classifiers, pages 54–67. IRMA-International, 2002.

[72] Fco. Javier Ordonez, Agapito Ledezma, and Araceli Sanchis. Genetic approach for optimizing ensembles of classifiers. In *Proceedings of the* 21^{st} *International Florida Artificial Intelligence Research Society Conference*, pages 89–94, 2008.

[73] Agapito Ledezma, Ricardo Aler, Araceli Sanchis, and Daniel Borrajo. Ga-stacking: evolutionary stacked generalization. *Intelligent Data Analysis*, 14(1):89–119, 2010.

[74] Riyaz Sikora and O'la Al-Laymoun. A modified stacking ensemble machine learning algorithm using genetic algorithms. *Journal of International Technology and Information Management*, 23(1):1–11, 2014.

[75] Zhi-quan QIN, Chang-jian WANG, Yu-xing PENG, and Yuan. An advanced genetic approach for stacking classifiers. In *International Conference on Artificial Intelligence: Techniques and Applications*, pages 276–284, 2016.

[76] Zhiqiang Zheng and Balaji Padmanabhan. Constructing ensembles from data envelopment analysis. *Journal on Computing*, 19(4):486–496, 2007.

[77] Dan Zhu. A hybrid approach for efficient ensembles. *Decision Support Systems*, 48(3):480–487, 2010.

[78] YiJun Chen, Man-Leung Wong, and Haibing Li. Applying ant colony optimization to configuring stacking ensembles for data mining. *Expert Systems with Applications*, 41(6):2688–2702, 2014.

[79] Rafael S. Parpinelli, Heitor S. Lopes, and Alex A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332, 2002.

[80] Samaira Hodnefjell and Ilaim Costa Junior. Classification rule discovery with ant colony optimization algorithm. *Intelligent Data Engineering and Automated Learning*, 7435:678–687, 2012.

[81] Ziqiang Wang and Boqin Feng. Classification rule mining with an improved ant colony algorithm. *Lecture notes in computer science*, 3339:357–367, 2004.

[82] Abdul Rauf Baig, Waseem Shahzad, and Salabat Khan. Correlation as a heuristic for accurate and comprehensible ant colony optimization based classifiers. *IEEE Transactions on Evolution Computing*, 17(5):686–704, 2013.

[83] Zhi-Hui Zhan, Jun Zhang, Yun Li, and Henry Shu-Hung Chung. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(6):1362–1381, 2009.

[84] H. Ghashochi Bargh and M. H. Sadr. Stacking sequence optimization of composite plates for maximum fundamental frequency using particle swarm optimization algorithm. *Meccanica*, 47(3):719–730, 2012.

[85] P. Shunmugapriya. Optimization of stacking ensemble configurations through artificial bee colony algorithm. *Swarm and Evolutionary Computation*, 12:24–32, 2013.

[86] Isvani Frias-Blanco, Alberto Verdecia-Cabrera, Agustin Ortiz-Diaz, and Andre Carvalho. Fast adaptive stacking of ensembles. In *Proceedings of the* 31^{st} *Annual ACM Symposium on Applied Computing*, pages 929–934, 2016.

[87] Zhongchen Ma and Qun Dai. Selected an stacking elms for time series prediction. *Neural Process Letter*, 44(3):831–856, 2016.

[88] T. Ryan Hoens and Nitesh V. Chawla. *Imbalanced learning: foundations, algorithms, and applications*, chapter Imbalanced datasets: from sampling to classifiers, pages 43–59. Wiley Online Library, 2013.

[89] Nathalie Japkowicz. *Imbalanced learning: foundations, algorithms, and applications*, chapter Assessment metrics for imbalanced learning, pages 1–2. Wiley Online Library, 2013.

[90] XuYing Liu and ZhiHua Zhou. *Imbalanced learning: foundations, algorithms, and applications*, chapter Ensemble methods for class imbalance learning, pages 1–2. Wiley Online Library, 2013.

[91] Piotr Juszczak and Robert P. W. Duin. Selective sampling methods in one-class classification problems. In *Artificial Neural Networks and Neural Information Processing*, pages 140–148, 2003.

[92] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging, boosting, and hybrid-based approaches. *IEEE Transactions on System, Man, Cybernetics*, 42(4):463–484, 2012.

[93] Charles Elkan. The foundations of cost-sensitive learning. In *Proceedings of the* 17^{th} *International Joint Conference on Artificial Intelligence*, pages 973–978, 2001.

[94] Kai Ming Ting. An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):659–665, 2002.

[95] Xu-ying Liu and Zhi-hua Zhou. The influence of class imbalance on cost-sensitive learning: an empirical study. In *Sixth International Conference on Data Mining*, 2006.

[96] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. In *Advances in Intelligent Computing*, pages 878–887, 2005.

[97] Seyda Ertekin. Adaptive oversampling for imbalanced data classification. In Erol Gelenbe and Ricardo Lent, editors, *Information Sciences and Systems*, pages 261–269, 2013.

[98] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. Safe-level-smote: safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In *Advances in Knowledge Discovery and Data Mining*, pages 475–482, 2009.

[99] Shengguo Hu, Yanfeng Liang, Lintao Ma, and Ying He. Msmote: improving classification performance when training data is imbalanced. In *Proceeding of the* 2^{nd} *International Workshop Computing, Science and Engineering*, volume 2, pages 13–17, 2009.

[100] Wacharasak Siriseriwan and Krung Sinapiromsaran. Adaptive neighbor synthetic minority oversampling technique under 1nn outcast handling. *Journal of Science and Technology*, 39(5):565–576, 2017.

[101] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. Dbsmote: density-based synthetic minority over-sampling technique. *Applied Intelligence*, 36(3):664–684, 2012.

[102] Elhassan T, M Aljurf, F Al-Mohanna, and M Shoukri. Classification of imbalance data using tomek link (t-link) combined with random under-sampling (rus) as a data reduction method. *Journal of Informatics and Data Mining*, 1:1–12, 2016.

[103] Shuo Wang and Xin Yao. Diversity analysis on imbalanced data sets by using ensemble models. In *IEEE Symposium on Computational Intelligence and Data Mining*, pages 324–331, 2009.

[104] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on System, Man and Cybernetics*, 42(4):463–484, 2012.

[105] Qi Wang, ZhiHao Luo, JinCai Huang, YangHe Feng, and Zhong Liu. A novel ensemble method for imbalanced data learning: bagging of extrapolation-smote svm. *Computational Intelligence and Neuroscience*, 2017:1–11, 2017.

[106] Shohei Hido, Hisashi Kashima, and Yutaka Takahashi. Roughly balanced bagging for imbalanced data. *Statistical Analysis and Data Mining*, 2(5-6):412–426, 2009.

[107] Nitesh V. Chawla, Aleksandar Lazarevic, Lawrence O. Hall, and Kevin W. Bowyer. Smoteboost: Improving prediction of the minority class in boosting. In *Knowledge Discovery in Databases*, pages 107–119, 2003.

[108] Hongyu Guo and Herna L. Viktor. Boosting with data generation: Improving the classification of hard to learn examples. In *Innovations in Applied Artificial Intelligence*, pages 1082–1091, 2004.

[109] Chris Seiffert, Taghi M. Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Rusboost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on System, Man and Cybernetics*, 40(1):185–197, 2010.

[110] Chris Seiffert, Taghi M. Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Rusboost: Improving classification performance when training data is skewed. In *19th International Conference on Pattern Recognition*, pages 1–4, 2008.

[111] M.V. Joshi, V. Kumar, and R.C. Agarwal. Evaluating boosting algorithms to classify rare classes: comparison and improvements. In *Proceedings of the IEEE International Conference on Data Mining*, pages 257–264, 2001.

[112] Yanmin Sun, Mohamed S. Kamel, Andrew K.C. Wong, and Yang Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40(12):3358–3378, 2007.

[113] Paul Viola and Michael Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, pages 1311–1318, 2001.

[114] Qing-Yan Yin, Jiang-She Zhang, Chun-Xia Zhang, and Sheng-Cai Liu. An empirical study on the performance of cost-sensitive boosting algorithms with different levels of class imbalance. *Mathematical Problems in Engineering*, 2013:1–12, 2013.

[115] Hongyu Guo and Herna L. Viktor. Learning from imbalanced data sets with boosting and data generation: The data-boost im approach. *ACM SIGKDD Explorations Newsletter*, 6(1):30–39, 2004.

[116] Bartosz Krawczyka, Mikel Galarb, Lukasz Jelenc, and Francisco Herrerade. Evolutionary undersampling boosting for imbalanced classification of breast cancer malignancy. *Applied Soft Computing*, 38:714–726, 2016.

[117] Tian-Yu Liu. Easy-ensemble and feature selection for imbalance data sets. In *Proceedings of the 2009 International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing*, pages 517–520, 2009.

[118] Geoffrey I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, 2000.

[119] Bartosz Krawczyk and Michal Wozniak. Wagging for combining weighted one-class support vector machines. *Procedia Computer Science*, 51:1565–1573, 2015.

[120] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4):367–378, 2002.

[121] Yang Yu, Zhi-Hua Zhou, and Kai Ming Ting. Cocktail ensemble for regression. In *The Seventh IEEE International Conference on Data Mining*, pages 721–726, 2007.

[122] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(2):539–550, 2009.

[123] Mike Galar, Alberto Fernandez, Edurne Barrenechea, and Fransisco Herra. Eusboost: enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. *Pattern Recognition*, 46(12):3460–3471, 2013.

[124] Jose F. Diez-Pastor, Juan J. Rodriguez, Cesar Garcia-Osorio, and Ludmila I. Kuncheva. Random balance: ensembles of variable priors classifiers for imbalanced data. *Knowledge-Based Systems*, 85:96–111, 2015.

[125] Kewen Li, Xianghua Fang, Jiannan Zhai, and Qinghua Lu. An imbalanced data classification method driven by boundary samples-boundary-boost. In *The 3rd International Conference on Information Science and Control Engineering*, pages 194–199, 2016.

[126] Guo Haixiang, Li Yijing, Li Yanan, Liu Xiao, and Li Jinling. Bpso-adaboost-knn ensemble learning algorithm for multi-class imbalanced data classification. *Engineering Applications of Artificial Intelligence*, 49:176–193, 2016.

[127] Wei Lu, Zhe Li, and Jinghui Chu. Adaptive ensemble undersampling-boost: A novel learning framework for imbalanced data. *Journal of systems and software*, 132:272–282, 2017.

[128] Mark J. Van Der Laan and Sandrine Dudoit. Unified cross-validation methodology for selection among estimators and a general cross-validated adaptive epsilon-net estimator: finite sample oracle inequalities and examples. Technical report, Division of Biostatistics, University of California, Berkeley, 2003.

[129] Erick C. Polley and Mark J. Van der Laan. Super learner in prediction. Technical report, Division of Biostatistics, University of California, Berkeley, 2010.

[130] A. Van der Vaart, S. Dudoit, and M. Van der Laan. Oracle inequalities for multi-fold cross validation. *Statistics and Risk Modeling*, 24(3), 2006.

[131] J. Alcala-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. Garcia, L. Sanchez, and F. Herrera. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011.

[132] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[133] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[134] Yun Qian, Yanchun Liang, Mu Li, Guoxiang Feng, and Xiaohu Shi. A resampling ensemble algorithm for classification of imbalance problem. *Neurocomputing*, 143:57–67, 2014.

[135] Rong Yan, Jelena Tesic, and John R. Smith. Model-shared subspace boosting for multi-label classification. In *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining*, pages 834–843, 2007.

[136] Jun Liao. *Totally corrective boosting algorithms that maximize the margin*. PhD thesis, Computer Science, 2006.

[137] Ayhan Demiriz, Kristin P. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1/2/3):255–254, 2002.

[138] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[139] David G. Lowe. Distinctive image features from scale-invariant key-points. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[140] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and LucVan Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[141] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

[142] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *NIPS*, pages 831–837, 2000.

[143] Eli Shechtman and Michal Irani. Matching local self-similarities across images and videos. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[144] Manu Viswanathan, Christian Siagian, and Laurent Itti. Comparisons of gist models in rapid scene categorization tasks, 2008.

[145] T. Ahonen, A. Hadid, and M. Pietikainen. Face description with local binary patterns: application to face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):2037–2041, 2006.

[146] A.C. Berg, T.L. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondences. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 26–33, 2005.

[147] Mouna Dammak, Mahmoud Mejdoub, and Chokri Ben Amar. A survey of extended methods to the bag of visual words for image categorization and retrieval. In *International Conference on Computer Vision Theory and Applications*, pages 676–683, 2014.

[148] Zheng Zhang, Yong Xu, Jian Yang, Xuelong Li, and David Zhang. A survey of sparse representation: Algorithms and applications. *IEEE Access*, 3:490–530, 2015.

[149] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[150] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[151] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[152] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[153] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[154] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[155] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[156] GE Hinton, S Osindero, and YW Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[157] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. A maximum entropy framework for part-based texture and object recognition. In *Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 832–838, 2005.

[158] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Semi-local affine parts for object recognition. In *BMVC*, pages 959–968, 2004.

[159] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: spatial pyramid matching for recognizing natural scene categories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2006.

[160] Kechen Song and Yunhui Yan. A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects. *Applied Surface Science*, 285:858–864, 2013.

[161] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 215–223, 2011.

[162] D. T. Pham and R. J. Alcock. *Smart inspection systems: Techniques and applications of intelligent vision*, chapter Automated visual inspection and artificial intelligence, pages 1–34. Science Direct, 1st edition, 2003.

[163] Louis Bream and Baljit Singh. Automatic defect classification system for patterned semiconductor wafers. In *Proceedings of International Symposium on Semiconductor Manufacturing*, pages 68–73, 1995.

[164] Shih-Jong J Lee and Chih-Chau L Kuan. Method and apparatus for incremental concurrent learning in automatic semiconductor wafer and liquid crystal display defect classification, 2000. US Patent 6,148,099.

[165] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–15, 2009.

[166] Tae-Hyeon Kim, Tai-Hoon Cho, Young Shik Moon, and Sung Han Park. Visual inspection system for the classification of solder joints. *Pattern Recognition*, 32(4):565–575, 1999.

[167] Giuseppe Acciani, Girolamo Fornarelli, and Antonio Giaquinto. A fuzzy method for global quality index evaluation of solder joints in surface mount technology. *IEEE Transactions on Industrial Informatics*, 7(1):115–124, 2011.

[168] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[169] Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang. Linear spatial pyramid matching using sparse coding for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1794–1801, 2009.

[170] Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3360–3367, 2010.

[171] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. CRC Press, 2015.

[172] Jun Liu, Shuiwang Ji, Jieping Ye, et al. Slep: Sparse learning with efficient projections, 2009.

[173] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning*, pages 111–118, 2010.

[174] Bashar M Haddad, Samuel F Dodge, Lina J Karam, Nital S Patel, and Martin W Braun. Locally adaptive statistical background modeling with deep learning based false positive rejection for defects detection in semiconductor units. *To be submitted to IEEE Transactions on Instrumentation and Measurement*, 2019.

[175] Bashar M Haddad, Sen Yang, Lina J Karam, Jieping Ye, Nital S Patel, and Martin W Braun. Multifeature, sparse-based approach for defects detection and classification in semiconductor units. *IEEE Transactions on Automation Science and Engineering*, 15(1):1–15, 2016.

[176] Andrews Sobral and Antoine Vacavant. A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos. *Computer Vision and Image Understanding*, 122:4–21, 2014.

[177] Sartajvir Singh and Rajneesh Talwar. Review on different change vector analysis algorithms based change detection techniques. In *IEEE Second International Conference on Image Information Processing*, pages 136–141, 2013.

[178] Richard J Radke, Srinivas Andra, Omar Al-Kofahi, and Badrinath Roysam. Image change detection algorithms: a systematic survey. *IEEE transactions on Image Processing*, 14(3):294–307, 2005.

[179] Weilong Ren, Jianshe Song, Song Tian, and Wenfeng Wu. Survey on unsupervised change detection techniques in sar images1. In *IEEE China Summit and International Conference on Signal and Information Processing*, pages 143–147, 2014.

[180] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[181] Marko Heikkila and Matti Pietikainen. A texture-based method for modeling the background and detecting moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):657–662, 2006.

[182] Csaba Kertész and Vincit Oy. Texture-based foreground detection. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 4(4):51–62, 2011.

[183] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.

[184] P. KaewTraKulPong and R. Bowden. *An improved adaptive background mixture model for real-time tracking with shadow detection*, pages 135–144. Springer US, 2002.

[185] SeungJong Noh and Moongu Jeon. A new framework for background subtraction using multiple cues. In *Asian Conference on Computer Vision*, pages 493–506, 2012.

[186] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002.

[187] Jian Yao and Jean-Marc Odobez. Multi-layer background subtraction based on color and texture. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[188] Jianbo Yu and Xiaolei Lu. Wafer map defect detection and recognition using joint local and nonlocal linear discriminant analysis. *IEEE Transactions on Semiconductor Manufacturing*, 29(1):33–43, 2016.

[189] Fatima Adly, Paul D Yoo, Sami Muhaidat, Yousof Al-Hammadi, Uihyoung Lee, and Mohammed Ismail. Randomized general regression network for identification of defect patterns in semiconductor wafer maps. *IEEE Transactions on Semiconductor Manufacturing*, 28(2):145–152, 2015.

[190] Chen-Fu Chien and Chia-Yu Hsu. Data mining for optimizing ic feature designs to enhance overall wafer effectiveness. *IEEE Transactions on Semiconductor Manufacturing*, 27(1):71–82, 2014.

[191] Ramy Baly and Hazem Hajj. Wafer classification using support vector machines. *IEEE Transactions on Semiconductor Manufacturing*, 25(3):373–383, 2012.

[192] Cheng-Wei Chang, Tsung-Ming Chao, Jorng-Tzong Horng, Chien-Feng Lu, and Rong-Hwei Yeh. Development pattern recognition model for the classification of circuit probe wafer maps on semiconductors. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 2(12):2089–2097, 2012.

[193] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[194] Eric JM Rignot and Jakob J Van Zyl. Change detection techniques for ers-1 sar data. *IEEE Transactions on Geoscience and Remote Sensing*, 31(4):896–906, 1993.

[195] Osama Yousif and Yifang Ban. Improving urban change detection from multitemporal sar images using pca-nlm. *IEEE Transactions on Geoscience and Remote Sensing*, 51(4):2032–2041, 2013.

[196] Qian Xu and Lina J Karam. Change detection on sar images by a parametric estimation of the kl-divergence between gaussian mixture models. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2109–2113, 2013.

[197] Qian Xu and Lina J Karam. Change detection on sar images using divisive normalization-based image representation. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4339–4343, 2014.

[198] HD Cheng, Yen-Hung Chen, and Ying Sun. A novel fuzzy entropy approach to image enhancement and thresholding. *Signal Processing*, 75(3):277–301, 1999.

[199] Lorenzo Bruzzone and Diego F Prieto. Automatic analysis of the difference image for unsupervised change detection. *IEEE Transactions on Geoscience and Remote Sensing*, 38(3):1171–1182, 2000.

[200] Turgay Celik. Unsupervised change detection in satellite images using principal component analysis and k-means clustering. *IEEE Geoscience and Remote Sensing Letters*, 6(4):772–776, 2009.

[201] Maoguo Gong, Zhiqiang Zhou, and Jingjing Ma. Change detection in synthetic aperture radar images based on image fusion and fuzzy clustering. *IEEE Transactions on Image Processing*, 21(4):2141–2151, 2012.

[202] Turgay Celik. Change detection in satellite images using a genetic algorithm approach. *IEEE Geoscience and Remote Sensing Letters*, 7(2):386–390, 2010.

[203] Lu Jia, Ming Li, Peng Zhang, Yan Wu, and Huahui Zhu. Sar image change detection based on multiple kernel k-means clustering with local-neighborhood information. *IEEE Geoscience and Remote Sensing Letters*, 13(6):856–860, 2016.

[204] Georgios D Evangelidis and Emmanouil Z Psarakis. Parametric image alignment using enhanced correlation coefficient maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(10):1858–1865, 2008.

[205] Georgios Evangelidis. Iat: A matlab toolbox for image alignment, 2013.

[206] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[207] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning*, pages 647–655, 2014.

[208] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint*, 2016.

[209] Nirbhar Neogi, Dusmanta K Mohanta, and Pranab K Dutta. Review of vision-based steel surface inspection systems. *Journal on Image and Video Processing*, 2014(1):50, 2014.

[210] Kechen Song and Yunhui Yan. A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects. *Applied Surface Science*, 285:858–864, 2013.

[211] Ruoxu Ren, Terence Hung, and Kay Chen Tan. A generic deep-learning-based approach for automated surface inspection. *IEEE Transactions on Cybernetics*, 48(3):929–940, 2018.

[212] Wen Chen, Yiping Gao, Liang Gao, and Xinyu Li. A new ensemble approach based on deep convolutional neural networks for steel surface defect classification. *CIRP Conference on Manufacturing Systems*, 72:1069–1072, 2018.

[213] Li Yi, Guangyao Li, and Mingming Jiang. An end-to-end steel strip surface defects recognition system based on convolutional neural networks. *Steel Research International*, 88(2):60–68, 2017.

[214] Xian Tao, Dapeng Zhang, Wenzhi Ma, Xilong Liu, and De Xu. Automatic metallic surface defect detection and recognition with convolutional neural networks. *Applied Sciences*, 8(9):1575, 2018.

[215] Mang Xiao, Mingming Jiang, Guangyao Li, Li Xie, and Li Yi. An evolutionary classifier for steel surface defects with small sample set. *EURASIP Journal on Image and Video Processing*, 2017(1):48, 2017.

[216] Huijun Hu, Yuanxiang Li, Maofu Liu, and Wenhao Liang. Classification of defects in steel strip surface based on multiclass support vector machine. *Multimedia Tools and Applications*, 69(1):199–216, 2014.

[217] Qiwu Luo, Yichuang Sun, Pengcheng Li, Oluyomi Simpson, Lu Tian, and Yigang He. Generalized completed local binary patterns for time-efficient steel surface defect classification. *IEEE Transactions on Instrumentation and Measurement*, pages 1–13, 2018.

[218] Rongfen Gong, Chengdong Wu, and Maoxiang Chu. Steel surface defect classification using multiple hyper-spheres support vector machine with additional information. *Chemometrics and Intelligent Laboratory Systems*, 172:109–117, 2018.

[219] AO Martins Luiz, LC Pádua Flávio, and EM Almeida Paulo. Automatic detection of surface defects on rolled steel using computer vision and artificial neural networks. In *Annual Conference on IEEE Industrial Electronics Society*, pages 1081–1086, 2010.

[220] Kaixiang Peng and Xuli Zhang. Classification technology for automatic surface defects detection of steel strip based on improved bp algorithm. In *International Conference on Natural Computation*, volume 1, pages 110–114, 2009.

[221] Santanu Ghorai, Anirban Mukherjee, M Gangadaran, and Pranab K Dutta. Automatic defect detection on hot-rolled flat steel products. *IEEE Transactions on Instrumentation and Measurement*, 62(3):612–621, 2013.

[222] Shiyang Zhou, Youping Chen, Dailin Zhang, Jingming Xie, and Yunfei Zhou. Classification of surface defects on steel sheet using convolutional neural networks. *Material Technology*, 51(1):123–131, 2017.

[223] Jiangyun Li, Zhenfeng Su, Jiahui Geng, and Yixin Yin. Real-time detection of steel strip surface defects based on improved yolo detection network. *IFAC-Papers On Line*, 51(21):76–81, 2018.

[224] Jonathan Masci, Ueli Meier, Dan Ciresan, Jürgen Schmidhuber, and Gabriel Fricout. Steel defect classification with max-pooling convolutional neural networks. In *The 2012 International Joint Conference on Neural Networks*, pages 1–6, 2012.

[225] Vidhya Natarajan, Tzu-Yi Hung, Sriram Vaikundam, and Liang-Tien Chia. Convolutional networks for voting-based anomaly classification in metal surface inspection. In *IEEE International Conference on Industrial Technology*, pages 986–991, 2017.