Improving Ontology Alignment Using Machine Learning Techniques

by

Tariq M Nasim

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2022 by the
Graduate Supervisory Committee:

Srividya Bansal, Chair
Ayan Banerjee
Alexandra Mehlhase

ARIZONA STATE UNIVERSITY

May 2022

ABSTRACT

Ontologies play an important role in storing and exchanging digitized data. As the need for semantic web information grows, organizations from around the globe has defined ontologies in different domains to better represent the data. But different organizations define ontologies of the same entity in their own way. Finding ontologies of the same entity in different fields and domains has become very important for unifying and improving interoperability of data between these multiple domains. Many different techniques have been used over the year, including human assisted, automated and hybrid. In recent years with the availability of many machine learning techniques, researchers are trying to apply these techniques to solve the ontology alignment problem across different domains. In this study I have looked into the use of different machine learning techniques such as Support Vector Machine, Stochastic Gradient Descent, Random Forest etc. for solving ontology alignment problem with some of the most commonly used datasets found from the famous Ontology Alignment Evaluation Initiative (OAEI). I have proposed a method OntoAlign which demonstrates the importance of using different types of similarity measures for feature extraction from ontology data in order to achieve better results for ontology alignment.

ACKNOWLEDGMENTS

My heartfelt gratitude to Dr. Srividya Bansal for her guidance and encouragement throughout the project. I also want to extend my gratitude to my committee for their guidance in my research and for taking the time to serve as committee members.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

The use of Resource Description Framework (RDF) data has increased in recent years due to the involvement and contributions of many organizations like DBPedia. RDFs has been defined as a framework for data interchange on the web (W3C). RDFs primarily store metadata information about a class of data that can be easily read by other web entities and help determine the relationship with a particular class among other classes. Ontology is used to connect such classes of data and to define the characteristics of distinct classes of data. Ontology alignment is the process of finding relationships and similarity between different ontologies from the same domain or different domains. Though ontology alignment has been studied a lot in recent years, automated ontology alignment is still a difficult process, especially when it comes to solving problems from different domains. Also, most approaches found in the literature still require advanced human interventions in addition to the automated procedure. The purpose of current research is to find an automated ontology alignment that requires no human interaction and which is as fast as or better than the existing ones.

1.1 Problem Statement

Ontologies contain not only the information itself but also the metadata of the information, for example the properties of that particular class of information, the relationship between that and other data and the modality of that class of data. For this reason, in order to evaluate ontology alignment it is important to take special

consideration to convert ontologies into a representation that can be used to match with other classes. For using machine learning techniques most researchers use vector representations of ontologies which helps design the inputs for machine learning or deep learning models. One difficult task for ontology alignment is to get the vector representation that accurately resembles all necessary properties and data of the entities. Another one is to handle the diversity of two different entities that might come from different domains and might contain completely different types of properties. Almost all the current methods require intensive human intervention which is very time-consuming and also "prone to" human errors. In our current research we are looking for a fully automated ontology matcher which can perform well in both same-domain and cross-domain areas.

1.2 Challenges

There are several reasons why ontology alignment is a difficult task.

First reason is the huge amount of semantic web data available. Because of the vastness of data, it is very difficult and time consuming to perform the alignment tasks by humans. Even with the help of machines, it is quite difficult to achieve accurate alignments.

Heterogeneity of data is also a common problem for not only ontology alignment, but also for researchers working on fields like robotics, big data etc. This problem arises due to the different process of collecting data. Due to the diverse nature of data it is difficult to combine them under same classification.

Lack of interoperability is another challenge for ontology alignment and semantic web in general. Many different organizations stores the same entity in different formats. It is therefore difficult to create linked entities and match similar entities from multiple domain. Little initiavies have been taken to overcome this problem of standardizing

the format of semantic web data. Ontology Alignment Evaluation Initiativ (OAEI) is one such initiative that addresses some of this interoperability issues. For example, the conference track dataset provides collection of ontologies describing the domain of organising conferences. The largebio track consists of data from three different groups: Foundational Model of Anatomy (FMA), SNOMED CT, and the National Cancer Institute Thesaurus (NCI). These datasets allows the researchers to build automated methods for ontology alignment.

Ontology matching is very time consuming and most of the times not accurate enough. Though there has been some initiatives like OAEI to help develop faster and more accurate methods for ontology alignment, only limited number of teams have developed tools and techniques that can complete required tasks for the largebio track data in 2020. Vast amount of knowledge is required to develop an efficient semantic web analyzer or ontology matcher.

Missing background knowledge is another problem in ontology matching. Although there are a huge amount of data available from different organizations, most of the times the required background knowledge gets lost in the process of encoding the data in specific format which is suitable for storing and sharing. This happens because when human experts encode those data, they are not able to convert all domain knowledge into the specific data or entity in question. For ontology alignment it is important to retain those background knowledge.

Ontology alignment is not deterministic. When different matches are presented by different alignment methods, there is no way to decide which one should be considered most accurate. The performance of one alignment technique varies based on the nature of data, domain of data, choice of parameters used in the ontology matcher etc. Therefore a matcher used on a specific dataset can not be used deterministically on another dataset. The lack of ground truth is another reason for this problem. Some

techniques are available to produce ground truth for ontology alignments, but it is not easy to determine their effectiveness and accuracy. Also, most of these techniques require intensive human intervention.

Next section describes how our method OntoAlign addresses these challenges.

1.3 Challenges Addressed in OntoAlign

In our proposed method, OntoAlign, we have addressed some of the challenges described above.

OntoAlign makes the process of ontology alignment automatic, which requires zero human involvement. In our method, all the steps including data processing, feature extraction, machine learning model training and alignment prediction, no human interaction is required.

OntoAlign can handle ontologies from heterogeneous sources and produces results that are comparable with state of the art methods. We have tested this using two different publicly available datasets from Ontology Alignment Evaluation Initiative (OAEI)[1].

Even though different organizations define the same ontologies in way, these can be resolved by using the feature extraction step of OntoAlign. We have used a large number of similarity measures including some language-based similarities, which enables the identification of similarities between ontologies defined using different terminologies.

Structural difference in ontologies defined in different sources is another issue that is considered in OntoAlign. By using the hierarchical information and meta-data of each ontologies, e.g. the parent and path of each entity, OntoAlign gives more accurate results than other methods that compares only the textual similarities.

[1]https://oaei.ontologymatching.org/

Combining all the above, OntoAlign gives a comprehensive ontology alignment method which produces results that is superior to most of the methods' available in this field. The comparative results with other methods are presented in 7.3 and 7.4.

1.4 Overview of Proposed Approach

Due to all the challenges mentioned in 1.2, it is very difficult to find an ontology alignment approach that will be suitable for all domains. Researchers have explored different approaches for ontology matching. We have discussed those in chapter 3. After studying other approaches, we propose OntoAlign, where we have decided to use different machine learning techniques to solve the problem with the help of multiple similarity measures between the entities of the ontologies. The reason behind using machine learning is, if we can train a model based on available data, it should be a simple task to find similarities between a new pair of ontologies.

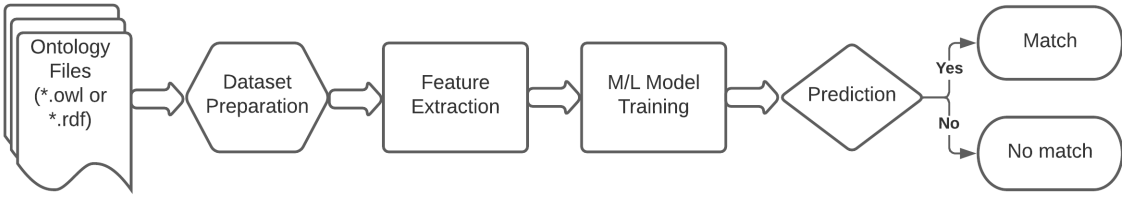


Figure 1.1: Overview of the Proposed Solution

For each ontology file, we first extract all the classes and properties. From the classes and properties, we generate all the class-to-class and property-to-property pairs and assign a value indicating whether the pair is a match or not. This is the data preparation step as shown in figure 1.1. We call this the entity-alignment tuple. Then, based on each of these pairs, we calculate different similarity measures. After normalizing the values of the similarity measures and combining them, we prepare a feature vector representing these two entities (two classes or two properties).

The feature vectors extracted from the entity-alignment tuple are then fed into our machine learning models. We have designed several machine learning models and trained them based on the entity-alignment tuples we get from the previous step. The overall process is shown in figure 1.1. The source code of our tool OntoAlign can be found on github[2].

Different machine learning techniques have been proven quite successful in classifying different types of data especially text-based data. The ontology data that we have are also represented in textual form, though in a specific structure. In order address the structural and hierarchical representation of the data we initially extract the entities from the ontologies along with their hierarchical information (parent and path of the entities).

For feature extraction, we have used several text-similarity measures including those related to word-to-word relationship (word2vector). Since ontology data have specific format, we assumed that instead of feeding just the string similarity based information, it would be more fruitful to also extract and feed the syntactic and structural information as features into the machine learning models. While other researcher have focused on using only some of these features, our approach is to use all the available similarity measures. This is the primary contribution of our current research.

1.5 Organization

The whole study is divided into 10 chapters. The overview of this work is already expressed in chapter 1. Chapter 2 covers the background information relevant to the study. It includes the basics of Ontologies and their importance in Semantic Web. Chapter 3 describes the prior works done in ontology alignment. The structure and organization of the data and the necessary pre-processing is discussed in chapter

[2]https://github.com/tnasim/ontoalign

4. Chapter 5 contains descriptions of different types of similarity measures and the process used for feature extraction from the ontologies using these similarity measures. In chapter 6 we discuss different types of machine learning approach and how we decide on using the ones that are most useful in ontology alignment. Chapter 7 discusses the final process of ontology matching based on the trained machine learning models and present the results with comparison to other state of the art methods. Chapter 8 describes an ablation study, i.e. how we have decided on using the feature extraction methods and the specific parameters chosen for different machine learning methods. Possible future extensions to this study are provided in chapter 9. Chapter 10 gives an overview of the contributions of this study and provides the concluding remarks. All the references are listed after this chapter.

Chapter 2

BACKGROUND

2.1 Semantic Web

In order to understand ontologies, we first need to understand the field of Semantic Web. Semantic web is a research field which primarily focuses on creating a machine-readable web of information which enables higher level of interoperability between information stored on the web as opposed to the currently prevalent human understandable form of web which makes it very difficult to exchange data between multiple entities on the internet [Bernstein *et al.* (2016)]. In other words, the goal of Semantic Web is the creation, maintenance and application of tools and methods that enables machine-readable form of information which can be easily exchanged over the web with zero or minimum human involvement. The Semantic Web is sometimes envisioned as the enhanced form of World Wide Web [Hitzler (2021)]. Since the initiation of this field with the 2001 Scientific American article by T Berners-Lee [Berners-Lee *et al.* (2001)], there has been many publications in this field primarily in The Semantic Web journal[1] the Journal of Web Semantics[2] and the proceedings of the annual International Semantic Web Conference[3]. Many concepts and tools have been defined and standardized over the years to support the field of semantic web, namely the Web Ontology Language (OWL), Resource Description Framework (RDF), Rule Interchange Format (RIF), SPARQL, Uniform Resource Identifier, XML etc. In his talk at a 2005 W2C conference, Oreste Signore presented an overview of the Semantic

[1] http://www.semantic-web-journal.net/
[2] https://www.journals.elsevier.com/journal-of-web-semantics
[3] http://swsa.semanticweb.org/content/international-semantic-web-conference-iswc

Web Stack which shows the different components of The Semantic Web [4].

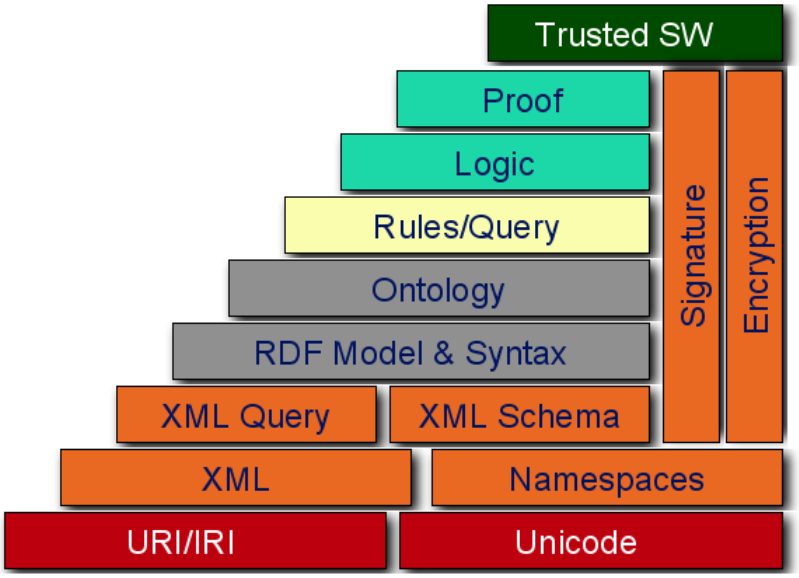


Figure 2.1: The Semantic Web Stack [Signore *et al.* (2005)]

2.2 Ontology

An ontology is a set of structural rules designed to represent concepts in order to perform logic-based operations to retrieve or infer new information. It is "a formal, explicit specification of shared conceptualization" [Guarino *et al.* (2009)], where conceptualisation is an abstract model of some phenomenon in the world. Ontologies were created to facilitate the sharing of knowledge and its reuse [O'Leary (2005)]. They are used for organization of knowledge and for communication between computing systems, people, computing systems and people.

Ontologies deal with the following kinds of entities: classes, properties and individuals. A class (concept) of ontology is a collection of objects, i.e., "Person" (the class of all people) or "Car" (the class of all cars). Property (attribute) describes char-

[4]http://www.w3c.it/talks/2005/openCulture/slide7-0.html

```
1 <owl:Class rdf:ID="Person"/>
2 <owl:Class rdf:ID="Man">
3         <rdfs:subClassOf rdf:resource="#Person"/>
4         <rdfs:disjointWith rdf:resource="#Woman"/>
5 </owl:Class>
6 <owl:Class rdf:ID="Woman">
7         <rdfs:subClassOf rdf:resource="#Person"/>
8         <rdfs:disjointWith rdf:resource="#Man"/>
9 </owl:Class>
10 <owl:Class rdf:ID="Father">
11         <rdfs:subClassOf rdf:resource="Man"/>
12         <owl:Restriction owl:minCardinality="1">
13                 <owl:onProperty rdf:resource="#hasChild"
14         </owl:Restriction>
15 </owl:Class>
16 <owl:ObjectProperty rdf:ID="hasChild">
17         <rdfs:domain rdf:resource="#Parent"/>
18         <rdfs:range rdf:resource="#Person"/>
19 </owl:ObjectProperty>
```

Figure 2.2: A Sample Ontology File

acteristics of a class or relations between classes, i.e., “has as name” or “is created by”. Individual (instance) is a particular instance or object represented by a concept, i.e., “a human cytochrome C” is an instance of the concept “Protein” [Euzenat *et al.* (2007)]. A sample ontology file is shown in Figure 2.2. First a ’Person’ entity is defined and then other ontology like ”Man” and ”Women” are defined that are childs of ”Person”. The ontology ”Father” is defined as a child of ”Man” which also has a property ”hasChild”. ”hasChild” property is defined at line 16 which defines relationship between two persons.

2.3 Ontology Alignment

Ontology matching is a process of establishing correspondences between semantically related entities in different ontologies [Euzenat *et al.* (2007)]. A set of correspon-

dences (equivalence, subsumption, disjointness) between ontologies elements is called an alignment. Ontology matching can be applied in many different subject areas: Semantic Web, Peer-to-Peer (P2P) systems, learning systems, multi-agent systems [Otero-Cerdeira *et al.* (2015)] [Shvaiko and Euzenat (2005)]. In the Semantic Web, logical conclusions from data is extracted using ontologies. Many ontologies on the same subject areas have been created recently. The format of these ontologies are different and as a result it is difficult to exchange information between them. So, application of ontology matching is important [Euzenat *et al.* (2007)]. In P2P systems ontology matching is used to reduce the semantical heterogeneity (differences in the interpretation of the meaning) between the queries of the users to system [Atencia *et al.* (2011)]. In learning systems ontology matching is a way to ease the knowledge share and reuse [O'Leary (2005)]. In multi-agent systems, ontology matching is used for interaction of different agents [Mascardi *et al.* (2011)]. Ontology matching can also be used for schema mapping during data integration [Li *et al.* (2015)]. Data integration is a process of combining the heterogeneous data sources into a unified view. Schema mapping is a process of establishing correspondences between elements of two different semantically related schemas (i.e. database schemas) [Rahm and Bernstein (2001)]. Ontology matching can help to resolve semantical heterogeneity during schema mapping, for instance, if the schemas have ontologies as metadata or external domain knowledge [Hlaing (2009)].

A sample ontology alignment process is described in Figure 2.3. It shows how two movies can be defined by two different sources with different names for the movies. We can also see that the relationships between the entities for each of the ontologies are different: one defines the movie as an 'isA' relationship and the other defines it as a 'type'. The number of childs can also differ. For the first ontology, both of the lead characters are described, but for the second one only the primary character

for the movie is listed. Because of these differences, it becomes difficult to identify similarrities between the entities from different ontologies.

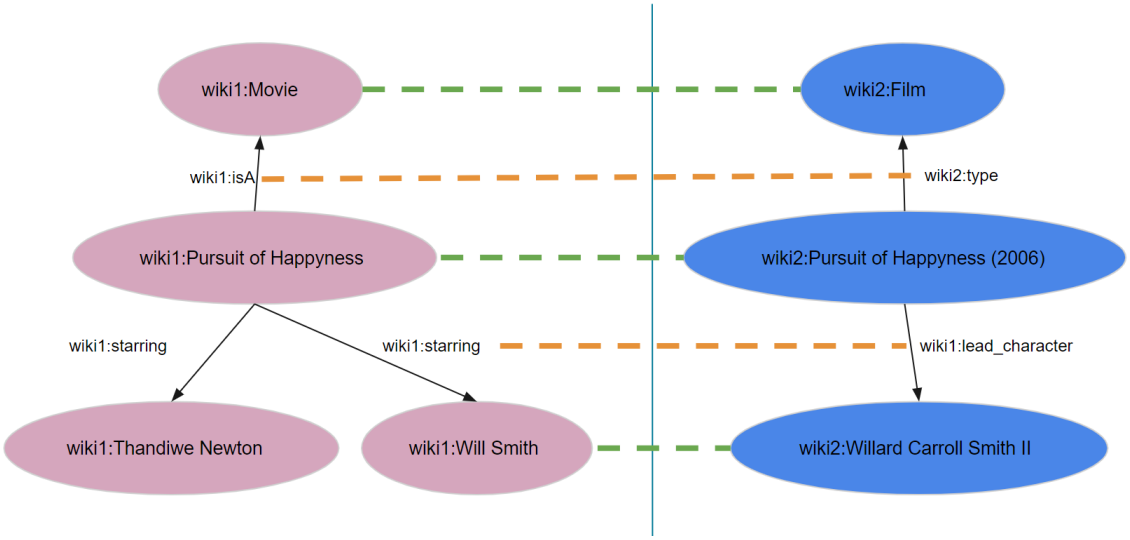


Figure 2.3: A Sample Ontology Alignment for Two Movies

As Semantic Web technologies are expanding and becoming more popular, the amount of data that needs to be represented grows proportionally, as does its complexity. Since web technologies are designed to be decentralised, redundancy inevitably occurs among knowledge bases. For this reason, finding methods to align equivalent data or their model is crucial. This is the goal of our research, which is described in this study.

2.4 Multi-Domain Ontology Alignment

Ontologies differ greatly based on the domain they are taken from. For example, the anatomy of a human body and the anatomy of that of a mouse has certain similarity and they might share some of the classes and properties that can be considered similar. But, since they are from different type of animals, the naming of the classes and properties can be different. Also, there can be ontologies for the same concept from different organizations and their way of defining the classes and properties might

vary, but an ideal ontology matcher should be able to identify the similar concepts and distinguish among the different concepts with high percentage. Multi-domain ontology alignment is the process of finding similarities among ontologies from different domains.

2.5 The Focus of This Research

Ontology, though a much older concept, in the area of semantic web, it can be visualized as a building block of the broader domain of Semantic Web, especially since we are trying to represent information in a structured way. Ontologies support the The Semantic Web by defining the concepts and relationships used to describe and represent an area of knowledge. Besides defining the terminologies for specific contexts, the constraints needed on the properties and logical representations of the properties, ontologies also define how to find equivalence among the terms in different ontologies.

Different types of approaches for ontology alignment are discussed in chapter 3. We have chosen a supervised learning based ontology alignment technique which is suitable for cross-domain matching of ontologies. The position of our research in the domain of Semantic Web is represented hierarchically in Figure 2.4.

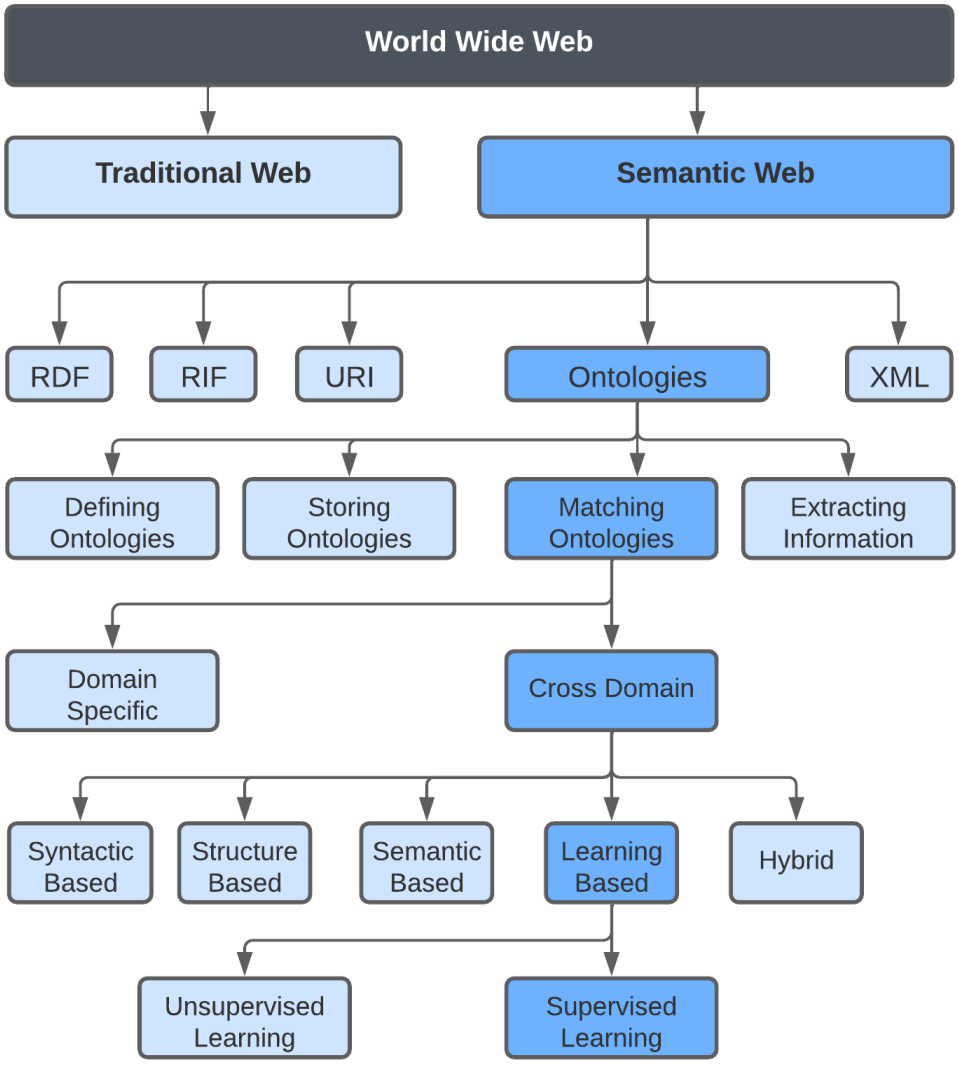


Figure 2.4: Focus of Current Research in the Domain of Semantic Web

Chapter 3

RELATED WORK

A lot of different approaches have been used by researchers over the years to solve the ontology alignment problem. A well known initiative is the Ontology Alignment Evaluation Initiative (OAEI 2021) which provides measures to evaluate different ontology matching systems. They held an annual competition to evaluate submitted ontology alignment systems. These systems can be divided into the following categories. First we can divide the ontology alignment systems into two categories, learning based and rule based. The rule-based approaches can further be categorized as syntactic based approaches, structural based approaches and semantic based approaches.

Syntactic based approaches take into account the names, descriptions and different terms of the entities. String based approach is the most common Syntactic based ontology alignment. The string based approach creates a bag of words based on the names and descriptions of the entities and compares similarities of these words to determine similarity between entities. Limes [Ngomo and Auer (2011)], AgreementMakerLight [Faria *et al.* (2013)] and COMA++ [Aumueller *et al.* (2005)] are tools that use different types of distance metrics.

Structural based ontology alignment uses some kinds of external tool like WordNet in order to find similarity between entities. They use different metrics to find similarity based on the graph representation of the entities and their relationship. Since they depend highly on external tools like WordNet, they are not extensible for

multilingual domains and specialized domains like biomedical data. Wu palmer [Wu and Palmer (1994)], Resnik similarity [Resnik (1995)] etc are common metrics used in structural based approaches.

Semantic based approaches make use of logical models like propositional satisfiability (SAT), Description Logic, Rule based inference etc. The description logics are used to determine whether an entity contains another one and similar other subsumption properties. The most common semantic based approaches are Paris [Suchanek *et al.* (2011)], CtxMatch/CtxMatch2 [Bouquet *et al.* (2003)], LogMap/LogMap2 [Jiménez-Ruiz *et al.* (2013)] and S-Match [Giunchiglia *et al.* (2004)].

Now, we look into the learning-based approaches. In recent years, many researchers have focused on applying supervised machine learning techniques to solve the ontology alignment problem. GLUE [Doan *et al.* (2003)], Yet Another Matcher (YAM++) [Ngo and Bellahsene (2012)], Context and Inference-based alignER (CIDER) [Gracia *et al.* (2011)], DL-Learner [Bühmann *et al.* (2018)] systems use heuristic learning methods. GLUE uses probabilistic measures on each pair of source and target ontological entities and applies different machine learning techniques on them. CIDER compares each pair of terms in the entities baked on their context and later aggregates the results using an artificial neural network. YAM++ calculates different similarity metrics between the ontological entities. The similarity metrics are determined by using different machine learning methods like Decision Trees, Support Vector Machines or Naive Bayes. PARIS is also a probabilistic approach.

OntoConnect is a recent work that has addressed this issue and came up with solutions based on Recursive Neural Networks that are unsupervised. In this method, no

human involvement is required. This approach is also domain independent. Though the results obtained in this research was promising, the model developed was not tested using other popular datasets provided by the organizers of OAEI 2021 [1].

Though some of the tools described above performs well in specific cases, they have some disadvantages when we consider different scenarios. These tools addresses some particular challenges in terms of ontology matching but due to a lot of restrictions they have some disadvantages. For example, the semantic based approach PARIS [Suchanek *et al.* (2011)] is not able to handle structural heterogeneity well. If the first ontology is more precisely defined than the second one e.g. in a biological ontology countries are mentioned as the place of origin and in another such ontology, continents might be mentioned as the place of origin. Syntactic-based ontology alignment approaches like Limes [Ngomo and Auer (2011)] and COMA++ [Aumueller *et al.* (2005)] sometimes cannot calculate correct alignment since they are using string similarity based approaches. For example, different entity names which are synonyms. e.g., "Student" vs. "Pupil" and name of entities that are written in abbreviated forms, i.e., "RDF" vs "Resource Description Framework", etc. On the other hand the approaches that uses different machine learning methods, they require intensive involvement of human experts in order to obtain better labels for the datasets to be fed into the machine learning algorithms. Coming up with a solution using any one of the above approaches and performing well in diverse range of ontologies is not easy. That's why we can also see similar kinds of drawbacks in other state-of-the-art tools. In order to overcome this, many tools tried to compute the different similarities parallelly and combined them to a single similarity value. GLUE [Doan *et al.* (2003)], AgreementMaker [Faria *et al.* (2013)], YAM++ [**?**] and

[1] http://oaei.ontologymatching.org/2021/

similar other tools are using static methods of combining multiple similarities. On the other hand, some tools like RiMOM [Li *et al.* (2008)] uses dynamic strategies to calculate the weights of different similarities. Despite using dynamic strategy, it relies on a number of threshold values. To overcome this challenge, a number of tools are using different machine learning techniques. Among them CIDER [Gracia *et al.* (2011)], DL-Learner [Bühmann *et al.* (2018)], GLUE (Doan et al. 2003) are some examples that use machine learning techniques. Though they perform quite well in ontology matching, the process of coming up with proper labeled data is tedious and sometimes require help from domain experts.

In our current research, we have combined some of the above approaches to achieve high performance which does not require involvement of domain experts, in-fact, our approach can perform well in cross domain ontology matching. We have taken advantage of the syntactic based approach by using different types of string similarity measures. Some of them are character based (e.g. Levenshtein), some are sequence based (e.g. LCS), few of them are token based (Q-grams, Dice) and others are phonetic (Smith-Waterman). We have combined the structural based approach by using WordNet and by using the parent and path information of ontologies while creating features. We have also used semantic based approach by incorporating Word2vec. On top of all these similarity measures, we are using some advanced machine learning techniques which gives us models that perform well in multi-domain ontology matching. Also, the use of a huge array of similarity measures between the entities of the ontologies greatly simplifies the process of model training as well as improve the results.

Chapter 4

DATA PREPROCESSING

In this chapter we first describe the structure, nature and properties of the datasets we have used. Then we explain how we divide the dataset and preprocess them into a string-based format that can be later used for calculating the similarity measures in the feature extraction phase (Chapter 5).

4.1 Description of Datasets

We have used two popular datasets from the OAEI competition: the conference track dataset and the benchmark dataset.

Dataset#1 - Benchmark Dataset: The OAEI 'benchmark' dataset[1] consists of data sets that are built from reference ontologies of different sizes and from different domains. Most of the ontologies of this dataset are from bibliocal references from different domains. This dataset was created based on a research on generating ontology matching benchmark dataset by Jérôme Euzenat [Euzenat *et al.* (2013)]. For this research, as Dataset#1, we have chosen a set of the benchmark test library. 7 ontologies have been selected, among which ontology #101 is the reference ontology, and the remaining 6 (#102, #103, #301, #302, #303 #304) are matched against ontology #101. The idea of this partitioning has been taken from the research of Nezhadi et. al. [Nezhadi *et al.* (2011)] and Bulygin et. al. [Bulygin and Stupnikov (2019)]. In their paper, among the 6 true alignments, they have used 3 alignments for training (101-102, 101-103, 101-301) and 3 others for testing (101-302, 101-303,

[1] http://oaei.ontologymatching.org/2016/benchmarks/index.html

101-304). We have used the similar partitioning which enables us to compare our results with similar other works. The properties of this dataset are listed in Table 4.2.

Dataset#1 - Benchmark	
Ontologies	7
Classes	270
Properties	359
Alignment Files	6
Train-Test Split	3-3 (50%-50%)

Table 4.1: Properties of the Dataset#1 (Benchmark)

Dataset#2 - Conference Dataset: Conference track[2] contains 16 ontologies from the same domain (conference organization). These ontologies are suitable for ontology matching task because of their heterogenous character of origin. There are 867 classes and 724 properties. Among the 21 alignmnet files, we have taken 4 (20%) for training and 17 (80%) for testing. This split is taken similar to the other relevant works who used learning based approaches. We have kept the split similar in order to make our results comparable with the other works in the field. The properties of this dataset are listed in Table 4.2.

Dataset#2 - Conference	
Ontologies	16
Classes	867
Properties	724
Alignment Files	21
Train-Test Split	4-17 (20%-80%)

Table 4.2: Properties of the Dataset#2 (Conference)

Each of these datasets contain two types of files, the ontology files and the ontology-alignment files for each pair of the ontologies. Ontology files define the

[2]http://oaei.ontologymatching.org/2021/conference/index.html

structure of the ontology: the classes and properties along with their hierarchical relationships. Each ontology alignment file contains the information whether the classes and properties for that ontology pair align with each other or not. As an example, part of the alignment file 101-204.rdf is presented in Figure 4.1. It contains the ontology mapping between two ontologies #101 and #204. The 'match' information for each of the entities from these ontologies is given in this ontology-alignment file. We extract these information as described in the data preparation step later in this chapter (section 4.3).

dataset2 > alignments > 101-204.rdf

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rdf:RDF ... >
<Alignment>
  <xml>yes</xml>
  <level>0</level>
  <type>11</type>
  <onto1>http://oaei.ontologymatching.org/tests/101/onto.rdf</onto1>
  <onto2>http://oaei.ontologymatching.org/tests/204/onto.rdf</onto2>
  <uri1>http://oaei.ontologymatching.org/tests/101/onto.rdf</uri1>
  <uri2>http://oaei.ontologymatching.org/tests/204/onto.rdf</uri2>
  <map>
    <Cell>
      <entity1 rdf:resource="http://oaei.ontologymatching.org/tests/101/onto.rdf#type"/>
      <entity2 rdf:resource="http://oaei.ontologymatching.org/tests/204/onto.rdf#type"/>
      <measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
  <map>
    <Cell>
      <entity1 rdf:resource="http://oaei.ontologymatching.org/tests/101/onto.rdf#howPublished"/>
      <entity2 rdf:resource="http://oaei.ontologymatching.org/tests/204/onto.rdf#how_published"/>
      <measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
  .
  .
  .
</Alignment>
</rdf:RDF>
```

Figure 4.1: Sample Alignment File

4.2 Dividing Data Into Train and Test Sets

We have divided both the datasets into two parts, one part for training and and the other for testing. The training set is used for training the machine learning models and the testing set is used for testing the alignments (Chapter 6). For Dataset#1, among the 6 alignment files, we have taken 3 for training and 3 for testing. For Dataset#2, there are in total 21 alignment files. We have taken 4 of them for training, and 17 for testing. Among the test set of Dataset#2, there are some ontologies that are not present in any of the training alignments, for example, 'sigkdd' and 'confof' ontologies. Dividing the dataset in this way gives us the opportunity to prove that our models can work well for cross-domain ontology matching.

4.3 Data Preparation

In order to extract features from the ontology files, we first re-arrange the data in a format which contains pairs of classes or pairs of properties from different ontologies. For example, let's consider Ontology 1 and Ontology 2. They have many classes and properties defined in them. We use Owlready2 python library [Lamy (2017)] to extract all the classes and properties from these ontology files. Then we work on the classes and properties separately. First we take each class from Ontology 1 and pair them with each class in Ontology 2. Then based on the provided true ontology alignment information, we also append a variable 'match' which indicates whether each of those pairs is considered a match or not. The value of 'match' is extracted from the true-alignment files as shown in Figure 4.2. We do the same for all the properties defined in the ontologies as well. In other words, after getting the lists of all classes and properties, we make class pairs for each sets of classes from both the ontologies and property pairs for each sets of properties from both the ontologies. We

also include the parent and path information for each entity (class or property). A tuple is created from each of these pairs with all these information: the entity names, parent names, path and the 'match'.

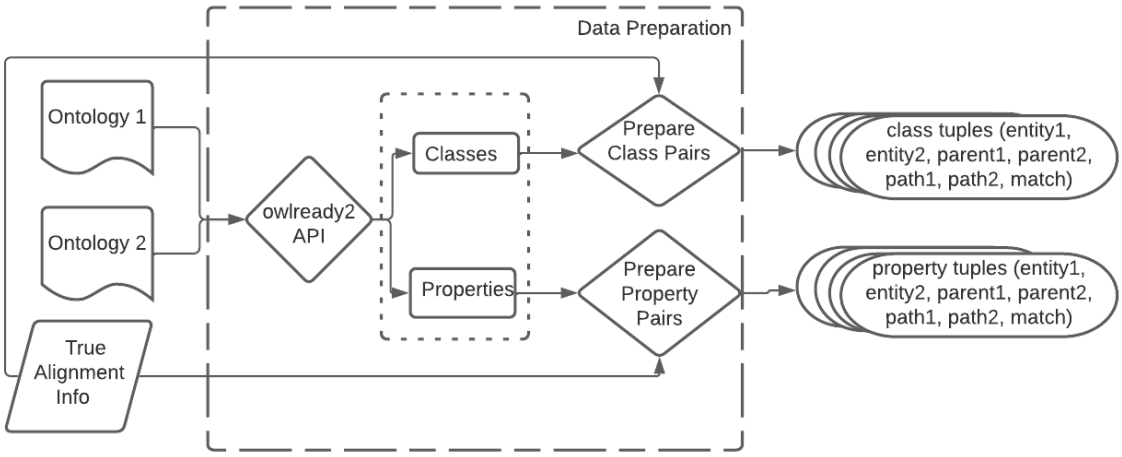


Figure 4.2: Overview of the Data Preparation Step

These tuples are then saved in a CSV file for the feature extraction step. A sample CSV file is given in Table 4.3. Some fields (i.e. Parent2, Path1, Path2) are omitted from the table for brevity. During the feature extracton step, all the features are calculated based on similarity measures for the entities, parents and paths.

Ontology1	Ontology2	Entity1	Entity2	Parent1	...	Match	Type
101.rdf	304.rdf	Report	PageRange	Reference	...	0	Class
101.rdf	304.rdf	Report	Journal	Reference	...	0	Class

Table 4.3: Dataset#1 CSV File - Sample Contents after Data Preparation

Algorithm 5 defines the steps how datasets are created from ontology files. The overall data preparation step is illustrated in figure 4.2. After data preprocessing step, we get a single CSV file for each of the datasets which contains comma separated string values as shown in Table 4.3. In order to send them into the machine learning model, we need to convert them into numerical values which is described in the next chapter.

Chapter 5

FEATURE EXTRACTION

The first important task for ontology alignment is to get a feasible format of the ontology data so that it can be compared with each other in a machine readable way. We also need a format that is ready to be fed as the inputs of machine learning or neural network models. In this research we have explored different ways to get a vecrorized form of the ontologies so that they can be fed into the machine learning models.

5.1 Similarity Measures

After the data preprocessing step (Chapter 4) we got tuples of strings for each pair of classes and properties from different ontologies. Now, in order to convert them into meaningful numbers, we have used different similarity measures. Each of these similarity measures gives us a similarity score between 0 and 1. All these similarity scores can then be used together to prepare the features for the machine learning models. The similarity measures are described in this section.

5.1.1 String Based Similarity

We have used following string based similarity measures in this work.

N-gram consider similarity of substrings and it is efficient when some characters are missing [Euzenat and Shvaiko (2007)]. For example, when N=2, the words are divided into 2 character sub-strings and then the number of match between each list is counted.

$$similarity(s1, s2) = \frac{2[pairs(s1) \cap pairs(s2)]}{pairs(s1) + pairs(s2)}$$

Dice coefficient is defined as twice the number of common words of compared strings over the total number of words in both strings [Cohen and Fienberg (2007)]. This is calculated in the following way:

$$d = \frac{2n_t}{(n_x + n_y)}$$

where n_t is the number of character bigrams found in both strings, n_x is the number of bigrams in string x and n_y is the number of bigrams in string y.

Jaccard and **Generalized Jaccard** similarity are defined as the size of the intersection divided by the size of the union of the sample sets of words [Stoilos and Kollias (2005a)]. Jaccard similarity measure for two sets A and B can be expressed using the following equation:

$$J(A, B) = \frac{A \cap B}{A \cup B}$$

If two datasets share the exact same members, their Jaccard Similarity Index will be 1. Conversely, if they have no members in common then their similarity will be 0.

Levenshtein distance between two strings is the minimum number of single-character edits required to change one word into the other [Euzenat (2004)]. Levenshtein distance can be measured using the following formula:

$$L = \begin{cases} |a|, & \text{if } |b| = 0 \\ |b|, & \text{if } |a| = 0 \\ lev(tail(a), tail(b)), & \text{if } a[0] = b[0] \\ 1 + min \begin{cases} lev(tail(a), b) \\ lev(a, tail(b)) \\ lev(tail(a), tail(b)) \end{cases} & \text{otherwise} \end{cases}$$

where the $tail()$ of some string x is a string of all but the first character of x, and $x[n]$ is the nth character of the string x, counting from 0.

Jaro and **Jaro-Winkler** measures is edit distance measure designed for short strings [David (2007)]. Jaro similarity can be measured using the following equation:

$$J = \begin{cases} 0, & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s1|} + \frac{m}{s2} + \frac{m-t}{m}\right), & \text{for } m \neq 0 \end{cases}$$

where,

- m is the number of matching characters
- t is half the number of transpositions
- where $|s1|$ and $|s2|$ are the lengths of strings $s1$ and $s2$ respectively.

Jaro-winkler similarity can be measured using the following equation:

$$Sw = Sj + P * L * (1–Sj)$$

where,

- Sj, is jaro similarity
- Sw, is jaro- winkler similarity
- P is the scaling factor
- L is the length of the matching prefix.

Monge-Elkan is a type of hybrid similarity measure that combines the benefits of sequence-based and set-based methods [Straccia and Troncy (2005)].The algorithm

uses similarity function (Example : Jaro-Winkler or Levenshtein score) as inner function. The inner function is used to compute the scores of the best matching token.

$$\text{MongeElkanSimilarity} = \frac{1}{|x|}\sum_{i=1}^{|x|} max_{j=1,|y|} sim'(x_i, y_j)$$

The **Smith-Waterman measure** determine similar regions between two strings [Cohen and Fienberg (2003)]. Instead of looking at an entire sequence at once, the S-W algorithm compares multi-lengthed segments, looking for whichever segment maximizes the scoring measure. The algorithm itself is recursive in nature. First we need to determine the substitution matrix and the gap penalty scheme

- $s(a, b)$ - Similarity score of the elements that constituted the two sequences
- W_k - The penalty of a gap that has length k

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + s(a_i, b_j) \\ H_{i-k,j} - W_k \\ H_{i,j-1} - W_1 \\ 0 \end{cases}$$

The **Needleman-Wunsh distance** is computed by assigning a score to each alignment between the two input strings and choosing the score of the best alignment [Needleman (1970)].

The **Affine gap** distance is an extension of the Needleman-Wunsch measure that handles the longer gaps more gracefully [Doan (2012)].

The **Bag distance** is edit distance for sets of words [52]. Cosine similarity transforms a string into vector so Euclidean cosine rule is used to determine similarity

[Stoilos and Kollias (2005b)].

Fuzzy Wuzzy Partial Ratio finds the similarity measure between the shorter string and every substring of length m of the longer string, and returns the maximum of those similarity measures [Rao and Prasad Reddy (2018)].

Soft TF-IDF and **TF-IDF** are numerical statistics that are intended to reflect how important a word is to a document in a collection or corpus [Needleman (1970)].

Partial Token Sort and **Token Sort** are obtained by splitting the two strings into tokens and then sorting the tokens. The score is the fuzzy wuzzy partial ratio raw score of the transformed strings.

Fuzzy Wuzzy Ratio is the ratio of the number of matching characters to the total number of characters of two strings [Rao and Prasad Reddy (2018)].

Editex [Zobel and Dart (1996)] and **Soundex3** are phonetic matching measures.

Tversky index is an asymmetric similarity measure on sets that compares a variant to a prototype [Tversky (1977)].

Overlap coefficient is defined as the size of the intersection divided by the smaller of the size of the two sets [Vijaymeena and Kavitha (2016)].

5.1.2 Language Based Similarity

It is possible that words differ but are close in meaning, e.g. "car" and "auto". WordNet can solve this problem. Wu and Palmer similarity are used for handling this scenario [Wu (1994)]. If the strings consist of several words then the maximum similarity measure of all possible pairs of sets of words is taken. But the weakness of WordNet is that it contains only a part of all words of the language. Usage of vector representations of words from Word2vec models [Mikolov *et al.* (2013)] facilitates this problem. In order to get word2vec similarity, we have tried all the pre-trained models

and corpora from the Gensim Data Project[1]. Among these the 'word2vec-google-news-300' has given us the best result. This dataset is based on a database containing about 100 billion words and contains 3000000 vector representations. Using these, we calculated the cosine similarity between two vector representations of words. If the strings consists of several words then Sentence2vec algorithm from [Zhang *et al.* (2014)] is used. The process of calculating word2vector similarity is explained in Algorithm 1. This algorithm takes two rowSets taken from the data preparation step (section 4.3) and then using a pre-trained word2vector *model* (using the 'word2vec-google-news-300' dataset), it calculates the maximum similarity for a word from the first rowSet with all other words from the second rowSet. Finally it returns the average of all these maximum similarity scores.

Algorithm 1: Word2VectorSimilarity() - Calculate Word2Vector Similarity

Data: $rowSet1$, $rowSet2$, $model$ - two rows of words and the word2vector model to use
Result: $similarityScore$ - similarity score for $rowSet1$ and $rowSet2$

$score \leftarrow 0$;
$N = max(len(rowSet1), len(rowSet2))$;
/* Loop through all pairs of words from each rowSet */
for $word1 \in rowSet1$ **do**
    $maxSimilarity \leftarrow 0$;
    for $word2 \in rowSet2$ **do**
        /* Calculate similarity score using the given model */
        $similarity \leftarrow model.wv.similarity(word1, word2)$;
        if $maxSimilarity < similarity$ **then**
            $maxSimilarity \leftarrow similarity$;
        else
    $score \leftarrow score + maxSimilarity$;
$score = score/N$;
return $score$;

[1]https://github.com/RaRe-Technologies/gensim-data

5.1.3 Meta-data and Structure Based Similarity

It's not possible to get a good ontology matcher without considering the meta-data of the ontologies. Metadata contains the properties of the entities defined in the ontologies, the relationships between different entities (classes). These information is vital for identifying the similarity between two ontologies.

In order to address that issue, in this work we take the similarity measures of the parents of each entity and add them with the original features. We also take the path of each entities and get the similarity measures for them. The assumption behind this is the parents and path for similar entities will have similar information. By taking the similarity measures for the parents and paths of each entity we are ensuring that we don't lose the hierarchical information of the entities.

5.2 Using the Similarity Measures as Features

Based on the above three types of similarity measures, the feature vectors are prepared for each ontology. For each pair of entities, the feature vector contains numerical values in the range $[0, 1]$. The size of the feature vector is same for each pair because we are applying the same number of similarity measures for a particular model. While matching, we only match classes against classes and properties against properties. We are not considering the instances of the ontologies. The overall process of feature extraction is illustrated in figure 5.1

The 'Classification Model' in Figure 5.1 represents the machine learning process which we describe in chapter 6. In general, the output from the feature extraction will be the feature vector along with the 'match' information. While training the machine learning models, the features are sent as input and we get a model after training. During the alignment prediction step, again these features are sent as input into the

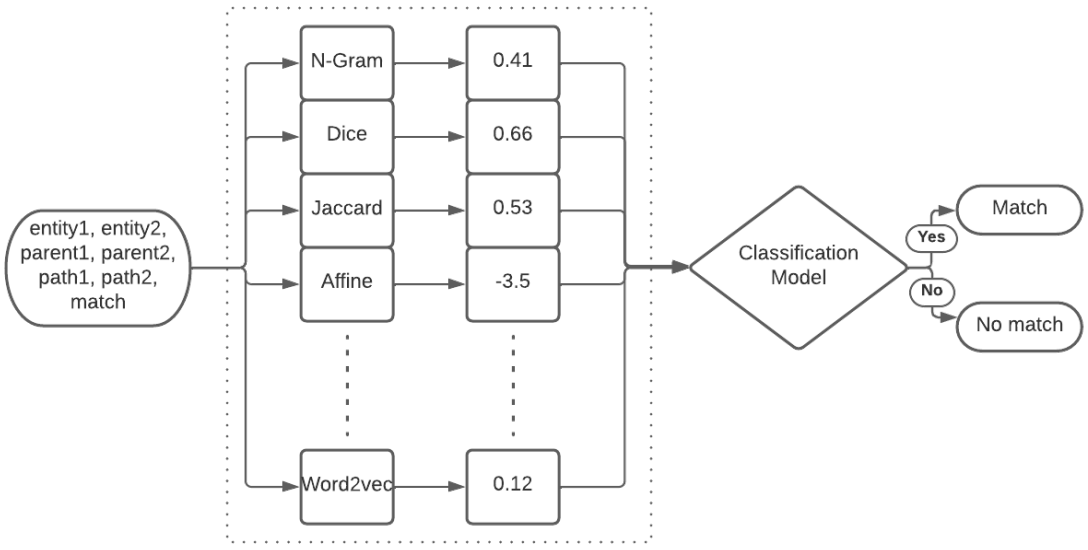


Figure 5.1: Feature Extraction Based on Similarity Measures

model and a prediction is found as a result. The prediction is then compared with the true alignment to decide it's correctness. Precision, recall and finally f-measure is calculated based on the predicted number of matches and the true number of matches.

5.3 Normalizing the Features

Normalization is a standard procedure for dealing with any data that needs to be sent into machine learning models. It ensures that the features are balanced for all the domains. Normally four common normalization techniques are useful:

- Scaling to a range
- Clipping
- Log scaling
- Z-score

In our case, the data that needs to be fed into mathine learning models are the similarity scores found from the methods of py_stringmatching library. When we

calculated string similarity measures the method $get\_raw\_score()$ of that library takes care of fitting the value in the range of $[0, 1]$. So, we already get the data scaled to a specific range.

Now that the features are all ready and we have prepared the numerical feature vectors, we need to design some machine learning models that can help recognize a match among the entity pairs when those feature vectors are passed as input. In the next section, we have discussed definitions of different machine learning models, why we have selected some of those and how they have performed in our experiments.

Chapter 6

MACHINE LEARNING FOR ONTOLOGY ALIGNMENT

In this section first we are going to discuss how we have selected certain machine learning techniques and how they have performed in aligning entities from different ontologies.

6.1 Why Machine Learning?

First question is, why do we need machine learning to solve this problem? In section 1.2 we have identified several challenges of ontology alignment problem. Because of the huge amount of semantic web data, heterogeneity of the data, lack of interoperability among multiple organizations and the time-consuming process of matching ontologies are primary reason why it is difficult to solve the ontology alignment problem. The rule-based approaches might be able to define specific rules for a specific set of ontologies, but the use of that in other areas is difficult and erroneous. On the other hand, if we can train a machine learning model based on sufficient amount of data, it will be able to identify alignments among ontologies from different origins.

6.2 Supervised vs Unsupervised

There are primarily two types of machine learning techniques, supervised and unsupervised. **Supervised learning** is a machine learning approach that takes the advantage of already labeled data sets. These data sets are designed to train or "supervise" algorithms into classifying data or predicting outcomes accurately. Using labeled inputs and outputs, the model can measure its accuracy and learn over time. On the other hand **unsupervised learning** uses machine learning algorithms to

analyze and cluster unlabeled data sets. Without the need for human supervision or intervention, these algorithms are able to discover hidden patterns in data (hence, they are "unsupervised").

In our case, we do have labeled data, i.e. whether two entities 'match' or not. Here, 'match' is the label which takes values '0' (not match) or '1' (match). The entity-pairs from different ontologies (class or property) are the inputs for which the labels ('match') are provided by human experts. The availability of labeled data has enabled us to use supervised machine learning techniques.

6.3 Regression vs Classification

Now, for supervised techniques, there are two types of methods: regression and classification. Regression is normally used in the case when the label or prediction has values of continuous nature. Classification is used when the labels are categorical or discrete in nature. For example, if we want to predict the percentage of rain on a particular day based on weather data from past few months, we need a regression problem because percentage is a continuous output. But, if we ask whether it will rain or not on a certain day, that will become a classification problem (a binary classification problem in particular).

For our ontology data, we want to find out, given two entities (class or property) from different ontologies, whether the entities identify the same thing or not; in other words, whether the entities match with each other or not. It's a binary classification problem. So, we have chosen to apply some classification based machine learning techniques in our experiments.

6.4 Classification Algorithms

Different classification algorithms are taken into account for all types of analysis performed here (Sousa *et al.*, 2015; Liang *et al.*, 2011). Descriptions of all these methods are given in the following section and the abbreviations listed in this table are referred in the results tables at the end of chapter 7.

Abbreviation	**Algorithm Name**
LR	Logistic Regression Classifier
RF	Random Forrest Classifier
ADA	Ada Boost Classifier
DT	Decision Tree Classifier
SVM	Support Vector Machine Classifier
LDA	Linear Discriminant Analysis
SGD	Stochastic Gradient Descent Classifier
KNN	K-Nearest Neighbor Classifier
MLP	Multi Layer Perceptron
NB	Gaussian Naive Bayes Classifier

Table 6.1: Machine Learning Algorithms Used in the Experiment

6.4.1 Logistic Regression Classifier

Logistic regression is a classification algorithm, used when the value of the target variable is categorical in nature. Logistic regression is most commonly used when the data in question has binary output, so when it belongs to one class or another, or is either a 0 or 1. Regression models usually produce output in a continuous range, but we need binary classification, either 0 or 1. In order to achieve this, logistic regression uses a special function called 'sigmoid function' which resembles an 'S' shaped curve when plotted on the graph and squishes the values towards the end both ends of Y axis. The sigmoid function is defined as $y = 1/(1 + e^{-x})$. The value of y becomes close to y 0 if x is a large negative value and close to 1 if x is a large positive value.

Analysis from our experiments: In our work, we have used sklearn's[1] LogisticRegression class with the following parameters:

- penalty='l2', this is used to prevent the function from overfitting problem.
- C=1.5, represents the inverse of regularization strength, the higher the value of C, the less the regularization. We can see the effect of this in Figure 8.10a.
- class_weight=None, means all classes have equal weight, i.e. 1
- max_iter=100, maximum number of iterations is 100.

6.4.2 Decision Tree

Decision Trees are designed with the following goals in mind: 1) accurately classify most of the training samples; 2) be adaptive enough to be equally or at least comparably good when working on the test dataset; 3) be easy to update for new incoming training data; 4) and boast a pleasingly simple structure. While designing a DT the following are key points that are taken into account: the choice of the structure, the selection of feature subsets for each internal node and the choice of the strategy to be devised for each node. When the DT design is approached from a Bayesian perspective, the following optimization problem needs to be solved subject to limited training data:

$$\min_{T,F,d} p_e(T, F, d) \tag{6.1}$$

where p_e, is the total probability of error, T represents the selected tree structure and F and d are the feature subsets and strategies respectively which are to be used at the internal nodes. This optimization problem can be solved in two steps:

[1] https://scikit-learn.org/

Step 1 For a given T and F search for d^* as follows:

$$d^* = d^*(T, F) \text{ such that } p_e(T, F, d^*(T, F)) = \min_d P_e(T, F, d) \tag{6.2}$$

Step 2 Find T^* and F^* such that:

$$P_e(T^*, F^*, d^*(T^*, F^*)) = \min_{T,F} p_e(T, F, d^*(T, F)) \tag{6.3}$$

Analysis from our experiments: We have used decision trees of different depth and finalized with a depth of 2. In Figure 8.8a we can see that with the increase of depth, the performance of the ontology alignment method decreases. The reason behind this is, as we go deeper into the decision tree, the feature vector are broken into smaller values which individually does not carry much information to identify a particular class.

6.4.3 Stochastic Gradient Descent Classifier

Gradient descent is an iterative optimization technique wherein the solution at each step is improved by taking a step along the negative of the gradient of the function to be minimized at the current point. SGD allows the optimization procedure to take a step along a random direction, as long as the expected value of the direction is the negative of the gradient. Here, the direction need not be updated to be based exactly on the gradient. Instead, the direction is set to be a random vector and its expected value at each iteration equals the gradient direction. In other words, the expected value of the random vector will be a subgradient of the function at the current vector. SGD is a simple and an efficient approach to discriminative learning of linear classifiers under convex loss functions. SGD-Classifier is a Linear classifier

with SGD training. The linear classifier to be used is determined based on the hypter parameter loss. So, if loss='hinge' is used, it is an implementation of Linear SVM and if we take loss='log' it is an implementation of Logisitic regression.

Analysis from our experiments: We have taken loss='log', i.e. implementing SGD classifier for Logistic Regression. e could not use the 'hinge' loss, which uses linear SVM, because it does not give the probabilistic prediction that we need. For the sk-learn SGDClasiier, by default the maximum number of passes (i.e. epochs) is set to 1000. In our case after experimenting, we have found that 700 is the optimal solution for both the datasets we used. In our ablation studies, in Figure 8.10b, it shows how the number of iterations affects the F-measure for Dataset#2.

6.4.4 Support Vector Classifier

SVMs are binary classifiers and their primary goal is to locate a separating hyperplane in the space between the two classes by mapping the data into a higher dimensional space.

Let there be a dataset constituting l patterns where each l happens to be a pair of the type $(x_i, y_i) \forall i \in [1, ..., l], x_i \in \mathbb{R}^m$, and $y_i = \pm 1$. A standard binary SVM is formulated by minimizing a Convex Constrained Quadratic Programming (CCQP) objective function as follows:

$$\min_{\alpha} \tfrac{1}{2}\alpha^{\mathbf{T}}\mathbf{Q}\alpha - \mathbf{r}^{\mathbf{T}}\alpha \tag{6.4}$$

$$0 \leq \alpha_i \leq C \forall i \in [1, \ldots, l], \tag{6.5}$$

$$\mathbf{y}^{\mathbf{T}}\alpha = \mathbf{0}, \tag{6.6}$$

where C is the regularization parameter or penalty term which controls how well the SVM would perform on unseen test data, $r_i = 1 \forall i$ and Q is the symmetric positive semidefinite $l \times l$ kernel matrix where $q_{i,j} = y_i y_j K(x_i, x_j)$.

The solution of the CCQP function affords us with the $\alpha_i \forall i \in [1, \ldots, l]$ values which are plugged into the Feed-Forward Phase (FFP) of the SVM as follows in order to do class prediction:

$$f(x) = \sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b \tag{6.7}$$

where b is the bias term and is generally computed based on the support vectors that lie in the margins.

Analysis from our experiments: We have tried to use different kernels, i.e. 'linear', 'polynomial', Radial Basis Function (RBF) and 'sigmoid'. For natural language and text based datasets, the features created by non-linear kernels (polynomial, RBF and sigmoid) can become very high. That's why using the 'linear' kernel gave us the optimal results since the non-linear kernels can easily handle thousands of features without overfitting.

We have done some experiments on the regularization parameter, C. In general C instructs how strictly SVM will behave in defining the boundary between the two classes (for binary classification). The larger the value of C, the more strict it becomes, i.e. tries to find a narrow boundary, but also it increases the chance of overfitting. The less the value of C, it allows more mistakes, but the chance of overfitting is less. We have chosen a small value of C (0.025) after doing some experiments that gave us optimal results.

6.4.5 Random Forest Classifier

Random Forest (RF) operates by adding multiple decision trees using and using the Bootstrap Aggregation Method. It is a highly efficient ML algorithm for predictive analysis. The random forest model is very suitable at handling tabular data with numerical features. Unlike linear models, random forests are able to capture non-linear interaction between the features and the target. However, the tree-based models are not designed to work with very sparse features. When dealing with sparse input data, the sparse features need to either be pre-processed to generate numerical statistics or in such cases, a linear model should be incorporated.

Though random forests are build from decision trees, there are differences. In general, random forest algorithm randomly selects observations and features to build several decision trees and then takes an average of the results. When the depth of decision trees is increased, it causes overfitting problem. But Most of the time, random forest prevents this by creating random subsets of the features and building smaller trees using those subsets. Afterwards, it combines the subtrees. It's important to note this doesn't work every time and it also makes the computation slower, depending on how many trees the random forest builds.

Analysis from our experiments: We have used different depth for the random forest algorithm on Dataset#2 and found that it does not cause much change in the result. This is because, the predictions does not emphasize much on a single level of the decision trees, rather takes average of the results found in different levels. So, in a large sample, like the one in our dataset, the final result does not vary too much based on the depth taken. This is also true for the 'number of estimators', i.e. the number of decision trees to create from the features. The results of changing these parameters are shown in graphical form in Figure 8.8b and Figure 8.9 respectively.

6.4.6 Linear Discriminent Analysis

Linear Discriminant Analysis (LDA) is based on the originally proposed discriminant analysis in an article by Sir Ronald Fisher (Fisher (1936)). LDA is the process of finding a set of linear combination of features that characterizes two or more entities. This combination can be used to classify the entities. Given a data pool X with higher dimension, LDA reduces them to a set of lower dimension components each of which represents a specific class in the original data pool. Each class x in the dataset can be assigned to a particular class j based on two primary rules:

- Maximum likelihood rule: If the probability of each class to occur is equal, then x would be classified as j if
  $j = argmax_i f_i(x)$
- Bayesian Rule: If the prior probability of x to be in class j is π, then
  $j = argmax_i \pi_i f_i(x)$

LDA vs PCA: There is similarity between Principal Component Analysis (PCA) and LDA because both can be used for dimensionality reduction. LDA deliberately focuses on modeling the difference among the classes. On the other hand, PCA builds the feature combinations based on differences in the data rather than similarities and ignores the differences in classes. In our experiments, LDA has been proved quite useful because of this nature. Since LDA focuses on separating the features for each class, it can identify the features that correspond to each entity from different ontologies. If the entities are same or similar, the features would tend to be same.

Analysis from our experiments: There are different types of solvers that can be used for LDA: singular value decomposition (SVD), least squares solution (lsqr) and eigenvalue decomposition (eigen). SVD does not compute the covariance matrix,

therefore this solver is recommended for data with a large number of features. In our case, we are using a large number of features extracted from the similarity measures of the ontology entities. That's why we have used the SVD solver in our experiment. For 'eigen' solver, we have tried to find a good 'shrinkage' value. It appears that the less the value (i.e. close to 0), the better the results are. This experiment is displayed in Figure 8.12.

6.4.7 K Nearest Neighbor Classifier

K-Nearest Neighbor classifier works by identifying the class which is most common among it's k closest neighbor based on a selected distance measure. Choosing the right value of k is important for KNN. It is important to search for the optimal value of k for a particular dataset.

Analysis from our experiments: In our experiment for ontology matching, we have searched for the right value of k for each of the datasets separately. $K = 11$ gives us the best result for Dataset#1. $K = 6$ gives us the best result for Dataset#2.

6.4.8 Multi-Layer Perceptron

Multi layer perceptron is a basic step towards artificial neural network, which eventually was inspired by the complex connections of human brain (Callan (1998)). Multi layer perceptron was designed based on the observation that the connection between neurons (the synapses) in human brain can encode necessary information to make intelligent decisions. During the late 1940s, the McCulloch-Pitts model proposed a preliminary model for a neurone used in Perceptrons and MLPs. A neuron

is modelled based on a threshold function.

$$f(x) = \begin{cases} 1, & \text{if } x \geq 1 \\ 0, & \text{otherwise} \end{cases}$$

Now, in a multilayer perceptron, there are three layers of nodes, input layer, hidden layer(s) and the output layer. Each node works as a neuron and with them a non-linear activation function is associated. Also a set of weights w_{ij} are associated with each node that connects to each of the nodes in the next layer. Based on the previous iteration, and observing the change in the calculated error, the weights are adjusted in each node. This process continues until no change is observed in the error.

Analysis from our experiments: As an initial step towards solving ontology alignment problem using artificial neural networks, we have developed an MLP classifier model and evaluated on the two datasets. Number of hidden layers (4, 2) and (6, 4) worked best for Dataset#1 and Dataset#2 respectively. We have also tested using different number of random states for MLP classifier. But, we could not find any pattern to help us define the number of random states to use.

6.4.9 Ada Boost Classifier

Ada Boost classifier is an ensemble method that uses 'boosting' to improve the model iteratively by learning the mistakes done in previous model. The name comes from 'Adaptive Boosting'. Normally it combines multiple weak classifiers into a single robust classifier. There are couple of important parameters that needs to be tuned for AdaBoost Classifier.

- n_estimators: Number of weak learners to train in each iteration.
- learning_rate: It contributes to the weights of weak learners. It uses 1 as a

default value.

Analysis from our experiments: In our experiments, we have primarily tested with different number of estimators, i.e. the number of weak classifiers to use. In both cases, lower number of estimators performed well. The results of ablation studies are illustrated in Figure 8.11a and Figure 8.11b.

6.4.10 Other Machine Learning Methods

We have tried to use some other machine learning methods that didn't perform quite well. For example, Gaussian Process Classifier, Gaussian Naive Bayes and Quadratic Discriminant Analysis (QDA). All of these methods performed poorly for our setup. Ada Boost Classifier gave us quite good result for Dataset#1 (average F-Measure 0.81).

6.4.11 Summary

In order to create a classification model that can find match between two entities from different ontologies, we have created some models using some popular machine learning methods. Some of these models, e.g. Random Forest, Ada Boost, Logistic Regression, Linear Discriminant Analysis, SVM etc. have given us results that are comparable to some of the most prominent tools developed for ontology alignment. We have discussed the definition of each machine learning technique in this chapter and also provided some insight why we have used certain parameters of these techniques. In the next chapter we discuss in detail how the features extracted in Chapter 5 are used to train these models.

Chapter 7

ONTOLOGY MATCHING

7.1 Using Machine Learning

For ontology alignment with machine learning, in our method OntoAlign, we define an alignment with the tuple (entity1, entity2, align) where the first two are the entities that we are matching and 'align' is the percentage of match between the entities. These tuples can be either obtained manually by domain experts (the true alignments) or from the output of a machine learning model (the predicted alignment).

We have designed the machine learning models that can predict the ontology alignments and gives as output the predicted alignments. The inputs to the models are the vectorized forms of the entities. The input vectors are obtained by taking different similarity measures.

7.1.1 Alignment Creation Using Similarity Measures

The first step of preparing the dataset for input is to extract the classes and properties from the ontologies. We use Owlready2 python API (Lamy (2017)) in order to obtain the classes and properties from the ontologies. Then for each class and properties we get the values for all the similarity measures described in section 5.1. Then, each class from the first ontology is matched with each class from the second ontology. For example, if in the first ontology includes 20 classes and in the second ontology includes 12 classes, then 240 pairs are matched. Each pair is fed to the input

of a machine learning model, which calculates the probability (confidence) of matching for each pair. Then the threshold is set: if the probability is above the threshold, then the pair is added to the final alignment. Similar procedure is performed on the properties of the ontologies. This process is described in Algorithm 2 and Algorithm 3.

Algorithm 2: ontoMatch() - Match two ontologies

Data: $ontology1$, $ontology2$
Result: $finalAlignment$ - alignment for $ontology1$ and $ontology2$

```
/* Getting the classes for each ontology                                    */
classes1 ← getClasses(ontology1);
classes2 ← getClasses(ontology2);
/* Create alignmnets for the classes using Algorithm 2                      */
classAlignments ← createAlignment(classes1, classes2);
/* Getting the properties for each ontology                                 */
properties1 ← getProperties(ontology1);
properties2 ← getProperties(ontology2);
/* Create alignmnets for the properties using Algorithm 2                   */
propertyAlignments ← createAlignment(properties1, properties2);
finalAlignment ← classAlignments ∪ propertyAlignment;
return finalAlignment;
```

Algorithm 3: createAlignment() - Predict alignment for two lists of entities

Data: $entities1$, $entities2$ - lists of entities, either classes, or properties, $threshold$
Result: $alignment$ - alignment for $entities1$ and $entities2$

```
/* Loop through all pairs of entities                                       */
for entity1 ∈ entities1 do
    for entity2 ∈ entities2 do
        /* Calculate similarity measures for each entity pair               */
        simMeasures ← calculateAllSimMeasures(entity1, entity2);
        match ← predictMatch(simMeasures);
        if match > threshold then
            alignment ← alignment ∪ (entity1, entity2)
        else
return alignment;
```

Name, parent name, and the full hierarchical path are retrieved from each class. The parent of a class is its super class. The full path is a string that describe the entire hierarchy of classes: from the most general class to the current class. For example, the class "Book" has the name "Book", the parent name "Publication" and the full path "Thing/Publication/Book". Thus, a list of pairs for matching is generated. For properties, the parent is the class that it describes. And the full path is a string describing the complete hierarchy up to the class that describes the property. For each pair, all similarity measures listed in section 5.1 are calculated. Then all similarity measures are combined into a list. This process is described in Algorithm 4.

Algorithm 4: calculateAllSimMeasures() - Calculate similarity Measures for given pair of Entities

Data: $entity1$, $entity2$
Result: $finalAlignment$ - alignment for $entity1$ and $entity2$

```
1  name1 ← getName(entity1) ;          /* get name for entity1 */
2  name2 ← getName(entity2) ;          /* get name for entity2 */
3  parent1 ← getParent(entity1) ;      /* get parent for entity1 */
4  parent2 ← getParent(entity2) ;      /* get parent for entity2 */
5  path1 ← getPath(entity1) ;          /* get path for entity1 */
6  path2 ← getPath(entity2) ;          /* get path for entity1 */
7  nameSimilarityMeasures ← calculateSimMeasures(name1, name2);
8  parentSimilarityMeasures ← calculateSimMeasures(parent1, parent2);
9  pathSimilarityMeasures ← calculateSimMeasures(path1, path2);
10 simMeasures ←
     nameSimMeasures ∪ parentSimMeasures ∪ pathSimMeasures;
11 return simMeasures;
```

7.1.2 Creating dataset and training a machine learning model

For this part, the input data is a list of ontology pairs and the true alignment between them. A model from the list of machine learning models (see Table 6.1) is used and appropriate parameters are also selected. The process is similar to the

Algorithm 5: createDatasetAndTrainModel() - Creates dataset and then train the machine learning models

Data: $trainPairsOntologies$ - tuple containing ($ontology1$, $ontology2$ and $trueAlignment$);
$modelName$ - the machine learning model to use;
$modelParams$ - parameters for the ML model
Result: $model$ - trained model

```
/* Loop through all given tuples and extract the classes and properties   */
for ontology1, ontology2, trueAlignment ∈ trainPairsOntologies do
    classes1 ← getClasses(ontology1);
    classes2 ← getClasses(ontology2);
    trainDatasetClasses ←
      createDataset(classes1, classes2, trueAlignment, 'Class');
    properties1 ← getProperties(ontology1);
    properties2 ← getProperties(ontology2);
    trainDatasetProperties ←
      createDataset(properties1, properties2, trueAlignment, 'Property');
    /* Merge the training datasets for classes and properties             */
    trainDataset ←
      trainDataset ∪ trainDatasetClasses ∪ trainDatasetProperties;
model ← trainModel(trainDataset, modelName, modelParams);
return model;
```

previous step: the names of objects, the names of parents and full paths are retrieved, and the similarity measures are calculated. First, a dataset is created for the classes, then for the properties, and after that both the datasets are combined.

A true alignment, two lists of entities and the type of input entities are provided as the input. Each entity from the first list is mapped to each entity from the second list. Then, if a pair of entities is contained in the true alignment, then the pair is assigned label "1", otherwise - label "0". Also, each pair indicates the type of entity (either "Class" or "Property") because the same model was used to map classes and properties. Then all pairs are combined into one dataset. Further, the model is trained on the created dataset with the selected parameters. The process is described in Algorithm 5 and Algorithm 6.

Algorithm 6: createDataset() - Create dataset given two lists of entities

Data: $entities1$, $entities2$ - lists of entities, either classes, or properties, $threshold$, $trueAlignment$, $entityType$

Result: $trainingDataset$ - tuple containing $entity1$ $entity2$, $match$, $entityType$, $simMeasures$

```
/* Loop through all pairs of entities                                    */
for entity1 ∈ entities1 do
    for entity1 ∈ entities1 do
        /* Calculate similarity measures for each entity pair            */
        simMeasures ← calculateAllSimMeasures(entity1, entity2);
        if (entity1, entity2) ∈ trueAlignment then
            trainDataset ←
              trainDataset ∪ (entity1, entity2, 1, entityType, simMeasures)
        else
            trainDataset ←
              trainDataset ∪ (entity1, entity2, 0, entityType, simMeasures)X
return trainDataset;
```

7.2 Results

The results from using the above discussed machine learning techniques on different datasets are discussed in this section. In order to measure the similarity measure word2vec, we have used dataset 'GoogleNews-vectors-negative300' for the results in table 7.1 and table 7.2.

Alignments	**LR**	**RF**	**ADA**	**DT**	**SVM**	**LDA**	**SGD**	**KNN**	**MLP**
101-302.rdf	0.73	0.73	0.72	0.7	0.73	0.67	0.68	0.38	0.28
101-303.rdf	0.83	0.83	0.80	0.73	0.82	0.65	0.71	0.61	0.77
101-304.rdf	0.9	0.92	0.90	0.88	0.89	0.82	0.75	0.83	0.86
Average	0.82	0.83	0.81	0.77	0.81	0.71	0.71	0.61	0.63

Table 7.1: F-Measures from Different Machine Learning Algorithms on Dataset-1. Word2Vec Dataset: GoogleNews-vectors-negative300

7.2.1 Results for the OAEI 'Benchmark' Dataset

We have used the OAEI 2016 'benchmark' dataset as the "Dataset#1" for our experiments.

From the results in 7.1, we can see that Random Forrest, Logistic Regression Classifier, SVM Classifier have the best f-measures, while Decision Tree Classifier, KNN Classifier, Linear Discriminant Analysis have slightly lower performance for dataset1.

Alignments	**LR**	**RF**	**ADA**	**DT**	**SVM**	**LDA**	**SGD**	**KNN**	**MLP**
conf-edas	0.50	0.56	0.56	0.44	0.56	0.62	0.4	0.36	0.50
cmt-sigkdd	0.63	0.74	0.74	0.8	0.67	0.73	0.52	0.73	0.67
edas-sigkdd	0.61	0.64	0.64	0.64	0.64	0.64	0.61	0.5	0.56
ekaw-sigkdd	0.76	0.78	0.78	0.71	0.78	0.78	0.70	0.71	0.78
cmt-edas	0.73	0.7	0.76	0.57	0.76	0.73	0.7	0.50	0.64
conf-sigkdd	0.56	0.64	0.67	0.48	0.67	0.62	0.45	0.38	0.67
confof-edas	0.58	0.54	0.50	0.56	0.54	0.58	0.53	0.51	0.6
confof-iasted	0.62	0.62	0.62	0.62	0.62	0.62	0.57	0.53	0.71
conf-confof	0.64	0.61	0.61	0.45	0.61	0.72	0.41	0.32	0.61
cmt-confof	0.38	0.41	0.36	0.29	0.36	0.36	0.38	0.38	0.38
conf-ekaw	0.44	0.47	0.47	0.32	0.44	0.41	0.33	0.27	0.49
cmt-ekaw	0.62	0.62	0.58	0.53	0.59	0.67	0.43	0.53	0.59
confof-ekaw	0.67	0.61	0.60	0.79	0.48	0.61	0.67	0.53	0.67
iasted-sigkdd	0.69	0.81	0.76	0.77	0.8	0.71	0.58	0.67	0.75
cmt-iasted	1.00	0.89	0.89	0.89	0.89	0.89	0.43	0.67	0.89
edas-iasted	0.33	0.52	0.46	0.48	0.52	0.52	0.32	0.44	0.44
ekaw-iasted	0.71	0.75	0.75	0.57	0.75	0.75	0.57	0.44	0.75
confof-sigkdd	0.73	0.73	0.73	0.67	0.73	0.73	0.73	0.40	0.73
Average	0.62	0.65	0.64	0.59	0.63	0.65	0.52	0.49	0.63

Table 7.2: F-Measures from Different Machine Learning Algorithms on Dataset 2. Word2Vec Dataset: GoogleNews-vectors-negative300

7.2.2 Results for the OAEI 'Conference' Dataset

We have used OAEI 2021 'conference' dataset as our dataset 2.

From the results in 7.2, we can see that Random Forrest, Logistic Regression Clas-

sifier, SVM Classifier have the best f-measures, while Decision Tree Classifier, KNN Classifier, Linear Discriminant Analysis have slightly lower performance for dataset2.

7.2.3 Comparison of Results with Existing Works

We have compared the results of our method, OntoAlign, with other state of the art tools/methods that participated in the OAEI competition. The F-Measure scores are taken from the OAEI 2016 benchmark dataset results page (Achichi *et al.* (2016)). The results are compared with our results in Table 7.3. We can see that OntoAlign outperforms all the other tools except CorMatch and Lily for the benchmark dataset.

Algorithm/Method	**F-Measure**
CorMatch	0.89
Lily	0.89
OntoAlign-RandomForest	**0.83**
OntoAlign-LogisticRegression	**0.82**
OntoAlign-SVM	**0.81**
OntoAlign-AdaBoost	**0.81**
OntoAlign-DecisionTree	**0.77**
OntoAlign-LDA	**0.71**
OntoAlign-SGD-Classifier	**0.71**
OntoAlign-MLP	**0.63**
OntoAlign-KNN	**0.61**
XMap	0.56
LogMap	0.55
LogMapLite	0.46
AML	0.38
LogMapBio	0.32

Table 7.3: Comparison with State of the Art Methods Based on Dataset-1.

The results for the OAEI 'conference' dataset is compared with our results in Table 7.4. The F-Measure scores are collected from the OAEI 2021 conference track results page (Pour *et al.* (2020)). We can see that our method using Random Forest and Linear Discriminant Analysis (LDA) outperforms most of the state-of-the-art

methods except the AgreementMakerLight (AML).

Algorithm/Method	**F-Measure**
AgreementMakerLight (AML)	0.69
OntoAlign-RandomForest	**0.65**
OntoAlign-LDA	**0.65**
LogMap	0.64
OntoAlign-AdaBoost	**0.64**
OntoAlign-SVM	**0.63**
OntoAlign-MLP	**0.63**
OntoAlign-LogisticRegression	**0.62**
GMap	0.61
ATMatcher	0.59
Wiktionary	0.59
OntoAlign-DecisionTree	**0.59**
FineTOM	0.58
TOM	0.57
ALOD2Vec	0.56
edna	0.56
LogMapLite	0.56
LSMatch	0.55
AMD	0.54
StringEquiv	0.53
KGMatcher	0.52
OntoAlign-SGD	**0.52**
OntoAlign-KNN	**0.49**

Table 7.4: Comparison with State of the Art Methods Based on Dataset-2.

7.3 Analysis of the Results

Based on the results from both datasets, we can see that Random Forest has given us best results. As discussed in section 6.4.5, random forest works by randomly dividing the features and using them to create multiple decision trees. In this way, it improves the results from using decision trees individually. We can see from Table 7.1 and Table 7.2 that for each instance of the alignments random forest gives us better f-measure than decision tree as expected.

Logistic Regression Classifier and Ada Boost Classifier also gave very similar results for both the datasets. For ontology alignment, the classification is categorical in nature, i.e. the entity tuple in hand either match or does not match. Logistic regression works well in such cases of categorical outputs as we can also find from our results. Ada Boost combines multiple sub models and makes a better model to overcome the weaknesses of the sub models. The concept is quite similar as we saw in random forest. Here instead of using a particular sub-model i.e. decision tree, Ada Boost uses different sub-classifiers and combines the results. For this reason the results of random forest and Ada Boost based models are very close to each other.

Among other state of the art methods, CorMatch and Lily has performed very well compared to our results for dataset1. Lily uses subgraph similarity that makes it find the local meaning of a the components in the entity instead of trying to find similarity based on the overall entity [Wang and Wang (2015)]. We could not find any publication related to CorMatch, only the results reported on the OAEI results website were found.

For dataset2, AML [Faria *et al.* (2013)] performed better than our approach. Instead of using learning based techniques for matching ontologies, it uses four types of matchers namely lexical matcher, mediating matcher, word matcher and parametric string matcher. Comparing to our method, we have used similarity measures to train machine learning models which is analogous to the the matcher defined by AML, but less robust. We believe our results can be improved by incorporating the ideas of using multiple matchers from AML.

Chapter 8

ABLATION STUDIES

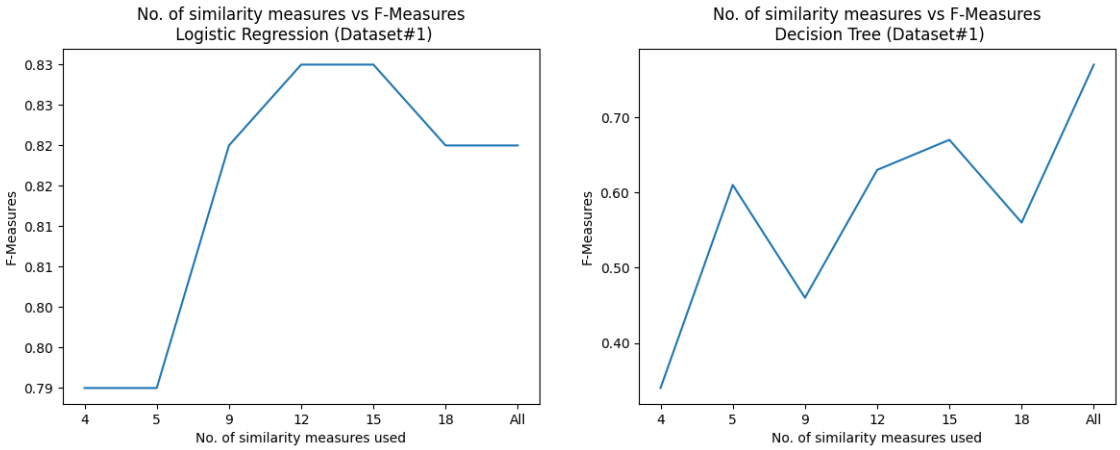
In any machine learning based solution, ablation study is very important. Ablation study is the procedure to compare different approaches against the proposed approach. There are many different parameters for each of the machine learning techniques we used. One purpose of ablation study is to show how changing the parameters affect the classification results. In our work, there are multiple factors that needs to be considered for ablation studies.

8.1 Similarity Measures

One primary observation of our research is the effect of using different number of similarity measures while creating the features for the machine learning methods. In most of the cases we have found that the increase in the number of similarity measures improves the f-measures in the results. Figures 8.1, 8.2, 8.3 and 8.11 shows how increasing the number of similarity measures has changed the outcome for the different machine learning techniques we have used. Most of the algorithms give better performance when we use more similarity measures. For Logistic Regression the f-measure starts falling if we use more that 15 similarity measures. It is also similar for Linear Discriminant Analysis, i.e. the f-measure decreases after 15 similarity measures. SVM doesn't improve much after we use more than 9 similarity measures, but it keeps giving at a certain level of f-measure.

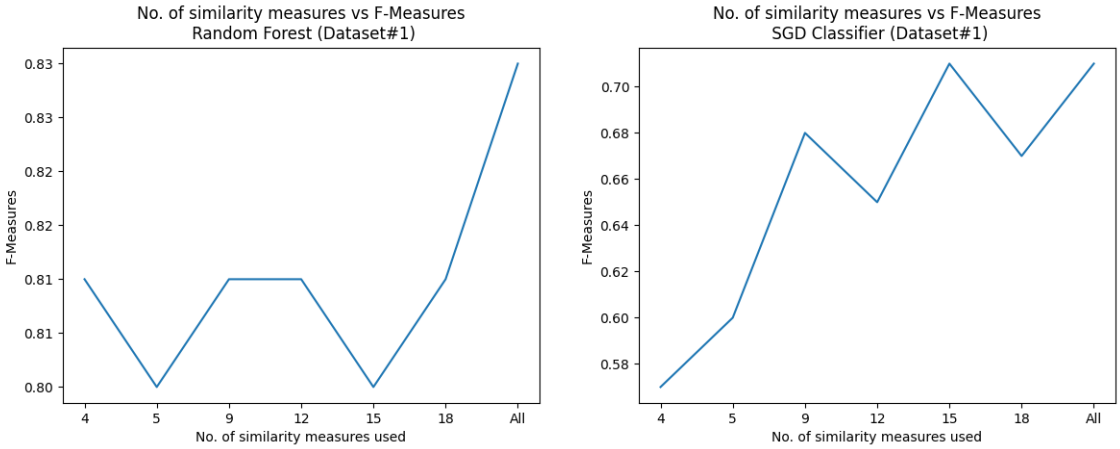
Figure 8.1 shows how logistic regression and decision tree methods gives higher results when higher number of similarity measures are used. For logistic regression we get the best f-measure when 16 similarity measures are used. For decision tree, the best

f-measure is found with all the similarity measures.



(a) Logistic Regression (b) Decision Tree

Figure 8.1: Impact of Number of Similarity Measures on Logistic Regression and Decision Tree (Dataset#1)

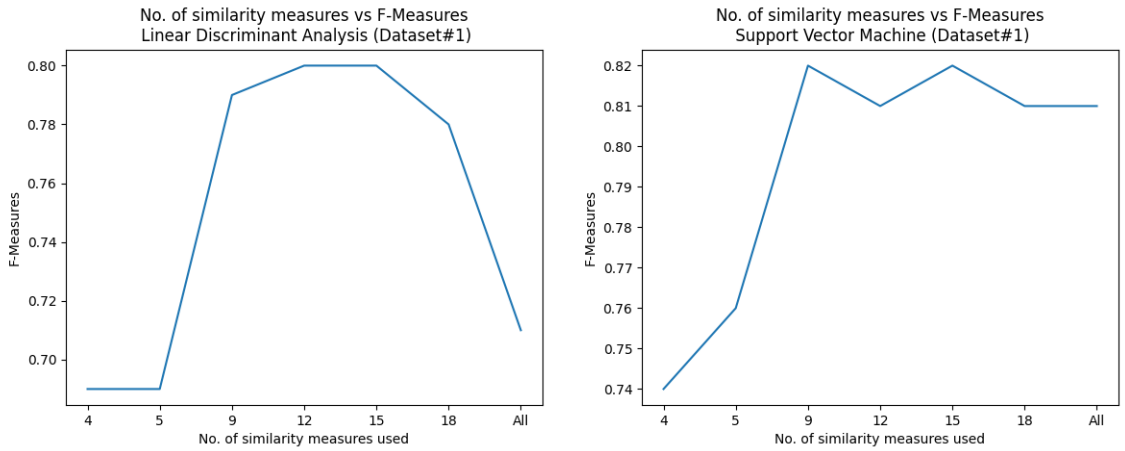


(a) Random Forest (b) SGD Classifier

Figure 8.2: Impact of Number of Similarity Measures on Random Forest and Stochastic Gradient Descent Classifier (Dataset#1)

Figure 8.2 shows the results for random forest and SGD classifier with different numbers of similarity measures used. random forest doesn't change much until we use more than 18 similarity measures. The f-measure improves significantly when all

the similarity measures are used. For SGD classifier, the we get an improvement of the f-measure until 15 similarity measures and then it remains almost same after that with all the similarity measures.

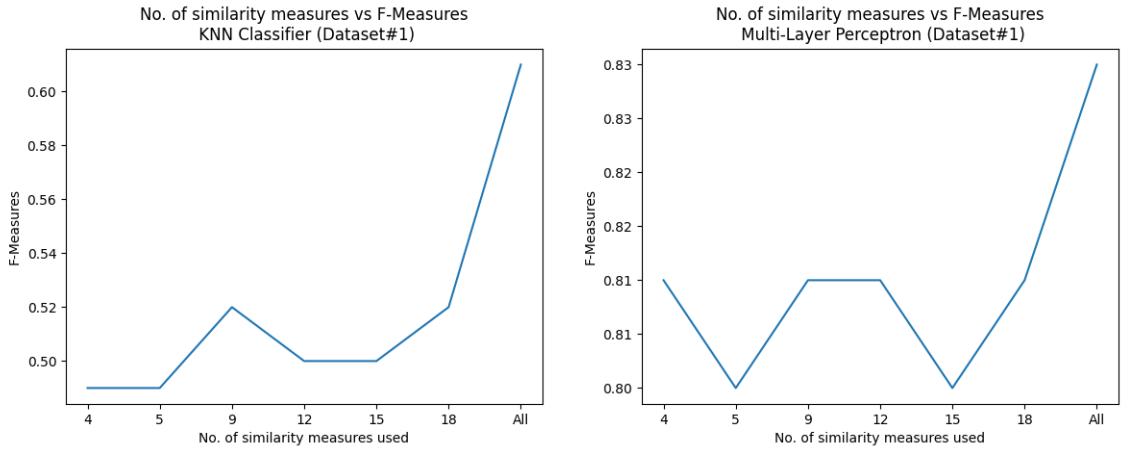


(a) LDA (b) SVM

Figure 8.3: Impact of Number of Similarity Measures on Linear Discriminant Analysis and SVM Classifier (Dataset#1)

Linear discriminant analysis and SVM does not improve much after using 9 similarity measures (Figure 8.3). In fact LDA gives lower results when all the similarity measures are used while the f-measure for SVM remains almost same after 9 similarity measures.

Figure 8.11 also supports the improvement of the machine learning methods KNN and MLP when we take more similarity measures and the f-measures for them peak when all the similarity measures are considered.



(a) KNN (b) MLP

Figure 8.4: Impact of Number of Similarity Measures on K-nearest Neighbour Classifier and MLP Classifier (Dataset#1)

We have also calculated the average of the f-measures for all 9 of the machine learning algorithms against the number of similarity measures. If we look at Figure 8.5, it is clear that using more similarity measures has improved the average F-Measures for the machine learning algorithms.

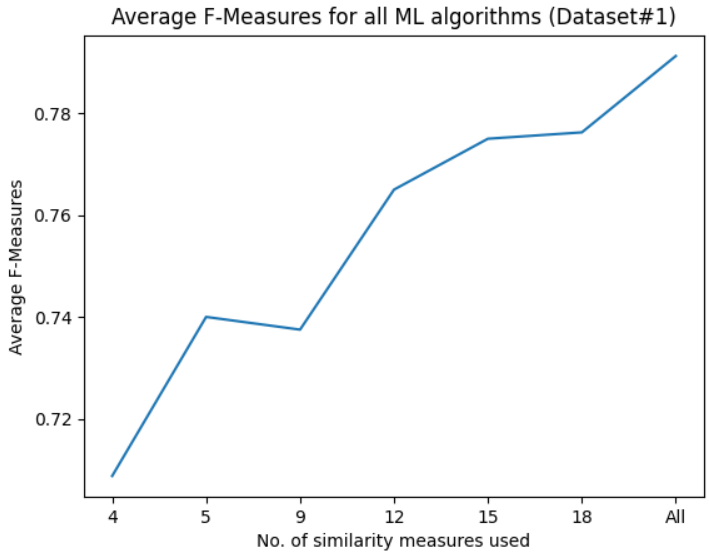


Figure 8.5: Average F-measures for All Machine Learning Methods Against the Number of Similarity Measures Used

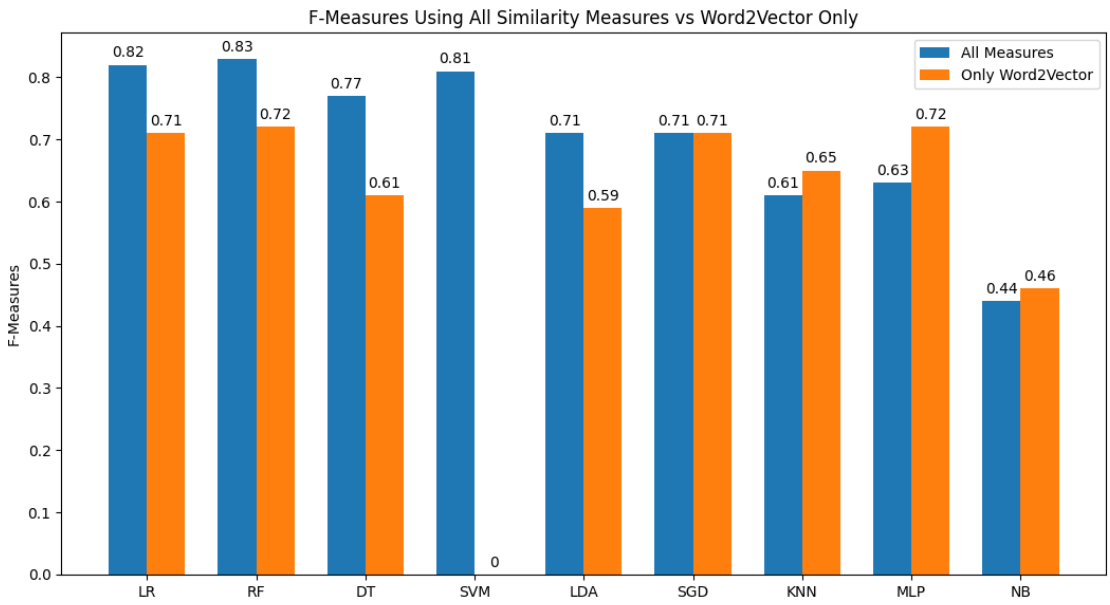


Figure 8.6: Using Word2vector Alone as a Similarity Measure

8.2 Using Word2Vector

We have also observed how the results change if we use only word2vector as a similarity measure. Some of the machine learning algorithms show significant improvement when we used all similarity measures including word2vector. Logistic Regression, Random Forest, Decision Tree, Linear Discriminant Analysis has shown 10% to 20% improvement when all other similarity measures are used along with word2vector. SVM didn't work well with only word2vector. Multi-layer Perceptron, K-Nearest Neighbor and Naive Bayes on the other hand produced better results with only word2vector as a similarity measure. Figure 8.6 shows the comparison.

We have also compared the results excluding word2vector as opposed to including it with all other similarity measures. 8.7 displays the result for both of the cases, i.e. including or excluding word2vector. For almost all methods, the results doesn't change much when word2vector is used except KNN and Decision Tree for which the results improve when we use word2vector.

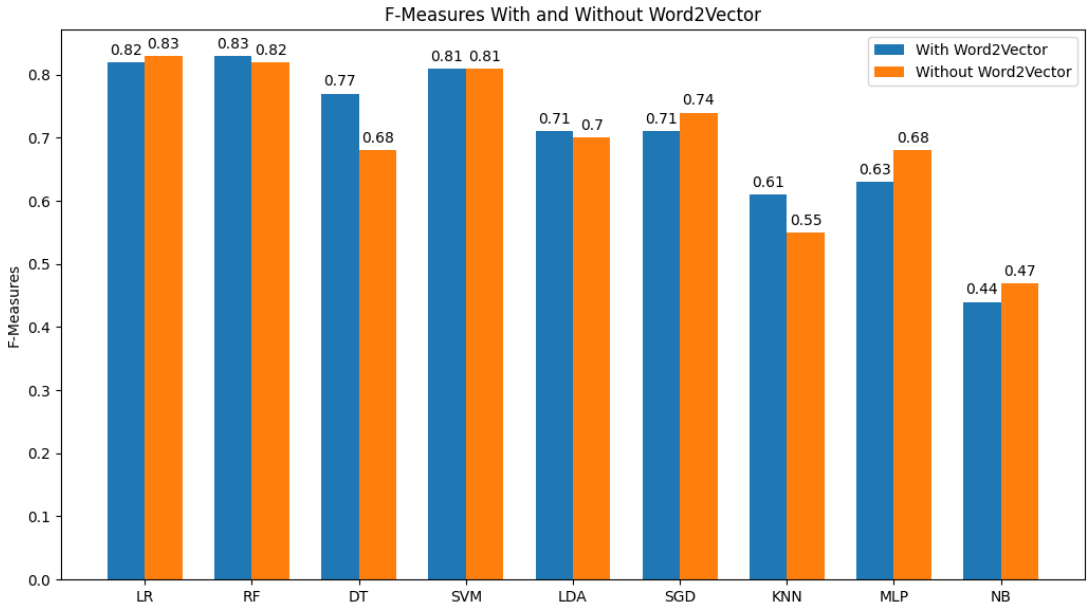


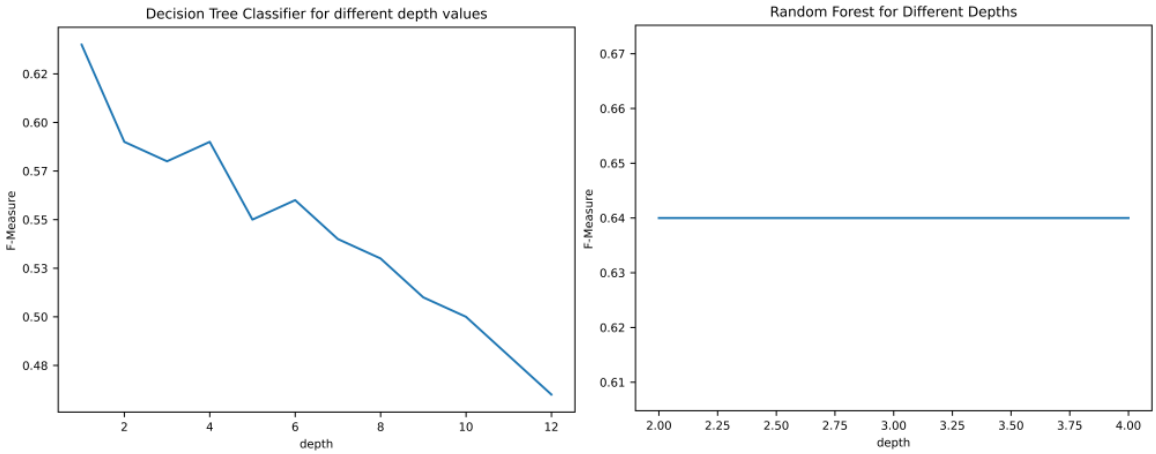
Figure 8.7: Including Word2vector as a Similarity Measure

8.3 Varying ML Parameters

Determining the right parameters for the machine learning methods is important to come up with the optimal model. In this research, we have gone through different values for the important parameters of each machine learning methods. The effects of choosing different parameter values is shown graphically in the following section.

8.3.1 Decision Tree

For decision tree, we have changed the depth of the tree and observed the change in results. The increase of depth actually reduced the f-measures for this method. The results are displayed graphically in Figure 8.8a. For our final model, we have taken $depth = 2$ for decision tree.



(a) Decision Tree - change of depth (b) Random Forest - change of depth

Figure 8.8: Impact of 'Depth' for Decision Tree and Random Forest Method

8.3.2 Random Forest

For random forest, we have changed the depth and observed the change in results. But in this case, there were no effect of changing the depth. Figure 8.8b shows the results. We have also changed the number of estimators for random forest and observed the change in results. The increase in the number of estimators actually worsened the result in this case. Figure 8.9 shows the results.

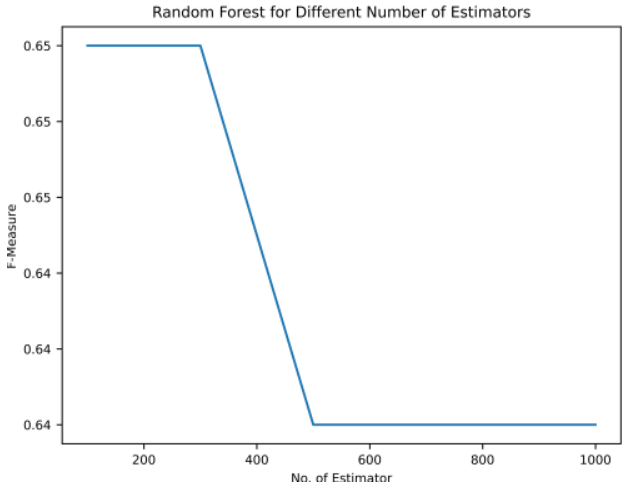
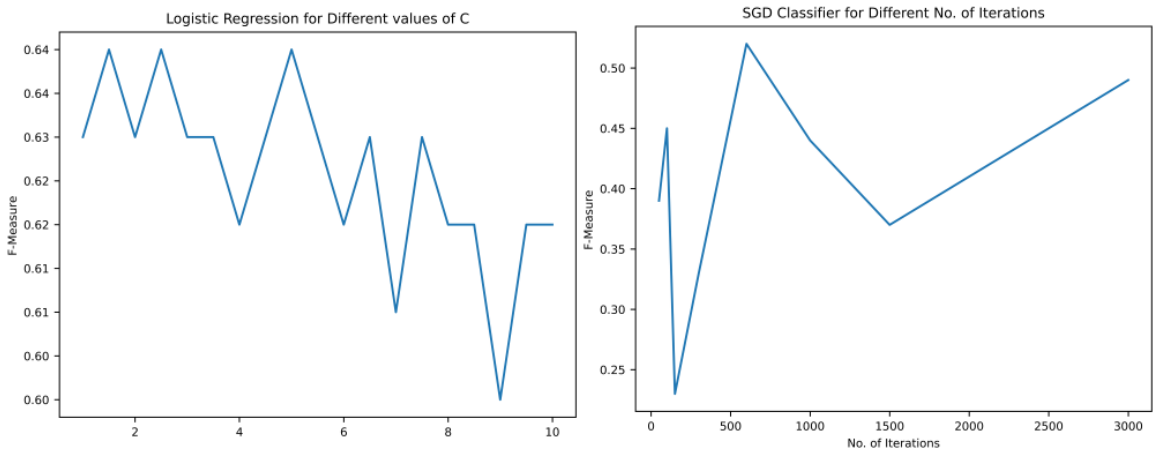


Figure 8.9: Random Forest - Change of Estimators

8.3.3 *Logistic Regression*

For logistic regression, we changed the c-value in the algorithm and observed the change in results. It appeared that $c = 5$ gives us the best results. Figure 8.10a shows the results.



(a) Logistic Regression - change of C-value

(b) SGD Classifier - number of iterations

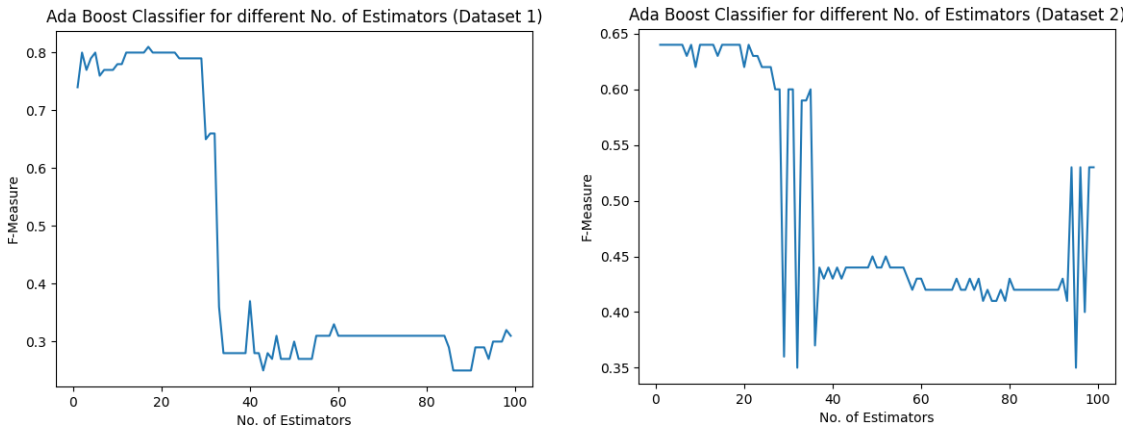
Figure 8.10: Impact of Parameter Change on Logistic Regression and SGD Classifier

8.3.4 *SGD Classifier*

For SGD classifier, in general, the increase in the number of iterations gives better results as shown in Figure 8.10b

8.3.5 *Ada Boost Classifier*

For Ada Boost classifier, we have tried different number estimators. For Dataset#1, number of estimators 17 gives us best result. For Dataset#2, number estimators 1 gives best result.



(a) Impact on Dataset#1 (b) Impact on Dataset#2

Figure 8.11: Impact of the Change of Number of Estimators on Ada Boost Classifier

8.3.6 LDA

For LDA classifier we have tested for different shrinkage values for 'eigen' solver. In general, the increase in the value of 'shrinkage' gives less accurate results as shown in Figure 8.12

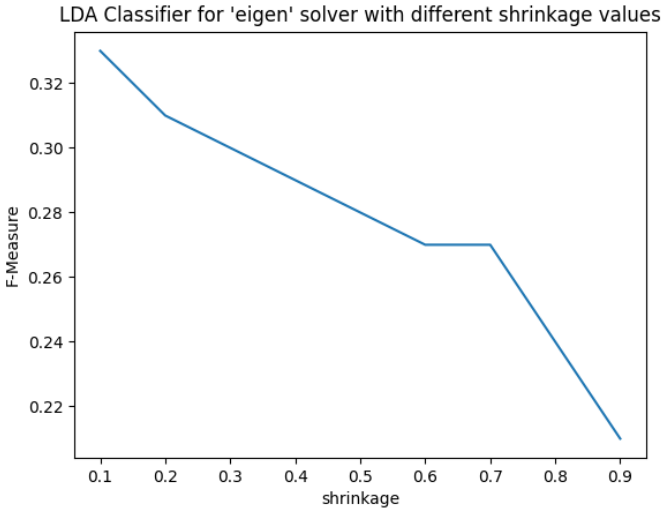


Figure 8.12: LDA - Change of Shrinkage Value for Eigen Solver

8.3.7 *KNN Classifier*

For KNN, we have used different values of K. For dataset1, KNN gives best result for $K = 11$ and f-measures doesn't improve after that. For dataset2, KNN gives best result for $K = 6$ and f-measures start getting lower after that. The results are shown in Figure 8.13 and Figure 8.14

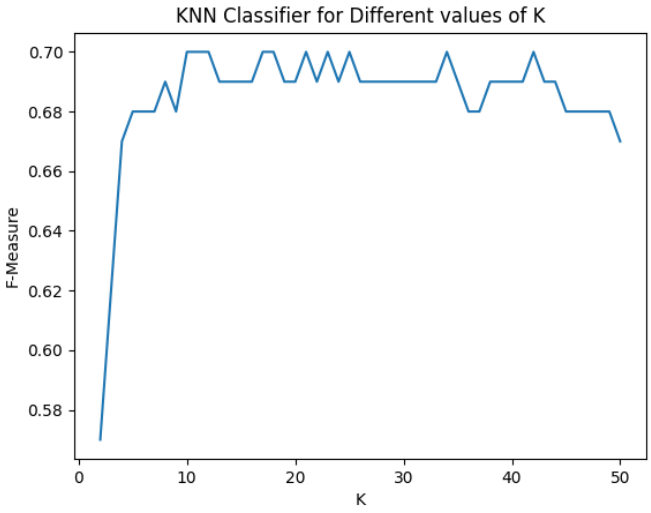


Figure 8.13: KNN Classifier - Change of K (Dataset#1)

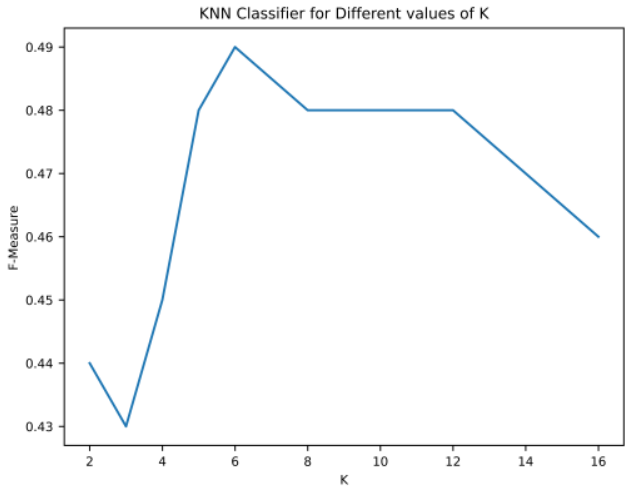


Figure 8.14: KNN Classifier - Change of K (Dataset#2)

8.3.8 *MLP Classifier*

For MLP classifier, we have tried different number of layers. For Dataset#1, number of hidden layers (4, 2) gives us best result. For Dataset#2, number of hidden layers=(6, 4) gives best result.

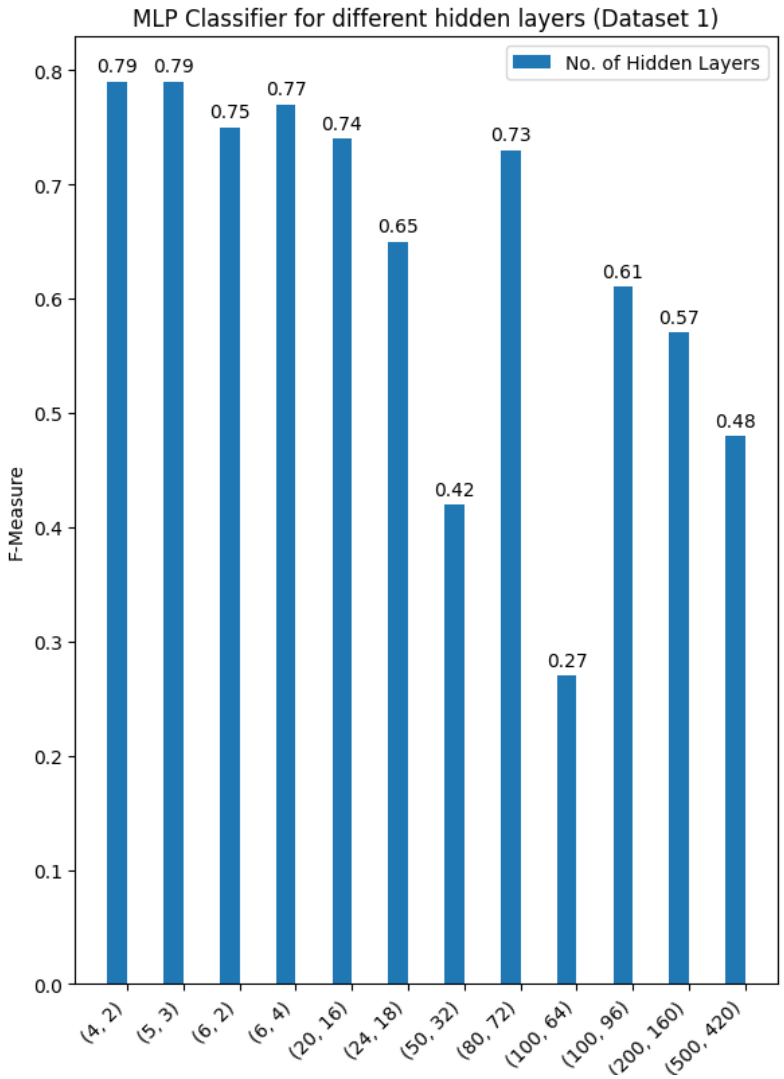


Figure 8.15: MLP Classifier - Number of Layers (Dataset#1)

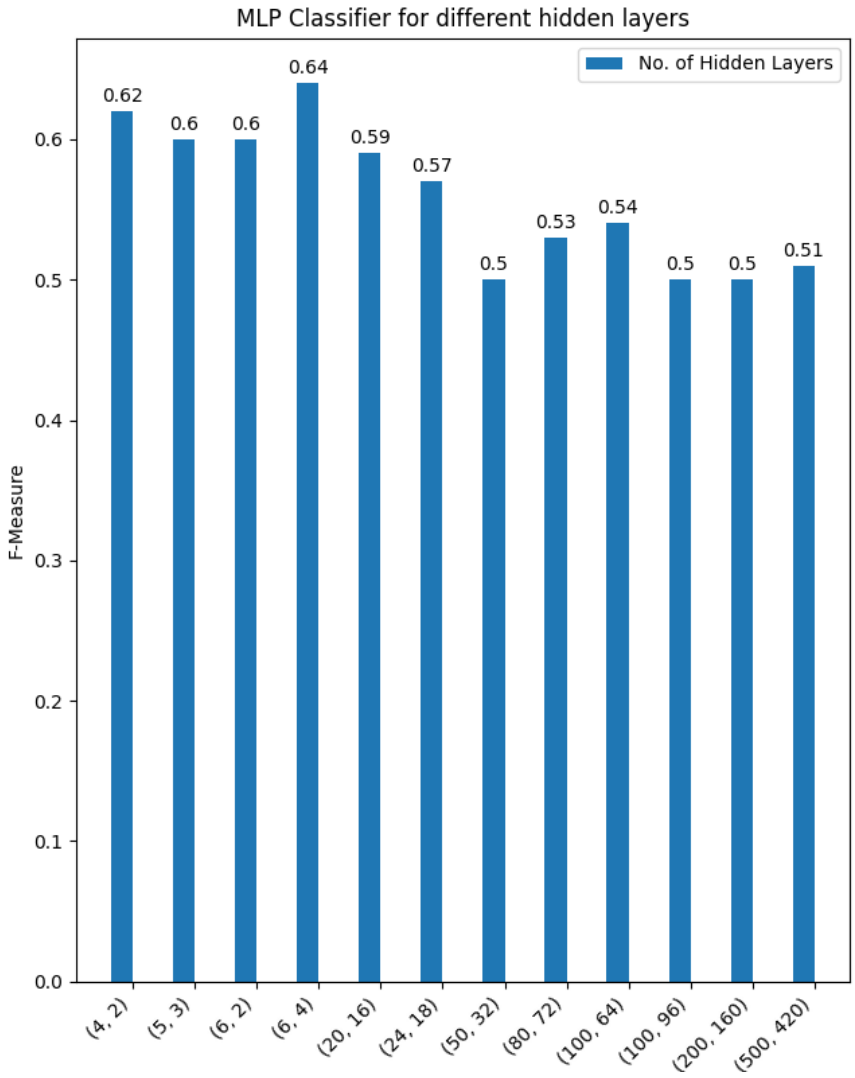


Figure 8.16: MLP Classifier - Number of Layers (Dataset#2)

Chapter 9

FUTURE WORK

9.1 Possible Extension of Current Work

There are datasets that are larger in size. If those datasets can be used to train the machine learning models, it might lead to finding better models that can give better results for a wide range of ontologies from multiple domains. In our experiment, due to limited processing power and memory, we could only use the two datasets mentioned above (benchmark and conference datasets from OAEI). With higher processing power and memory, it will be also possible to combine multiple word2vector libraries, which might improve the matching algorithm further. In our case, we could use upto three word2vector libraries.

9.2 Neural Network based solutions

Neural Network based solutions should perform well in ontology matching. Since it is often very difficult to get properly annotated or labeled data for ontology-alignment, unsupervised approaches can come to rescue. Chakraborty et. al. proposed OntoConnect Chakraborty *et al.* (2021) where they have demonstrated a very effective method using recursive neural networks. It would be an interesting work if all the similarity measures used in current work can be used with the recursive neural network based solution proposed by OntoConnect.

9.3 Transfer Learning

Transfer learning is being used in many recent works especially in the fileds of NLP and Image Processing. When we don't have enough data with ground truth for our domain in question, we can try training a deep learning model from another domain that has enough data and use that pre-trained model to predict for the domain we need results for. For example, there are sometimes not enough data related to medical image processing. But, since there are enough data from natural images on the web, a deep learning model can be trained based on the natural images, and later that pre-trained model can be used to predict features (cancers, tumors etc) in medical images.

Chapter 10

CONCLUSION

In this research we have designed an ontology matcher and conducted experiments on ontology alignment using a couple of well known datasets from the Ontology Alignment Evaluation Initiative (OAEI) challenge. We propose a model based on specialized feature extraction and using supervised machine learning models that will be able to handle large datasets from the real world and provide better results in terms of ontology matching. We have shown that feature extraction using larger number of similarity measures between entities of the ontologies can produce better ontology matching. The primary contribution of this research is the concept of using multiple types of similarity measures to create features for machine learning models. Our method, OntoAlign, can be used to automatically find alignments between ontology pairs without any human intervention. The results are highly comparable to the existing state of the art tools and methods presented in the OAEI competition.

REFERENCES

Achichi, M., M. Cheatham, Z. Dragisic, J. Euzenat, D. Faria, A. Ferrara, G. Flouris, I. Fundulaki, I. Harrow, V. Ivanova *et al.*, "Results of the ontology alignment evaluation initiative 2016", in "OM: Ontology matching", No. 1766, pp. 73–129 (2016).

Atencia, M., J. Euzenat, G. Pirro and M.-C. Rousset, "Alignment-based trust for resource finding in semantic p2p networks", in "International Semantic Web Conference", pp. 51–66 (Springer, 2011).

Aumueller, D., H.-H. Do, S. Massmann and E. Rahm, "Schema and ontology matching with coma++", in "Proceedings of the 2005 ACM SIGMOD international conference on Management of data", pp. 906–908 (2005).

Berners-Lee, T., J. Hendler and O. Lassila, "The semantic web", Scientific american **284**, 5, 34–43 (2001).

Bernstein, A., J. Hendler and N. Noy, "A new look at the semantic web", Communications of the ACM **59**, 9, 35–37 (2016).

Bouquet, P., F. Giunchiglia, F. v. Harmelen, L. Serafini and H. Stuckenschmidt, "C-owl: Contextualizing ontologies", in "International Semantic Web Conference", pp. 164–179 (Springer, 2003).

Bühmann, L., J. Lehmann, P. Westphal and S. Bin, "Dl-learner structured machine learning on semantic web data", in "Companion Proceedings of the The Web Conference 2018", pp. 467–471 (2018).

Bulygin, L. and S. A. Stupnikov, "Applying of machine learning techniques to combine string-based, language-based and structure-based similarity measures for ontology matching.", in "DAMDID/RCDL", pp. 129–147 (2019).

Callan, R., *Essence of neural networks* (Prentice Hall PTR, 1998).

Chakraborty, J., S. K. Bansal, L. Virgili, K. Konar and B. Yaman, "Ontoconnect: Unsupervised ontology alignment with recursive neural network", in "Proceedings of the 36th Annual ACM Symposium on Applied Computing", pp. 1874–1882 (2021).

Cohen, R. P., W. and S. Fienberg, "A comparison of string metrics for matching names and records", In Kdd workshop on data cleaning and object consolidation **3**, 73–78 (2003).

Cohen, R. P., W. and S. Fienberg, "A comparison of string metrics for matching names and records", In Kdd workshop on data cleaning and object consolidation **3**, 73–78 (2007).

David, J., "Association rule ontology matching approach", International Journal on Semantic Web and Information Systems (IJSWIS) **3**, 2, 27–49 (2007).

Doan, A., J. Madhavan, R. Dhamankar, P. Domingos and A. Halevy, "Learning to match ontologies on the semantic web", The VLDB journal **12**, 4, 303–319 (2003).

Doan, H. A. I. Z., A., "Principles of data integration", (2012).

Euzenat, J., "An api for ontology alignment", In International Semantic Web Conference. Springer, Berlin, Heidelberg. **November**, 698–712 (2004).

Euzenat, J., M.-E. Roşoiu and C. Trojahn, "Ontology matching benchmarks: generation, stability, and discriminability", Journal of web semantics **21**, 30–48 (2013).

Euzenat, J. and P. Shvaiko, "Ontology matching", Springer-Verlag Berlin Heidelberg, Berlin **18** (2007).

Euzenat, J., P. Shvaiko *et al.*, *Ontology matching*, vol. 18 (Springer, 2007).

Faria, D., C. Pesquita, E. Santos, M. Palmonari, I. F. Cruz and F. M. Couto, "The agreementmakerlight ontology matching system", in "OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"", pp. 527–541 (Springer, 2013).

Fisher, R. A., "The use of multiple measurements in taxonomic problems", Annals of eugenics **7**, 2, 179–188 (1936).

Giunchiglia, F., P. Shvaiko and M. Yatskevich, "S-match: an algorithm and an implementation of semantic matching", in "European semantic web symposium", pp. 61–75 (Springer, 2004).

Gracia, J., J. Bernad and E. Mena, "Ontology matching with cider: evaluation report for oaei 2011", Ontology Matching **126** (2011).

Guarino, N., D. Oberle and S. Staab, "What is an ontology?", in "Handbook on ontologies", pp. 1–17 (Springer, 2009).

Hitzler, P., "A review of the semantic web field", Communications of the ACM **64**, 2, 76–83 (2021).

Hlaing, S. S., "Ontology based schema matching and mapping approach for structured databases", in "Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human", pp. 853–859 (2009).

Jiménez-Ruiz, E., B. C. Grau and I. Horrocks, "Logmap and logmaplt results for oaei 2012", Ontology Matching **152** (2013).

Lamy, J.-B., "Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies", Artificial intelligence in medicine **80**, 11–28 (2017).

Li, J., J. Tang, Y. Li and Q. Luo, "Rimom: A dynamic multistrategy ontology alignment framework", IEEE Transactions on Knowledge and data Engineering **21**, 8, 1218–1232 (2008).

Li, X., X. L. Dong, K. B. Lyons, W. Meng and D. Srivastava, "Scaling up copy detection", in "2015 IEEE 31st International Conference on Data Engineering", pp. 89–100 (IEEE, 2015).

Liang, S.-F., C.-E. Kuo, Y.-H. Hu and Y.-S. Cheng, "A rule-based automatic sleep staging method.", in "33rd IEEE EMBS Annual International Conference of the Engineering in Medicine and Biology Society", pp. 6067–6070 (2011).

Mascardi, V., D. Ancona, R. H. Bordini and A. Ricci, "Cool-agentspeak: Enhancing agentspeak-dl agents with plan exchange and ontology services", in "2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology", vol. 2, pp. 109–116 (IEEE, 2011).

Mikolov, T., K. Chen, G. Corrado and J. Dean, "Efficient estimation of word representations in vector space", (2013).

Needleman, W. C., S., "A general method applicable to search for similarities in amino acid sequence of 2 proteins", In: Journal of Molecular Biology **48**, 3, 443–53 (1970).

Nezhadi, A. H., B. Shadgar and A. Osareh, "Ontology alignment using machine learning techniques", AIRCC's International Journal of Computer Science and Information Technology **3**, 2, 139–150 (2011).

Ngo, D. H. and Z. Bellahsene, "Yam++:(not) yet another matcher for ontology matching task", in "BDA: Bases de Données Avancées", (2012).

Ngomo, A.-C. N. and S. Auer, "Limes—a time-efficient approach for large-scale link discovery on the web of data", in "Twenty-Second International Joint Conference on Artificial Intelligence", (2011).

O'Leary, D., "Ontologies: A silver bullet for knowledge management and electronic commerce", (2005).

Otero-Cerdeira, L., F. J. Rodríguez-Martínez and A. Gómez-Rodríguez, "Ontology matching: A literature review", Expert Systems with Applications **42**, 2, 949–971 (2015).

Pour, N., A. Algergawy, R. Amini, D. Faria, I. Fundulaki, I. Harrow, S. Hertling, E. Jiménez-Ruiz, C. Jonquet, N. Karam *et al.*, "Results of the ontology alignment evaluation initiative 2020", in "Proceedings of the 15th International Workshop on Ontology Matching (OM 2020)", vol. 2788, pp. 92–138 (CEUR-WS, 2020).

Rahm, E. and P. A. Bernstein, "A survey of approaches to automatic schema matching", the VLDB Journal **10**, 4, 334–350 (2001).

Rao, S. G. V. K., G.A. and P. Prasad Reddy, "A partial ratio and ratio based fuzzy-wuzzy procedure for characteristic mining of mathematical formulas from documents", IJSC—ICTACT J Soft Comput **8**, 4, 1728–1732 (2018).

Resnik, P., "Using information content to evaluate semantic similarity in a taxonomy", arXiv preprint cmp-lg/9511007 (1995).

Shvaiko, P. and J. Euzenat, "A survey of schema-based matching approaches", in "Journal on data semantics IV", pp. 146–171 (Springer, 2005).

Signore, O. *et al.*, "Representing knowledge in the semantic web", in "Open Culture Conference (organised by the Italian office of W3C)", pp. 27–29 (2005).

Sousa, T., A. Cruz, S. Khalighi, G. Pires and U. Nunes, "A two-step utomatic sleep stage classification method with dubious range detection", Computers in Biology and Medicine **59**, 42–43 (2015).

Stoilos, S. G., G. and S. Kollias, "A string metric for ontology alignment", In International Semantic Web Conference. Springer, Berlin, Heidelberg. **3**, 624–637 (2005a).

Stoilos, S. G., G. and S. Kollias, "A string metric for ontology alignment", In International Semantic Web Conference. Springer, Berlin, Heidelberg **November**, 624–637 (2005b).

Straccia, U. and . Troncy, R., "omap: Combining classifiers for aligning automatically owl ontologies", In International Conference on Web Information Systems Engineering. Springer, Berlin, Heidelberg **November**, 133–147 (2005).

Suchanek, F. M., S. Abiteboul and P. Senellart, "Paris: Probabilistic alignment of relations, instances, and schema", arXiv preprint arXiv:1111.7164 (2011).

Tversky, A., "Features of similarity", In: Psychological Review **84**, 4, 327–352 (1977).

Vijaymeena, M. and K. Kavitha, "A survey on similarity measures in text mining", Machine Learning and Applications: An International Journal **3**, 2, 19–28 (2016).

Wang, W. and P. Wang, "Lily results for oaei 2015.", in "OM", pp. 162–170 (2015).

Wu, P. M., Z., "Verbs semantics and lexical selection", In Proceedings of the 32nd annual meeting on Association for Computational Linguistics (1994).

Wu, Z. and M. Palmer, "Verb semantics and lexical selection", arXiv preprint cmp-lg/9406033 (1994).

Zhang, Y., X. Wang, S. Lai, S. He, K. Liu, J. Zhao and X. Lv, "Ontology matching with word embeddings", in "Chinese computational linguistics and natural language processing based on naturally annotated big data", pp. 34–45 (Springer, 2014).

Zobel, J. and P. Dart, "Phonetic string matching: Lessons from information retrieval", In Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval pp. 166–172 (1996).