Procedia Computer Science

Volume 53, 2015, Pages 345–355

2015 INNS Conference on Big Data

A classification algorithm for high-dimensional data

Asim Roy[1]

[1]*Department of Information Systems, Arizona State University, Tempe, Arizona, USA*

Asim.Roy@asu.edu

Abstract

With the advent of high-dimensional stored big data and streaming data, suddenly machine learning on a very large scale has become a critical need. Such machine learning should be extremely fast, should scale up easily with volume and dimension, should be able to learn from streaming data, should automatically perform dimension reduction for high-dimensional data, and should be deployable on hardware. Neural networks are well positioned to address these challenges of large scale machine learning. In this paper, we present a method that can effectively handle large scale, high-dimensional data. It is an online method that can be used for both streaming and large volumes of stored big data. It primarily uses Kohonen nets, although only a few selected neurons (nodes) from multiple Kohonen nets are actually retained in the end; we discard all Kohonen nets after training. We use Kohonen nets both for dimensionality reduction through feature selection and for building an ensemble of classifiers using single Kohonen neurons. The method is meant to exploit massive parallelism and should be easily deployable on hardware that implements Kohonen nets. Some initial computational results are presented.

Keywords: Kohonen nets; classification algorithm; online learning; feature selection; high-dimensional data

1 Introduction

The arrival of big and streaming data is forcing major changes to the machine learning field. In this new era, there are significantly more demands on machine learning systems - from the need to handle very large volumes of data and fast learning to the need for automation of machine learning that requires less expert assistance and for hardware deployment. Traditional artificial neural network algorithms have many properties that can meet these demands of big data and thus can certainly play a key role in the major transformations that are taking place. For example, many neural net algorithms are based on the concept of online, incremental learning that does not require simultaneous access to large volumes of data. This mode of learning not only resolves many computational issues, it also

Selection and peer-review under responsibility of the Scientific Programme Committee of INNS-BigData2015

doi:10.1016/j.procs.2015.07.311

removes the headache of correctly sampling from large volumes of data. It also makes neural net algorithms highly scalable (i.e. they can easily handle large volumes of data without running into computer memory limitations and learning (processing) time scales up linearly with data volume because of incremental learning) and provides them the capability to learn from all of the data. Neural network algorithms also have the advantage that they use simple computations that can be highly parallelized. These algorithms are already being implemented on hardware that allows parallel computations (Oh & Jung, 2004) and more powerful hardware is on the way (Monroe, 2014; Poon & Zhou, 2011; Furber et al., 2013). All these features of neural network algorithms positions the field to become the backbone of machine learning applications in the era of big and streaming data.

In this paper, we present a new neural network learning method that (1) can be parallelized at different levels of granularity, (2) addresses the issue of high-dimensional data through class-based feature selection, (3) learns an ensemble of classifiers using selected Kohonen neurons (nodes) from different Kohonen nets (Kohonen, 2001), and (4) can be easily implemented on hardware. For dimensionality reduction through feature selection, we train a number of Kohonen nets in parallel with streaming data to create some representative data points. (Note that stored data can also be streamed.) Using these Kohonen nets, we perform class-based feature selection (Roy et al., 2013). The basic criteria for feature selection are to select features for each class that (1) makes the class more compact, and (2) at the same time, maximize the average distance from the other classes. Once class-based feature selection is complete, we discard these Kohonen nets. In the second phase, we construct several new Kohonen nets in parallel in different feature spaces, again from streaming data, based on the selected features. Once trained, we then extract just the active neurons from these different Kohonen nets, add class labels to them and create an ensemble of Kohonen neurons for classification. In the end, we just retain a set of dangling active Kohonen neurons from different Kohonen nets in different feature spaces and discard all Kohonen nets.

The paper is organized as follows. Section 2 provides an overview of the concept of class-based feature selection and separability index of features. Section 3 has the algorithm for class-based feature selection from streaming data using Kohonen nets in parallel. Sections 4 provide details on how an ensemble classifier is constructed using neurons from different Kohonen nets. Section 5 has computational results for several high-dimensional problems and the conclusions are in Section 6.

2 Class-specific feature selection, separability index of features and dimensionality reduction

A fundamental challenge for machine learning is learning from high-dimensional data. A number of new methods have been developed for both online feature selection and feature extraction for high-dimensional streaming data (Yan et al., 2006; Hoi et al., 2012; Wu et al., 2010; Law et al., 2006). However, none of them are for class-specific feature selection. Since 1997, at various conferences, Roy had proposed methods that use a subset of the original features in class-specific classifiers and Roy et al. (2013) presents one such method. However, the method in Roy et al. (2013) does not work for streaming data. In class-specific feature selection, we find separate feature sets for each class such that they are the best ones to separate that class from the rest of the classes. The concept we use is identical to the one used by LDA and Maximum Margin Criterion (MMC) methods (Li et al. 2006) that maximize the between-class scatter and minimize the within-class scatter. In other words, those methods try to maximize the distance between different class centers and at the same time make the data points in the same class as close as possible. Our method, although not a feature extraction method, is based on the same concept.

In an offline mode, where a collection of data points is available, it is fairly easy to select features that maximize the average distance of data points of one class from the rest of the classes and also, at the same time, minimize the average distance of data points within that class. Roy et al. (2013) ranks and selects features on this basis and computational experiments show that it works quite well. However, that method cannot be used for streaming data where no data is stored. In the proposed method, we use the same concept for feature selection, but train multiple Kohonen nets from streaming data to do so. By training multiple Kohonen nets, we essentially create some representative data points for each class and that's how we resolve the dilemma of not having access to a collection of data points. Once we have a collection of representative data points (represented by certain Kohonen neurons in the Kohonen nets), it is easy to use the class-based feature selection method proposed in Roy et al. (2013). We train many different Kohonen nets, of different grid sizes and for different feature subsets, and we train them in parallel on a distributed computing platform. We use Apache Spark (2015) as our distributed computing platform, but other similar platforms can be used. A Kohonen net forms clusters and the cluster centers (that is, the active Kohonen net nodes or neurons) are equivalent to representative examples of the streaming data. We then use these representative examples to select features by class.

Suppose there are *kc* total classes. Our basic feature ranking principles are that (1) a good feature for class *k* should produce good separation between patterns in class *k* and those not in class *k, k = 1…kc*, and (2) also make the patterns in class *k* more compact. Roy et al. (2013) uses a measure called the separability index that is based on these concepts and can rank features for each class. Suppose d_{kn}^{in} is the average distance between patterns within class *k* for feature *n*, and d_{kn}^{out} the average distance between the patterns in class *k* and those not in class *k* for feature *n*. Roy et al. (2013) uses the Euclidean distance for distance measure, but other distance measures could be used. The separability index is given by $r_{kn} = d_{kn}^{out} / d_{kn}^{in}$. Roy et al. (2013) uses this separability index r_{kn} to rank order features of class *k* where a higher ratio implies a higher rank. The sense of this measure is that a feature *n* with a lower d_{kn}^{in} makes class *k* more compact and with a higher d_{kn}^{out} increases the separation of class *k* from the other classes. Thus, higher the ratio r_{kn} for a feature *n*, greater is its ability to separate class *k* from the other classes and better is the feature.

2.1 Why class-based feature selection? An example

Gene Number		Separability Indices by Class	
		AML	ALL
AML Features	Good		
758		82.53	2.49
1809		75.25	1.85
4680		39.73	2.82
ALL Features	Good		
2288		0.85	114.75
760		0.93	98.76
6182		0.8	34.15

Table 2.1 – Separability indices for a few features in the AMLALL gene expression dataset

We solved a number of high-dimensional gene expression problems to test this method. One such problem is to predict the type of leukemia (AML or ALL) from gene expression values (Golub et al., 1999). There are a total of 72 data samples and 7129 genes (features). Table 2.1 shows a few of the genes and their separability indices by class. For example, genes 758, 1809 and 4680 are good predictors of the AML class and their separability indices for the AML class are 82.53, 75.25 and 39.73 respectively. On the other hand, the corresponding separability indices for the same genes for the ALL class are quite low; they are 2.49, 1.85 and 2.82 respectively. Hence, these three genes are not very good predictors of the ALL class. Table 2.1 also shows three genes that are good predictors of the ALL class (genes 2288, 760 and 6182) and have high separability indices for the ALL class (114.75, 98.76 and 34.15). However, they are not good predictors of the AML class as shown by their low separability indices of 0.85, 0.93 and 0.8. This example illustrates the power of class-based feature selection and their potential usefulness in understanding a phenomenon.

3 Kohonen network based class-specific feature selection for streaming data

We introduce some notation here. Suppose the N-dimensional vector x, $x=(X_1, X_2,...,X_N)$ represents an input pattern in the streaming data; X_n denotes the n^{th} element of the vector x. Let FP_q denote the q^{th} feature subset, $q = 1…FS$, where FS is the total number of feature subsets. Let KN_q^g be the g^{th} Kohonen net of a certain grid size for the q^{th} feature subset, $q = 1…FS$, $g = 1…FG$, where FG is total number of different Kohonen net grid sizes. kc denotes the total number of classes and k is a class. To compute the separability indices in an efficient and fast manner for high-dimensional data, we assume that we have access to parallel distributed computing facilities such as Apache Spark (Franklin 2013). This is parallelization of computation at the highest level. Further parallelization of computation at a lower level can be achieved if we have hardware that implements Kohonen nets.

Suppose that we use 10 different grid sizes ($FG = 10$) and that we have the computing resources to build 500 Kohonen nets in parallel. In that case, FS would be 50 (=500/10) and $KN_1^1...KN_{50}^{10}$ would denote the different Kohonen nets. And suppose that there are 1000 features in the data stream ($N = 1000$). In that case, we would partition the feature set randomly into 50 subsets (FS=50) of 20 (=1000/50) features each. For simplicity, let's assume that the first feature partition FP_1 includes the features $X_1...X_{20}$, the second partition FP_2 includes the features $X_{21}...X_{40}$ and so on. For the Kohonen nets KN_1^g, $g = 1…10$, the input vector would consist of the features in the set FP_1, for the Kohonen nets KN_2^g, $g = 1…10$, the input vector would consist of the features in the set FP_2 and so on. Thus, for each feature subset FP_q, we would train 10 different Kohonen nets of different grid sizes. If there are just a few classes in the classification problem, smaller grid sizes should suffice (e.g. grid sizes of 9x9, 8x8, 7x7 and so on). If there are thousands of classes, then larger grid sizes would be required. We are using feature partitions essentially for efficiency and speed. These smaller Kohonen nets can be trained much faster and in parallel compared to a single one that uses thousands of features. And the reason for using different grid sizes for the same feature partition is to get different representative examples to compute the separability indices.

3.1 Assigning class labels to Kohonen neurons

Some, but not all, of the active nodes of a Kohonen net serve as representative training examples and these can be used to compute the separability indices. Only the winning neurons are counted as active nodes. Once the Kohonen nets stabilize, we process some more streaming data to

assign class labels to these active nodes. In this phase, we do not change the weights of the Kohonen nets but only keep count of how many times the input pattern of a particular class activated a particular neuron. For example, let's say we have two classes, A and B. For each active node, we keep a count of how many times it is activated by an input pattern from each of these two classes. Suppose one such neuron is activated by class A patterns 85 times and class B patterns 15 times. At this node, we can say that 85% of the input patterns belong to class A and 15% belong to class B. Since class A patterns are in the majority, we can simply assign this neuron to class A. This phase of labeling active nodes ends once the class ratios (percentages) at every active node have converged and all active nodes (neurons) can be assigned to classes. If no class has a significant majority, we ignore that active node.

After class labels are assigned to active nodes and some of the active nodes dropped, we construct a list of active nodes assigned to each class in each feature partition. For example, class A can have 50 active nodes assigned to it in a particular feature partition and class B can have 20 such active nodes. Note that these active nodes can be from different Kohonen nets of different grid sizes, although in the same feature partition. These active nodes are effectively representative examples of each class in these feature subspaces. Using these active nodes, we compute the separability index of each feature for each class.

3.2 An algorithm to compute the separability indices of features by class for feature ranking

*Step 1***.** Randomly partition the features into *FS* subsets.
*Step 2***.** Randomly initialize all Kohonen nets, of different grid sizes for each feature partition, in parallel.
*Step 3***.** Train all Kohonen nets in parallel using streaming data and partitioning the input vector according to the feature subsets allocated to each Kohonen net. Stop training when all the Kohonen nets converge.
*Step 4***.** Process some more streaming data through the stabilized Kohonen nets, without changing weights, to find the active nodes (winning neurons) and their class counts.
*Step 5***.** Assign each active node (neuron) to a class if the percentage class count for the most active class at that node is greater than some threshold. Discard active neurons whose percentage counts are below the threshold.
*Step 6***.** Create a list of the remaining active nodes by class for each feature partition FP_q, q = 1…*FS*.
*Step 7***.** Compute the separability indices of the features separately in each feature partition FP_q, q = 1…*FS*, using the neurons in the remaining active node list as representative examples for the classes.
*Step 8***.** Repeat steps 1 through 7 a few times and compute the average separability indices of the features.
*Step 9***.** Rank features based on the average separability indices.

3.3 On feature combinations to explore to build classifiers and on the concept of buckets of features

Once rank ordered, Roy et al. (2013) sequentially combines these features one at a time, starting with the top ranked feature, and estimates the classification error rate for each combination of features by approximately fitting a set of hyperspheres to the data points. Roy et al. (2013) takes the highest ranked feature and estimates the error rate for a hypersphere classifier. It then takes the top two

ranked features and estimates its error rate, then top three and so on. It then selects a set of feature combinations that has the lowest error rates and builds more accurate hypersphere classifiers with those feature sets. With a parallel distributed computing system, we actually have the resources to build separate classifiers in parallel for different feature combinations. For this method, we can build Kohonen nets of different grid sizes for the top ranked feature, for the top two features, for the top three features and so on and these can all be done in parallel. Once these Kohonen nets are created in parallel, we can then test and select the best feature combinations that have the highest accuracy.

For the computational results reported here, we used a slightly different heuristic. Instead of combining one feature at a time, we actually combined several of the ranked features at a time and created corresponding Kohonen nets. For example, we created a bucket of features using the top 10 features, another bucket with top 15 features, a third bucket with top 20 features and so on. We then constructed Kohonen nets of various grid sizes in parallel for these different feature buckets. We also varied these bucket sizes randomly in our computational experiments. We effectively ignore features with separability index values lower than 1 since they are poor discriminators for a class. Note that our heuristic for combining features may not find the optimal combination of features that produces the best classifier. After class-based ranking of features is complete, we proceed with training of the final set of Kohonen nets for classification. All of the Kohonen nets built so far are discarded at this point.

4 Construction of a classification system based on an ensemble of Kohonen neurons

Here we present an algorithm to create a classification system based on an ensemble of Kohonen neurons. These Kohonen neurons are selected from various Kohonen nets in different feature spaces and of different grid sizes. Note that, at the end of this phase, all of the Kohonen nets are discarded and just a selected set of Kohonen neurons retained.

4.1 On assigning neurons (active nodes) to classes on a majority basis

The final Kohonen nets, in different feature spaces and of different grid sizes, are trained with streaming data. Once these Kohonen nets converge and are stable, we process some more streaming data, without changing the weights of the Kohonen nets, to get the class count percentages at each active node in a manner similar to the first phase. When we have stable class count percentages at each of the active nodes, we do some pruning of the active nodes similar to phase one. Thus, active nodes with small total counts are discarded and nodes where one particular class has a clear majority (e.g. 70% majority) are the only ones that are retained. In a sense, we pick out good neurons where clear majority exists for a class.

4.2 About the radius of a Kohonen neuron

Our classification system is based on the notion of radius of an active node (neuron) that determines approximately the boundary within which it is the winning neuron. This concept is critical to our approach because we just retain a few active nodes from a Kohonen net and discard the rest of the network. Once we discard the rest of the Kohonen net, there is no way to determine the winning or the best neuron for an input pattern. Hence, the radius is a substitute way to determine if a node would be the winning node. To determine the radius, we initialize the radii of the active nodes to zero and then update them by processing some more streaming data until the radii are stable for all active

nodes. We update the radius in the following way. If the streaming input pattern has the same class as the class of the active node, we compute the distance of the input pattern from the node and update the radius of the node if the distance is greater than the current radius. We update the radii of all active nodes before we discard the Kohonen nets.

4.3 An algorithm for training the final set of Kohonen nets for classification from streaming data

*Step 1***.** Initialize bucket number *j* to zero.
*Step 2***.** Increment bucket number *j* (*j* = *j* + 1) and add a few more top ranked features by class *k* to $(j\text{-}1)^{th}$ bucket to create the j^{th} bucket.
*Step 3***.** Randomly initialize final Kohonen nets, in parallel in a distributed computing system, of different grid sizes for each class *k* and j^{th} feature set. If the remaining ranked features have indices greater than 1, go back to step 2 to set up other Kohonen nets for the other feature buckets. Otherwise, go to step 4.
*Step 4***.** Train all Kohonen nets in parallel using streaming data and by selecting appropriate parts of the input pattern according to the feature subset that applies to each particular Kohonen net. Stop training when all the Kohonen nets converge.
*Step 5***.** Process some more streaming data through the stabilized Kohonen nets, without changing weights, to find the set of active nodes (neurons) for each class *k* and each bucket *j*. Also get the class counts of the active nodes and stop when class percentages become stable for all active nodes.
*Step 6***.** Assign an active node to the majority class if the class percentage for the majority class and the absolute class count are above minimum thresholds.
*Step 7***.** Process some more streaming data to compute the radius of each of the active nodes. Stop when the radii or widths are stable.
*Step 8***.** Retain just the active nodes which satisfy the thresholds and discard all the other nodes from all of the Kohonen nets.

4.4 The classifier is an ensemble of dangling Kohonen neurons (not Kohonen nets)

In the end, we have an ensemble of Kohonen neurons in different feature spaces and we use this ensemble for classification. Studies have shown that, in general, combining multiple classifiers can improve overall performance on a problem. Rokach (2009) has an excellent review and taxonomy of ensemble learning methods. Base classifiers can be combined in a variety of ways for final prediction. We currently use the following measures to determine the final classification of a test example.

a. Maximum Probability - Find the ensemble neuron with the highest probability (or confidence) and assign its class to the test example.
b. Minimum Distance - Find the neuron closest to the test example and assign its class to the test example.
c. Majority voting – Here we use votes from maximum probability and minimum distance neurons in each of the feature spaces and then determine the class of the test example based on the majority vote for a class.

5 Computational Results

Gene expression problems have received a lot of attention recently. These problems are characterized by high dimensionality (the number of features or genes are usually a few thousand) and a small number of training examples. We used seven widely referenced gene expression datasets to test the performance of this algorithm. Table 5.1 summarizes the main characteristics of these datasets. We randomly generated our training and test sets from the available data by randomly selecting nine-tenths of it for training and using the rest for testing. This random allocation was repeated 50 times and the average results of the 50 runs are reported here. In our implementation, we simulated online learning by reading one input pattern at a time.

	No. of genes	No. of classes	No. of examples
Leukemia (AML-ALL) (Golub et al. 1999)	7129	2	72
Central Nervous System (Pomeroy et al. 2002)	7129	2	60
Colon Tumor (Alon et al. 1999)	2000	2	62
SRBCT (Khan et al. 2001)	2308	4	63
Lymphoma (Alizadeh et al. 2000)	4026	3	62
Prostrate (Singh et al. 2002)	6033	2	102
Brain (Pomeroy et al. 2002)	5597	5	42

Table 5.1 – Characteristics of the gene expression problems

5.1 Parameter Settings

No parameters were fine-tuned for any of the problems solved with this algorithm. We did not use a parallel distributed computing platform for these experiments, although an implementation on Apache Spark is underway. All problems were solved on a desktop and laptops. We used the following parameter settings for this set of problems. For Kohonen nets, we used grid sizes of 9x9, 8x8, 7x7, 6x6, 5x5, 4x4 and 3x3 ($FG = 7$). For feature selection, each feature subset had 10 features. Minimum threshold for class percentage was 70%.

5.2 Experimental results - Feature selection

Table 5.2 shows the average number of features used by this method for the gene expression problems. As one can observe, the proposed method is fairly good at identifying a small set of genes (features) among the thousands. So dimension reduction through feature selection works well with this method.

	Total no. of attributes	Average No. of features used	% of features used
Leukemia (AML-ALL)	7129	20	0.28%
Central Nervous System	7129	10	0.14%
Colon Tumor	2000	31	1.55%
SRBCT	2308	54	2.34%
Lymphoma	4026	24	0.60%
Prostrate	6033	74	1.23%
Brain	5597	28	0.50%

Table 5.2 – Average number of features used by the Kohonen neuron ensemble method for the gene expression problems.

5.3 Experimental evaluation of the Kohonen neuron ensemble classifier system

Here we present experimental results for the Kohonen ensemble algorithm that consists of (1) class-specific feature selection, and (2) training an ensemble of Kohonen neurons for classification using the selected features. We compare the performance of the algorithm with other classification algorithms available in Apache Spark's machine learning library MLlib (2015). For Spark testing, we similarly split the data randomly into training and test sets and repeated 50 times. Table 5.3 shows the average error rates and standard deviations for a variety of algorithms in Spark's MLlib. Both SVMwithSGD and LogRegWithSGD use the stochastic gradient descent (SGD) method to train the classifiers. All of these algorithms were used with their default parameters without any feature selection. SVMwithSGD and LogRegWithSGD only work for two class problems, hence they don't have results in the table for the multiclass problems SRBCT, Lymphoma and Brain.

	Kohonen ensemble	SVMwithSGD	NaiveBayes	LogRegWithSGD	RandomForest
Leukemia (AML-ALL)	1.14 (3.96)	4.4 (7.0)	10.26 (8.0)	10.29 (12)	12.8 (10)
Central Nervous System	28 (16.47)	33.25 (14)	42.84 (11)	36.25 (13)	41.26 (11)
Colon Tumor	10.67(9.48)	17.56 (11)	8.33 (12)	12 (12)	18.67 (12)
SRBCT	0.67 (3.33)	-	7.33 (12)	-	21.33 (18)
Lymphoma	0.67(3.33)	-	1.9 (3)	-	5.67 (10)
Prostrate	5.6 (7.12)	13.64 (5)	36.97 (12)	13.4 (10)	19.6 (13)
Brain	20.0 (11.18)	-	18 (19)	-	47.5 (25)

Table 5.3 – Average test error rates and standard deviations for various classification algorithms of Apache Spark MLlib (2015).

6 Conclusions

We have presented here a new learning method for high-dimensional stored and streaming data. Since it is an online method, stored data would need to be streamed. Online methods are highly scalable because they don't need simultaneous access to all of the training data and learning (processing) time scales up linearly with data volume. Another advantage is that they can learn from all of the data and need not sample from it. The method can be parallelized at various levels. It can be easily parallelized on a distributed computing platform, such as Apache Spark, and also implemented at the massively parallel computing level of the neural hardware. A neural hardware implementation would be very suitable for Internet of Things (IoT) applications where extremely fast learning and response is desired for high-speed streaming data. A neural hardware implementation can also process stored data at a very high speed. In addition, as shown by our experimental results, the method, in particular, is very powerful in reducing the dimensions of a high-dimensional problem.

References

Oh, K. S., & Jung, K. (2004). GPU implementation of neural networks. *Pattern Recognition*, *37*(6), 1311-1314.

Monroe, D. (2014). Neuromorphic computing gets ready for the (really) big time. *Communications of the ACM*,*57* (6), 13-15.

Poon, C. S., & Zhou, K. (2011). Neuromorphic silicon neurons and large-scale neural networks: challenges and opportunities. *Frontiers in neuroscience*, *5*.

Furber, Steve; Lester, David; Plana, Luis; Garside, Jim; Painkras, Eustace; Temple, Steve and Brown, Andrew. Overview of the SpiNNaker system architecture. *IEEE Transactions on Computers*, vol.62, Issue 12, pp.2454-2467, December 2013

Kohonen, T. (2001). *Self-organizing maps* (Vol. 30). Springer.

Roy, A., Mackin, P., Mukhopadhyay, S. (2013). Methods for Pattern Selection, Class-specific Feature Selection and Classification for Automated Learning, *Neural Networks*, Vol. 41, pp. 113-129.

Yan, J., Zhang, B., Yan, S., Liu, N., Yang, Q., Cheng, Q., ... & Ma, W. Y. (2006). A scalable supervised algorithm for dimensionality reduction on streaming data. *Information Sciences*, *176*(14), 2042-2065.

Hoi, S. C., Wang, J., Zhao, P., & Jin, R. (2012, August). Online feature selection for mining big data. In *Proceedings of the 1st international workshop on big data, streams and heterogeneous source mining: Algorithms, systems, programming models and applications* (p. 93-100). ACM.

Wu, X., Yu, K., Wang, H., & Ding, W. (2010). Online streaming feature selection. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (pp. 1159-1166).

Law, M. H., & Jain, A. K. (2006). Incremental nonlinear dimensionality reduction by manifold learning. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *28*(3), 377-391.

Li, X. R., Jiang, T., & Zhang, K. (2006). Efficient and robust feature extraction by maximum margin criterion. *Neural Networks, IEEE Transactions on*, *17*(1), 157-165.

Franklin, M. (2013, October). The Berkeley Data Analytics Stack: Present and future. In *Big Data, 2013 IEEE International Conference on* (pp. 2-3). IEEE.

Golub, T., Slonim, D., Tamayo, P. et al. (1999). Molecular classification of cancer: Class discovery and class prediction by gene expression. *Science*, 286, 531–537.

Rokach, Lior. (2009). "Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography." *Computational Statistics & Data Analysis* 53.12 (2009): 4046-4072.

Pomeroy, S. L., Tamayo, P., Gaasenbeek, M., Sturla, L. M., Angelo, M., McLaughlin, M. E., ... &

Golub, T. R. (2002). Prediction of central nervous system embryonal tumour outcome based on gene expression. *Nature*, *415*(6870), 436-442.

Alon, U., Barkai,N., Notterman, D., Gish, K., Ybarra, S., Mack, D., and Levine, A. J. (1999). "Broad patterns of gene expression revealed by clustering analysisof tumor and normal colon tissues probed by oligonucleotide arrays." *Proc Natl Acad Sci* U S A, 96(12):6745-50.

Khan, J., Wei, J. S., Ringner, M., Saal, L. H., Ladanyi, M., Westermann, F., ... & Meltzer, P. S. (2001). Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature medicine*, *7*(6), 673-679.

Alizadeh, A. A., Eisen, M. B., Davis, R. E., Ma, C., Lossos, I. S., Rosenwald, A., ... & Staudt, L. M. (2000). Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature*, *403*(6769), 503-511.

Singh, D., Febbo, P. G., Ross, K., Jackson, D. G., Manola, J., Ladd, C., ... & Sellers, W. R. (2002). Gene expression correlates of clinical prostate cancer behavior. *Cancer cell*, *1*(2), 203-209.

Apache Spark MLlib. (2015). https://spark.apache.org/docs/1.1.0/mllib-guide.html; https://spark.apache.org/docs/latest/mllib-guide.html; https://spark.apache.org/mllib/