OntoConnect: Domain-Agnostic Ontology Alignment using Neural Networks

by

Jaydeep Chakraborty

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

Approved April 2021 by the
Graduate Supervisory Committee:

Srividya Bansal, Chair
Mohamed Sherif
Ajay Bansal
Sharon Hsiao

ARIZONA STATE UNIVERSITY

May 2021

©2021 Jaydeep Chakraborty

All Rights Reserved

ABSTRACT

An ontology is a vocabulary that provides a formal description of entities within a domain and their relationships with other entities. Along with basic schema information, it also captures information in the form of metadata about cardinality, restrictions, hierarchy, and semantic meaning. With the rapid growth of semantic (RDF) data on the web, many organizations like DBpedia, Earth Science Information Partners (ESIP) are publishing more and more data in RDF format. The ontology alignment task aims at linking two or more different ontologies from the same domain or different domains. It is a process of finding the semantic relationship between two or more ontological entities and/or instances. Information/data sharing among different systems is quite limited because of differences in data based on syntax, structures, and semantics. Ontology alignment is used to overcome the limitation of semantic interoperability of current vast distributed systems available on the Web.

In spite of the availability of large hierarchical domain-specific datasets, automated ontology mapping is still a complex problem. Over the years, many techniques have been proposed for ontology instance alignment, schema alignment, and link discovery. Most of the available approaches require human intervention or work within a specific domain. The challenge involves representing an entity as a vector that encodes all context information of the entity such as hierarchical information, properties, constraints, etc. The ontological representation is rich in comparison with the regular data schema because of metadata about various properties, constraints, relationship to other entities within the domain, etc. While finding similarities between entities this metadata is often overlooked. The second challenge is that the comparison of two ontologies is an intense operation and highly depends on the domain and the language that the ontologies are expressed in. Most current methods require human intervention that leads to a time-consuming and cumbersome process and the output is prone to human errors.

The proposed unsupervised recursive neural network technique achieves an F-measure of 80.3% on the Anatomy dataset and the proposed graph neural network technique achieves an F-measure of 81.0% on the Anatomy dataset.

DEDICATION

I dedicate this thesis to my parents who have been a great source of inspiration, love, and support. This thesis is also dedicated to Dr. Srividya Bansal who encouraged me in every step during my journey in Ph.D.

ACKNOWLEDGMENTS

The completion of the dissertation has been possible because of the support and contributions of many. Foremost, I would like to express my sincere gratitude and respect towards my supervisor Dr. Srividya Bansal. Her profound insights inspired me to explore the challenging areas of Ontology Alignment. Her guidance and helpful advice throughout the project has made the goals of the project achievable.

I am grateful to my committee members – Dr. Mohamed Sherif, Dr. Ajay Bansal, and Dr. Sharon Hsiao for their constructive criticism as well as insightful suggestions which helped me improve and prepare my final dissertation. I am appreciative of the valuable and wise counsel of all the professors of the Department of Computer Science.

I would like to acknowledge the support of the DBpedia organization during the Google Summer Code program in helping me understand the depth of the thesis problem statement and guiding me to work with different aspects of the problem.

I am immensely thankful to the Google Cloud Platform (GCP) credits program for allowing me to use Cloud Computing services for the execution of the research experiments. I would also like to thank Arizona State University for providing the necessary resources to continue my research work.

I take this opportunity to convey my regards and obligation towards my family and friends who have been constant sources of support and encouragement in my life. I wish to appreciate all the technical and non-technical staff of Arizona State University for their sympathetic cooperation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Due to the widespread usage of the World Wide Web, social media, devices, and shared applications, digital data is getting bigger and bigger. Because of the heterogeneous nature of data, it is difficult to annotate data. With the increase in the amount of data on the web, it is getting harder to make it machine-readable and machine-understandable. The main goal of the semantic web is to provide a uniform machine-readable format to all web resources and create a web of linked data. Ontologies are used to provide a representation for concepts and relationships between concepts thereby providing a semantic structure to data that is machine-readable and machine-understandable. The semantic web is an extension of the web which helps to represent information in a more meaningful way by providing a description of the content and meta-information. It enables information on the web more accessible and understandable by both humans and machines. Ontology is one the knowledge representations which provide a formal description of knowledge set which can be easily shared among human and machines. An ontology can be defined as a formal description of knowledge as a set of concepts within a domain and the relationships that hold between them. There are mainly four components in an ontology. **(i)** Class/Concept: It represents a group or collection of objects. **(ii)** Instance: It represents a specific object or element of a class/concept. **(iii)** Relation: It is the relationship between class/concepts in a given domain. **(iv)** Restriction: It is used to impose constraints on the values of classes/concepts and instances. Figure 1 shows an example of the ontology in a university domain. In the following ontology, “Course”, “Student”, “Professor” are classes/concepts. “takesCourse” is a relation between class “Student” and “Course” which depicts that the students are enrolled in a course. Figure 1 also includes the instances like “Brian Rice”, “GCP: Fundamentals”. Here, “Brian Rice” is an instance of class/concept “Professor” and “GCP: Fundamentals” is an instance of the class/concept “Course”. The upper part of the figure shows the schema (RDFS) of the university domain and the lower part shows a single instance of data from the domain.

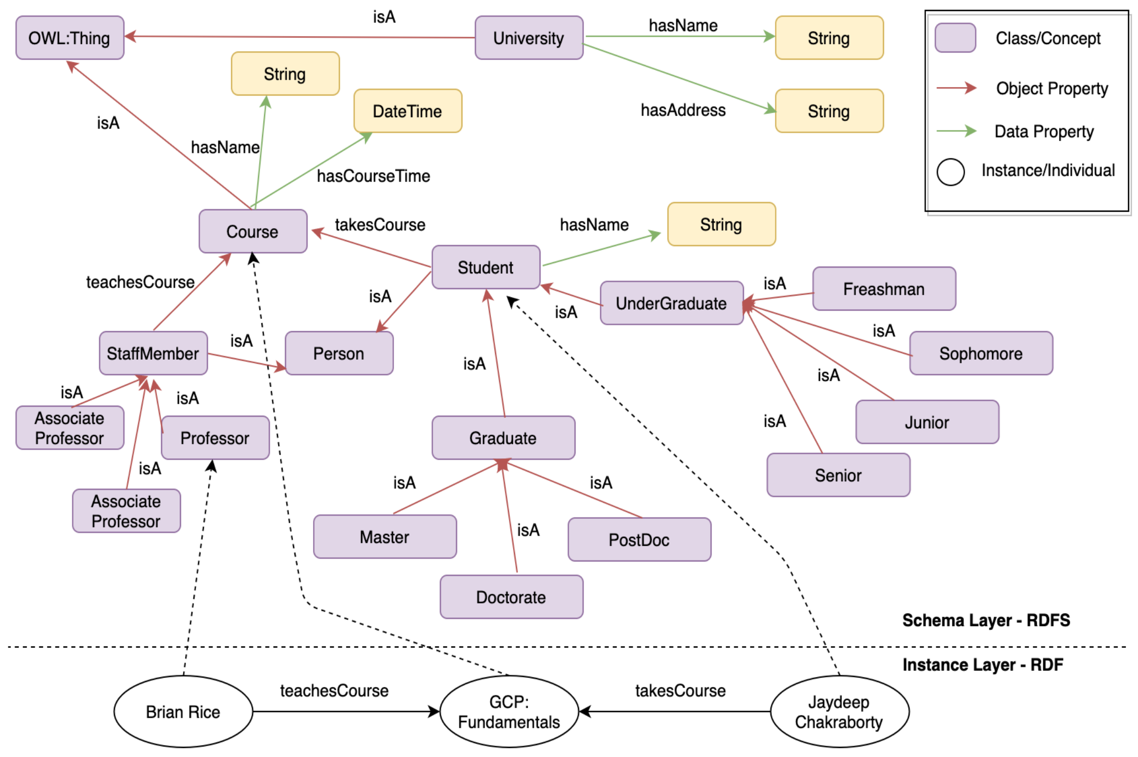


Figure 1: University Ontology

There are a number of processes to achieve semantic interoperability and ontology alignment is one of them. It is a process that can be used to establish semantic relationships and links between entities from different ontologies. In the next section, ontology alignment is discussed in detail.

1.1 Problem Description

Information or data sharing among different systems is very limited due to the heterogeneous nature of data in terms of syntax, structure, and semantic. Nowadays many applications, including bioinformatics, e-commerce, e-learning, information extraction, and web services use ontology instead of a traditional database. An ontology is a formal description of knowledge as a set of concepts and relationships between them representing a specific domain. It enables the database or knowledge source interoperability and is easy to reuse and can easily be used to capture domain knowledge, however, different applications use different ontologies to represent data from the same domain. There is a requirement for a reusable component to establish a semantic mapping between multiple ontologies. Ontology alignment is a process of finding the semantic relationships between two or more different ontological entities. Ontology alignment is an integral part of creating Linked Data that involves a process of publishing and linking structured data on the web. It is used to overcome the limitation of semantic interoperability between current vast distributed systems available on the internet. Despite the advancement in Linked Data, the ontology alignment is still mostly done manually by domain experts which is labor-intensive and error-prone. In this dissertation, we will discuss a novel method that applies unsupervised machine learning techniques to find mappings between concepts of different ontologies which require no domain-expert intervention.

Ontology alignment or linking between source and target ontology is generally done on two different levels. The first one is schema-level and the second one is instance-level. Schema level alignment is the alignment between concepts/classes of the source and target ontology and on the other hand, instance-level linking is mapping/alignment between instances/individuals of the source and target ontology. There is another type of alignment which is mixed of schema level and instance level of linking. In our approach, we will focus on schema level linking.

Figure 2 depicts a matching task between instances and schema of the Knowledge Graph Track in OAEI-2019 Shvaiko and Euzenat 2011. It contains the isolated knowledge graphs with instance and schema data. The goal of this task is to match both the instances and the schema. Schema alignments are shown in blue, while instance alignments are shown in red. In the figure, “Movie” and “Film” are class/concept, “starring” and “starred” are properties/relations, and “Chris_Evans” and “Christopher_Robert_Evans” are instances of the source and target ontologies respectively.

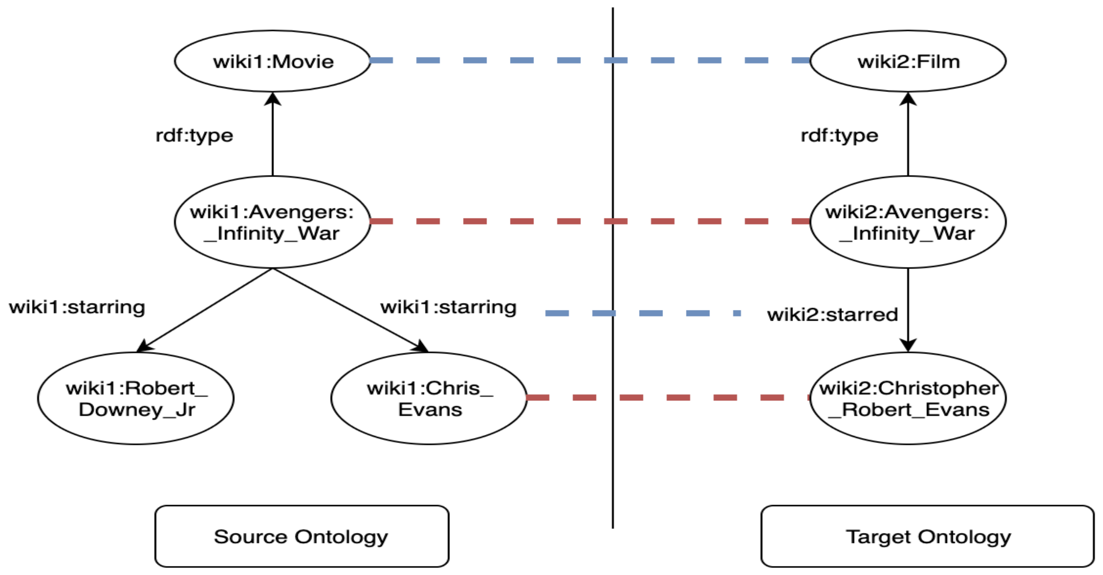


Figure 2: Schema Alignment and Instance Alignment.

To describe the problem, let′s take an example of an anatomy data set published in Ontology Alignment Evaluation Initiative “OAEI” 2020. The anatomy data set has two ontologies, one is the adult mouse anatomy ontology and another is the human anatomy ontology. The task is to find an alignment between concepts of mouse and human ontology.

In Figure 3 the left-side represents a set of body parts of mouse and the right-side represents a set of body parts of human. As we can see both left and right side encoded domains at different levels of details as they are designed independently and designed for different purposes like mouse anatomy versus human anatomy.

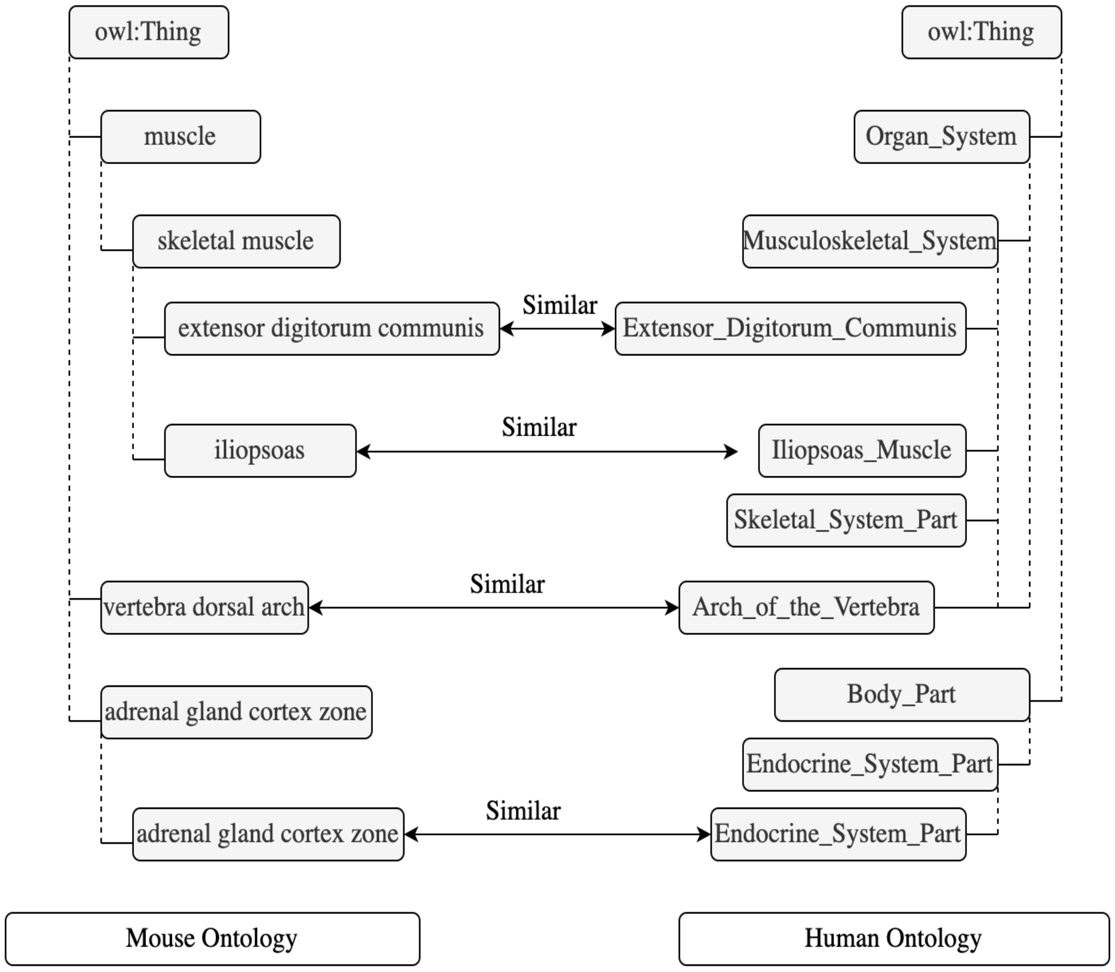


Figure 3: Adult mouse anatomy and human anatomy comparison

In this dissertation, we aim to link the classes/concepts, i.e., body parts mentioned in both human and mouse anatomy ontology so that we can create a linked dataset that can be useful for information extraction, querying, or Extract, Transform, and Load (ETL) pipelines.

1.2 Challenges

Data heterogeneity is the main challenge in creating linked data. The main purpose of ontology alignment is to remove heterogeneity between different data sets or data schemas. There could be different types or ranges of heterogeneity in data sets. Syntactic heterogeneity, Terminological heterogeneity, Conceptual heterogeneity, are among the main reasons which make alignment a cumbersome and erroneous process.

Decentralization is the key idea behind the Semantic Web. Given this nature, there is an exponential explosion in the number of ontologies. Although ontologies describe the same/similar domains, it is impossible to query over all the ontologies as they are not linked. Publishing a linked data system on large scale comprises a number of issues. Ontology Alignment Evaluation Initiative ("OAEI" 2020) is an initiative that organizes and addresses some of these issues. For example, Large BioMed Track ("LargeBIO" 2020) consists of three different ontologies: Foundational Model of Anatomy ("FMA" 2020), SNOMED CT ("SNOMED" 2020), and the National Cancer Institute Thesaurus ("NCI" 2020). Here, FMA contains 78,989 classes, NCI contains 66,724 classes, and SNOMED contains 30,6160 classes. All these three ontologies contain different concepts/classes from the same Bio-medical domain. Ontology alignment includes reading and parsing large ontology files, data pruning, creating training and testing data, encapsulating different information like syntactic, semantic, terminological, structural, etc, and evaluation. These processes require stable tools and techniques. In Ontology Alignment Evaluation Initiative ("OAEI" 2020), it is evident that such stable, mature, and reliable tools or process architecture are still not available.

The **performance** of Ontology alignment techniques is a big issue in the real-world semantic web. In Ontology Alignment Evaluation Initiative - Large BioMed Track - 2018 ("LargeBIO" 2020), it was reported that only seven systems were able to complete all six tasks and ten out of eighteen participating systems have been able to complete at least one of the tasks. These statistics prove that creating dynamic real-world semantic web applications requires a vast knowledge of coding which can be performed with limited memory.

Although there is the **availability** of a large number of ontologies, most of the time the background knowledge is missing. The ontologies are created by domain experts based on certain background knowledge as context but unfortunately, that background knowledge can not be part of the ontology. Most of the time lexical and structural information of an entity is not enough to find the linking between classes/concepts of different ontologies. It is very important to incorporate some amount of background knowledge to get good results with a higher confidence score.

Ontology alignment is **unpredictable**. There are a number of matches available and each produces different results and there is no explanation or logical reasoning of the result of the alignment techniques. As there is no explanation or logic of the behavior of matches, it is very difficult to select a proper matcher or combination of matches. The performance of a matcher depends on many criteria such as the domain of the data, expressiveness of the data, memory usage, choice of matcher parameters such as weights, threshold, etc. The process of selecting and combining matches based on multiple criteria in dynamic semantic web applications can not be automated fully. So the performance of a matcher can not be predictable or fixed as the matcher parameters need to be fine-tuned or changed for the different data sets.

Lack of ground truth is a massive problem to evaluate different alignment systems. Though there are different approaches available, their completeness and accuracy calculation are not reliable. To evaluate any ontology alignment system, the involvement of domain experts is necessary, however, user involvement always introduces human error and ambiguity and is more time-consuming.

Apart from the above difficulties, the **differences in literal expressions** of the ontological entities make the ontology alignment unpredictable. String-based approaches fail to calculate correct similarity in ontology alignment.

- Entity names which are syntactically similar but have different naming conventions are used. e.g., “E-mail” vs. “email”, “url” vs “U.R.L.”, etc.
- Entity names which are synonyms. e.g., “Participant” vs. “Attendee”.
- Entity names which are similar in meaning in a specific domain, e.g., “contribution” vs. “paper” in the conference organization domain, etc.
- Entity names which are represented in abbreviated forms. e.g., “PC Member” vs.“ProgramCommitteeMember”, or acronym, e.g., “WWW” vs “World Wide Web”, etc.
- Entity names which are tokenizable and their tokens (or only a part of tokens) are syntactic or meaning similar. e.g., “hasSurname” vs. “has the last name”, “Camera-ready contribution” vs “Final manuscript”, etc.
- Entity names which are partially syntactically similar. e.g., “email” vs “hasEmail”, “Regular author” vs. “author”, etc.

Table 1 shows different challenges in the ontology alignment process with the examples from the Anatomy dataset (Bodenreider et al. 2005). From the table, we can observe that the first case, i.e., Case-A contains human and mouse anatomy classes/concepts which are partially different. For example, “lunate” from mouse ontology and “Lunate_Bone” from the human ontology is presenting the same body part but their label is partially different syntactically. Another example is “lumbar vertebra 3” and “L3_Vertebra”. In this example “L” is an abbreviation of “lumbar”. In the second case, i.e., Case-B, the ontological classes/concepts are completely different syntactically. Here “cranium” and “skull” are semantically synonymous but syntactically completely different.

Table 1: Difficulties in Ontology Alignment

	Mouse Ontology	Human Ontology
Case-A	Partially Different	
	lunate	Lunate_Bone
	spleen periarteriolar lymphatic sheath	Periarteriolar_Lymphoid _Sheath
	lumbar vertebra 3	L3_Vertebra
	trochlear IV nerve	Trochlear_Nerve
	hair shaft	Shaft_of_the_Hair
Case-B	Completely Different	
	cranium	Skull
	external naris	Nostril

1.3 Motivation

Most of the ontology alignment systems depend on domain knowledge, which makes the alignment process domain-specific and labor-intensive. To address this challenge, we aim at developing an unsupervised ontology alignment approach that is independent of domain knowledge and does not need domain expert intervention. Our proposed approach explores the use of machine learning techniques in ontology linking in an unsupervised manner with no or minimum background knowledge and compares the results with state-of-the-art tools. Our goal is to perform ontology mapping without any domain expert intervention using an unsupervised neural approach utilizing metadata of ontological entities and evaluate the performance of the system in comparison with state-of-the-art systems.

1.4 Research Objective

In our proposed approach, we are trying to address the following research questions.

(i) Can ontology alignment be achieved using unsupervised machine learning instead of the traditional rule-based approaches?

(ii) Can ontology alignment be done independently of domain information and without the need for domain expert intervention?

(iii) Can ontology alignment be improved using the meta-information and structural information of ontologies?

Chapter 2

LITERATURE REVIEW & RELATED WORK

2.1 Ontology Matching Systems

Traditional data integration approaches are mainly applicable to relational data models. There are mainly two approaches used in traditional schema mapping. The first approach is called global-as-view which requires that the global schema must be expressed in terms of the data sources. The second approach is called local-as-view which requires the global schema has to be specified independently from the sources, and the relationships between the global schema and the sources are established by defining every source as a view over the global schema (Doan, Halevy, and Ives 2012)(Lenzerini 2002)(Xin 2015). As relational data models do not have semantic information, the traditional data integration techniques often fail.

The matching process in ontology becomes much more simple compare to traditional data. There has been a lot of researches in the area of ontology matching. Ontology Alignment Evaluation Initiative, i.e., OAEI ("OAEI" 2020) is one such initiative. It is a coordinated international initiative that organizes the evaluation of different ontology matching systems. It is an annual competition that provides a benchmark evaluation for ontology alignment systems. There is a number of different proposed approaches available with very impressive results. Despite the good result each initiative or approach has advantages and disadvantages. In this section, we review the different approaches to ontology alignment. Figure 4 shows the classification of the ontology alignment techniques. All these techniques are complementary to each other, various matching systems use a combination of these techniques.

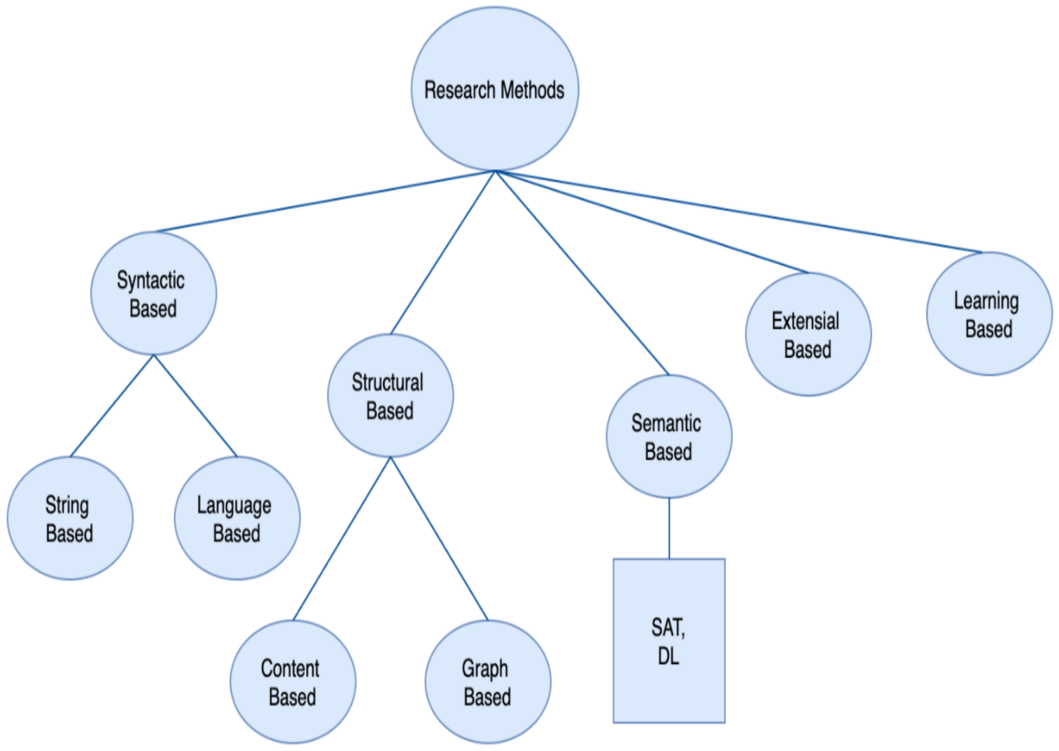


Figure 4: Classification of Ontology Alignment techniques

Syntactic-based approach: The syntactic approach includes a comparison of the names, descriptions, and terms of entities of the source and target ontology. It uses mainly two types of approaches, the first one is the String-based approach and the second one is a Language-based approach. String-based approaches are often used to match names and descriptions of ontology entities. It considers ontological class/concept names as a bag of characters or words. The more similar the strings, the more likely they denote the same entities. It uses different distance metrics to calculate similarity values, such as Hamming Distance, Levenshtein distance, Jaro-Winkler distance, etc. On the other hand, Language-based approaches are generally used as class/concept word pre-processing techniques, such as Normalization, Stemming/Lemmatization, Stop words, Tokenization, etc. Though a number of ontology alignment tools use Syntactic-based approaches, they fail to calculate correct similarity, for example, class/concept names which are synonyms. e.g., “Participant” vs. “Attendee” and class/concept names which are represented in abbreviated forms, i.e., “WWW” vs “World Wide Web”, etc. Limes (Ngomo and Auer 2011) AgreementMakerLight (Faria et al. 2013) and COMA++ (Aumueller et al. 2005) are such tools which use different type of distance metrics for the ontology alignment process.

Structural-based approach: The structural-based approach considers the structural information of the ontology of external linguistic resources like WordNet (Oram 2001) (a machine-readable dictionary) for finding similarity between entities. It applies different metrics on the graph properties of the thesaurus or the ontology, such as wu-palmer metric (Wu and Palmer 1994), Resnik similarity (Resnik 1995), lin similarity (Lin et al. 1998), Jiang-Conrath distance (Jiang and Conrath 1997). Structural-based approaches highly depend on an external Linguistic Resource like WordNet which is not applicable for cross-lingual ontology alignment or domain-specific ontology alignment like biomedical domain.

Semantic-based approach: Logical models, such as propositional satisfiability (SAT) modal or description logic are generally used in Semantic methods. Deduction Techniques of description logic, such as the subsumption, can be used to verify the semantic relations between entities, such as equivalence (similarity is equal to 1), the subsumption (similarity is between 0 and 1), or exclusion (similarity is equal to 0), and therefore used to deduce the similarity between the entities. There are number of tools using semantic-based approaches like SAT, Description Logic, Rule-based inference, Propositional Horn satisfiability etc, such as S-Match (Giunchiglia, Shvaiko, and

Yatskevich 2004), CtxMatch/CtxMatch2 (Serafini et al. 2003), BLOOMS/BLOOMS+ (Pesquita et al. 2010), LogMap/LogMap2 (Jiménez-Ruiz and Grau 2011), Paris (Suchanek, Abiteboul, and Senellart 2011).

Extensional-based approach: These methods utilize individual representations (or instances) of the classes. When two ontologies share the same set of individuals, entity mapping is easier. The easiest way to compare classes when they share instances is to test the intersection of their instance sets A and B. For example, A and B are equal if they contain the same instances, A and B are disjoint if they do not contain any similar instances, B is a hyponym of A if A contains all the instances of B and more, A is a hyponym of B if B contains all the instances of A and more. GLUE (Doan et al. 2003), RiMOM (Li et al. 2008), and ObjectCoref (Hu, Chen, and Qu 2011) are such tools that use methods that infer the similarity between two entities, especially concepts or classes, by analyzing their extensions, i.e., their instances.

Learning-based approach: Some ontology matching systems exploit different supervised machine learning algorithms. Approaches like Context and Inference-based alignER (CIDER) (Gracia, Bernad, and Mena 2011), GLUE (Doan et al. 2003), Yet Another Matcher (YAM++) (Ngo and Bellahsene 2012), DL-Learner (Bühmann et al. 2018) systems use heuristic learning methods. For instance, CIDER uses an artificial neural network for ontology alignment/linking. It is based on a schema-based ontology alignment system that compares each pair of ontology terms through their context and combines them using an artificial neural network. On the other hand, GLUE applies different machine learning techniques on every pair of concepts of source and target ontology to calculate the joint probability distribution of each pair of concepts. YAM++ calculates the similarity metric between the ontological entities. The similarity metric is calculated by machine learning-based combination methods such as Decision Tree, Support Vector Machine, Naive Bayes, and so on. Table 2 shows the comparison between approaches of the state-of-the-art tools. From the table, we can observe that most of the state-of-the-art tools use a combination of multiple approaches.

Table 2: Comparison of Ontology Alignment Tools approaches

Approach	Syntactic Based	Structural Based	Semantic Based	Extensional Based	Learning Based
AgreementMaker -Light	●				
COMA++	●	●			
Limes	●				
S-Match			●		
CtxMatch/ CtxMatch2			●		
BLOOMS/ BLOOMS+			●		
Paris			●		
GLUE		●		●	●
RiMOM				●	
ObjectCoref				●	
YAM++					●
DL-Learner					●
CIDER					●
LogMap/ LogMap2			●		
KARMA	●	●			

Followings are some ontology mapping tools and their approaches.

1. PARIS, i.e., Probabilistic Alignment of Relations, Instances, and Schema (Suchanek, Abiteboul, and Senellart 2011) is a probabilistic model to find ontology alignment. It measures the degree of matching based on probability estimates. The probabilistic model proposed by PARIS includes instances, sub

relations, and Subclasses of classes in source and target ontologies. Say x and y are instances of source ontology and $x\prime$ and $y\prime$ are instances of the target ontology. If x and $x\prime$ are the same or similar instances. The similarity score between y and $y\prime$ depends on relation r. PARIS works on the idea that if r(x, y) and r($x\prime$, $y\prime$) produce the same value or similar value then y and $y\prime$ will be the same or similar instance. The probability of y and $y\prime$ is defined as following equation 2.1.

$$Pr(y \equiv y\prime) = \prod_{r(x,y)} \left(1 - fun(r) \prod_{r(x\prime,y\prime)} (1 - Pr(x \equiv x\prime)) \right) \tag{2.1}$$

2. CIDER, i.e., Context and Inference baseD alignER (Gracia, Bernad, and Mena 2011) uses artificial neural network for ontology matcher. It is based on a schema-based ontology alignment system that compares each pair of ontology terms by their contexts and combines them using an artificial neural network. It has mainly four steps. The first one is ontological context extraction. In this step, the context of each ontology such as synonyms, textual descriptions, hypernyms, hyponyms, properties, domains, roles, associated concepts, etc is extracted. The second part is similarity computation. In this step, the similarities are calculated based on the lexical, taxonomies, and relationships among terms. In the third step, these individual similarities are combined using an artificial neural network. The network is a simple fully connected multi-layer perceptron. It composed of three layers: input, hidden, and output layer. The input layer is the value of an elementary similarity measure. The hidden layer uses a sigmoid function to combine inputs. In the output layer, the resultant similarity will be given.
3. GLUE (Doan et al. 2003) uses taxonomic information for ontology alignment. GLUE system uses machine learning techniques to create a semantic mapping.

It consists of three main modules, Distribution Estimator, Similarity Estimator, and Relaxation Labeler. The first module Distribution Estimator takes input both source and target ontology along with their instances. This module also applies different machine learning techniques to every pair of concepts of source and target ontology. Then it calculates the joint probability distribution of each pair of concepts. Say, the source ontology or taxonomy is S and the target ontology or taxonomy is T. C_1 and C_2 are two different concepts of source and target ontology or taxonomy respectively ($C_1 \subseteq S$ and $C_2 \subseteq T$). Here, it calculates four different parameters P(C_1, C_2), P(C_1, $\overline{\mathrm{C_2}}$), P($\overline{\mathrm{C_1}}$, C_2), and P($\overline{\mathrm{C_1}}$, $\overline{\mathrm{C_2}}$). P(C_1, C_2) is the probability that a randomly chosen instance from the universe belongs to both S and T, P(C_1, $\overline{\mathrm{C_2}}$) is the probability that a randomly chosen instance from the universe belongs to S but not to T, P($\overline{\mathrm{C_1}}$, C_2) is the probability that a randomly chosen instance from the universe belongs to T but not to S, and P($\overline{\mathrm{C_1}}$, $\overline{\mathrm{C_2}}$) is the probability that a randomly chosen instance from the universe that does not belong to S and T. After this the second module is Similarity Estimator uses user-defined similarity function such as Jaccard similarity on each pair of concepts from source and target ontology. At the end of the second module, it creates a similarity matrix between all the pairs of concepts. The last module, Relaxation Labeler uses a similarity matrix, domain-specific knowledge to find the best mapping.

4. RiMOM (Li et al. 2008) A Dynamic Multistrategy Ontology Alignment Framework. It uses Wordnet (a machine processible dictionary) (Oram 2001) to discover semantic similarities in textual descriptions to find out ontology similarity. The process of RiMOM includes Preprocessing, Linguistic-based ontology alignment, Similarity combination, Similarity propagation, Alignment generation and refine-

ment. The first part Preprocessing takes the source and target ontology as input and create description for each entity. In the next Linguistic-based ontology alignment process, multiple linguistic-based strategies such as Edit-Distance-Based Strategy, Vector-Distance (VD)-Based Strategy are executed to obtain a similarity result for each entity pair. The Similarity combination part combines the similarity results. The next Similarity propagation uses three similarity propagation strategies, namely, Concept-to-Concept, Property-to-Property, and Concept-to-Property. The last step is Alignment generation and refinement which fine tunes and outputs the alignment result.

5. COMA++ (Aumueller et al. 2005) is an ontology matching tool which uses string-based similarity. It has mainly three modules Storage, Match Execution, and Mapping Processing. The storage used for importing ontologies, schemas, mappings, and meta information. The next part Match Execution which is the core of COMA++. It runs different matching algorithms and calculates matching results between source and target ontology. It has a large collection of schema matching strategies based on string-based techniques, such as n-gram, edit distance. This module runs the matches, i.e., matching algorithms parallelly and combines the result. The Mapping Processing module is used for enriching mappings and merging ontologies. Apart from these modules, COMA++ has also an efficient graphical user interface which enables the user to upload the source and target ontology and get a pictorial representation of matched entities of source and target ontologies.

6. YAM++, i.e., Yet Another Matcher (Ngo and Bellahsene 2012) combines both semantic and structure-based approach. It discovers mappings between two input ontologies by two matches, the first one is element level matcher and the second is

structural level matcher. Element level matcher extracts annotation information for every entity and computes a similarity score between entities. The similarity metric is calculated by machine learning-based combination methods such as Decision Tree, Support Vector Machine, Naive Bayes, etc. Structural level matcher parsed and transformed the source and target ontology into a graph data structure. Structural level matcher module takes result obtained from the Element level matcher module and runs a similarity propagation process, i.e., Similarity Flooding algorithm. YAM++ also provides a graphical user interface that shows the resulting mappings of two ontologies' alignment.

7. AML or AgreementMaker (Faria et al. 2013) used lexicon-based approach for ontology mapping. It is the most efficient and flexible tool which uses a parallel mechanism to make the entire matching very fast. It consists of three layers. The first module is for similarity computation. In this module, inputs are source and target ontologies and similarity matrices are calculated. The second module is mappings selection. In the second module, the similarity matrix is scanned to find the best mappings having a threshold value. The third layer combines the result of multiple matches from previous layers and produces the final matching or alignment. AML or AgreementMaker uses syntactic, structural, and lexical comparison algorithms along with lexicon like WordNet.
8. LogMap (Jiménez-Ruiz and Grau 2011) is a highly scalable ontology matching system with 'built-in' reasoning and diagnosis capabilities. It can handle both lexically and structurally indexed ontologies. For lexical indexing, it uses WordNet as a knowledge base. On the other hand, for structural index, uses hierarchical structures of the classes/concepts. The main advantage of LogMap or LogMap-2 is its scalability. LogMap is faster on large ontologies in the

biomedical domain. In OAEI largeBio track, it handles Foundational Model of Anatomy ("FMA" 2020), SNOMED CT ("SNOMED" 2020), and the National Cancer Institute Thesaurus ("NCI" 2020) efficiently and scores high accuracy value.

9. LIMES (Ngomo and Auer 2011), the Link Discovery Framework for Metric Spaces, is a framework for discovering links between entities contained in Linked Data sources. It has main eight modules. The main module is the controller module. It orchestrates the entire matching process. It first calls the configuration module which reads the configuration file and takes the source and target ontology as input. The next module is the query module which retrieves instances and properties from the source and target ontology. The Cache module is responsible for storing the information in the cache which will be used in later modules. The rewriter, planner, and engine modules are responsible for Link Specification between source and target ontology. Limes use different machine learning algorithms to identify an appropriate Link Specification between source and target ontology and populate final results.

10. DL-Learner (Bühmann et al. 2018), is a software framework for ontology learning and enrichment. It contains five different modules. The first module is Knowledge sources which contain a different types of data sources. The data sources can be loaded locally or remotely. The second module is Reasoners which performs inference over knowledge sources. It uses OWL API (Horridge and Bechhofer 2011), OWLlink. The next module is Learning problems which is used by different learning algorithms for hypothesis testing. The Learning algorithms are responsible for the core learning strategy. The last module is Refinement Operators which is used by Learning algorithms. DL-Learner uses a supervised

machine learning algorithm. Here domain expert first creates positive and negative data from the knowledge sources which is used to train the machine learning algorithm. Based on the learning, it predicts all the possible similar classes pair with a confidence score.

Although the above-mentioned tools achieved high accuracy, they also have disadvantages. Each of the tools tried to address different challenges in ontology matching but because of so many restrictions they still have many disadvantages. For example, PARIS (Suchanek, Abiteboul, and Senellart 2011) cannot deal with structural heterogeneity. Say, if one ontology is more fine-grained than the another one e.g. in one ontology cities are mentioned as birthplaces and in another ontology, countries are mentioned as birthplace. It is really difficult to come up with a single approach to address all kinds of heterogeneity. We can observe some type of difficulties in other state-of-the-art tools also. To tackle this, many tools tried to compute the different similarities parallelly and combined them to a single similarity value. Tools like GLUE (Doan et al. 2003), AgreementMaker (Faria et al. 2013), YAM++ (Ngo and Bellahsene 2012) are using static mechanism to combine different similarities. On the other hand, 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. CIDER (Gracia, Bernad, and Mena 2011), DL-Learner (Bühmann et al. 2018), GLUE (Doan et al. 2003) are such examples of tools which use machine learning techniques. The problem with these approaches is it highly dependent on the labeled training examples. It needs experienced domain experts to generate labeled examples and depending on the nature of the example, the accuracy of the tool changes. Apart from this, many tools like COMA++ (Aumueller

et al. 2005) considers user involvement should be part of ontology matching but the user involvement introduces human error, inaccuracies. The output of tools like COMA++ depends on the knowledge of the user. Same as user involvement, the use of the knowledge base is also questionable. AgreementMaker (Faria et al. 2013), RiMOM (Li et al. 2008), LogMap (Jiménez-Ruiz and Grau 2011) use different knowledge base such as wordnet (Oram 2001). WordNet is a lexical database for the English language. It only contains commonly used English words. In OAEI most of the ontologies are from a bio-medical domain that contains specific medical domain words which are not commonly used in day-to-day life. It is also difficult for matching multilingual ontologies. For ontologies in different languages, one needs to use different types of converters to convert entity words from another language to English to understand the semantic meaning of the entity. Choosing these converters and a rich knowledge base has also a significant impact on the final result.
All these above-mentioned tools or approaches dependent on user interaction, restricted to metric spaces, labeled training examples, user's decision e.g. threshold value and weightage distribution, etc. For this reason, we are proposing an approach that will consider the entities and their meta-information at the same time and will predict the similarity value without any domain expert supervision. Our aim is to build a generalized approach that can be used by anyone with no or very little domain knowledge on a different type of ontologies.

2.2 Machine Learning Techniques

Currently, a number of ontology matching frameworks or tools are using machine learning techniques in mapping between ontologies. Based on the available data, machine learning techniques or algorithms can be divided into four categories. The first one has supervised machine learning algorithms. Supervised machine learning algorithms use only labeled data. It generates functions that map a relationship between the input data and their label or output. The most widely use supervised machine learning algorithms are Decision Tree, Naive Bayes, Support Vector Machine, and Neural Network. On the other hand, Unsupervised learning algorithms use unlabelled data. The main objective of Unsupervised learning algorithms is to learn features or patterns from the data. It is mainly used for clustering or feature reduction. K-Means Clustering, Principal Component Analysis is such examples of Unsupervised learning algorithms. The third one is Semi-Supervised learning algorithms. The Semi-Supervised learning algorithms combine both Supervised and unsupervised techniques. Semi-Supervised learning algorithms use both labeled and unlabeled examples to generate mapping functions. Generative models are examples of such a Semi-Supervised algorithm. The fourth one is Reinforcement learning. Reinforcement learning algorithms learn all possible ways to achieve a goal in any particular environment. It takes the decision or learns a policy based on its environment and awards the action which achieves the goal with less cost. Reinforcement learning algorithms keep on learning based on the award or feedback.

The neural network is a special type of machine learning technique that is a biologically-inspired approach that enables learning from observed data. Neural networks can be used as supervised and unsupervised in both ways. Neural Network is composed of different layers and each layer is composed of nodes. Nodes are the component where linear or nonlinear functions are fired or executed. A basic neural network has three different layers. The first one is the input layer where data is feed. The second layer is called the hidden layer where the computation happens. The third

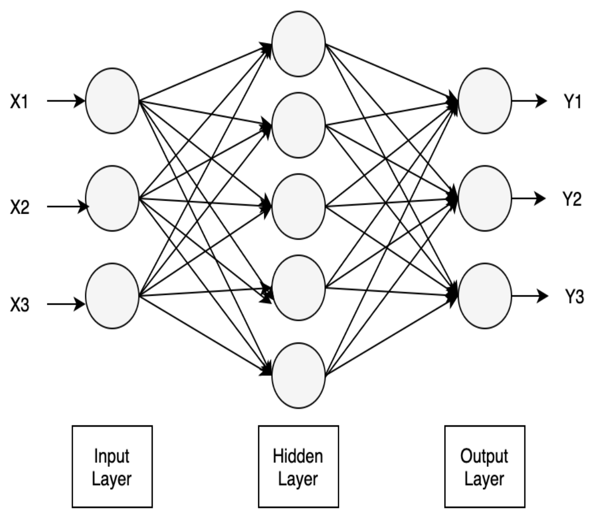


Figure 5: Basic Architecture of Neural Network

layer is the output layer where the output is predicted. Like Perceptron, the data is moved forward in the neural network but additionally, it does the backpropagation which calculates the error at the output layer and does the same computation again and again to minimize the error. There are different types of architectures available in a neural network. Figure 5 shows a basic structure of neural network.

There are many tools and frameworks mentioned in section 2.1 are using different kinds of machine learning techniques. For example, CIDER, i.e., Context and Inference baseD alignER (Gracia, Bernad, and Mena 2011) is using an artificial neural network for mapping/matching between ontologies. The artificial neural network is a network that is a simple fully connected multi-layer perceptron. It takes the value of an elementary similarity measure as input. After that, it uses a sigmoid function to combine all the inputs in the hidden layer and predict the resultant similarity value in the output layer. YAM++, i.e., Yet Another Matcher (Ngo and Bellahsene 2012) is another tool that uses different supervised machine learning methods like Decision Tree, Support Vector Machine, Naive Bayes, etc. DL-Learner (Bühmann et al. 2018) is also a software framework that uses a supervised machine learning algorithm. In this software tool, a domain expert creates positive and negative data from the knowledge base. As this data is labeled, DL-Learner can use it to generate possible mapping pairs with confidence scores. Recently, LogMap and AML both tools proposed an extension to traditional ontology alignment systems that utilize semi-supervised ML solutions which rely on labeled mappings (samples) to learn features and train models to predict mappings. (Chen et al. 2021) is using distant supervision for training, ontology embedding, and Siamese Neural Networks for incorporating richer semantics of the ontology entities.

In our proposed approach, we have proposed a Recursive Neural Network as an unsupervised machine learning technique in Ontology Matching. The main reason we are proposing this neural network structure is to find out whether a software tool can predict the similar entities between two different ontologies without any knowledge base or any domain expert or any labeled data.

The Recursive Neural Network is an extension of a Recurrent Neural Network. In the recurrent Neural Network, only sequential input is considered. Here, the output of the previous step is fed as input to the next step and along with time, it can learn weights and biases. The traditional Recurrent Neural Network uses tanh activation function. The Equation 2.2 shows the traditional Recurrent Neural Network cell equation where W and U are weights applied to the cell at each time step. x_t is the input at time t. h_{t-1} is the output of the previous step. h_t is the output of the previous step.

$$h_t = f(Wx_t + Uh_{t-1}), \text{ where } f \in \{\text{tanh}\} \tag{2.2}$$

The recurrent Neural Network is sequential input that can grow long and during back-propagation, the early words will have very little or no impact on the output. This is known as the vanishing gradient problem Hochreiter (1998) (Hochreiter 1998). Hochreiter and Schmidhuber (1997) proposed Long short-term memory, i.e., LSTM as a Recurrent Neural Network cell which can remember long sequences.

On the other hand, the recursive neural network is a non-linear model which can learn more deeply. The main advantage of this neural network structure is the input can be of arbitrary length and there is no need to do masking or padding. Each training/testing data can be of any length.

Figure 6 shows a very basic structure of the recurrent neural network. This network model can be used for one to one, one to many, many to one, and many to many learning but the data should be linear because the learned weights and biases are propagated with time in a linear fashion. Because of its architecture, it can not allow any other type of data like tree or graph data which is non-linear. For our problem, we need such a neural architecture that can support tree or graph data. It is also useful as in our problem statement each data can hive a different number of meta-information, so the network should be designed in such a way that can accept the variable length of inputs without padding or trimming. Figure 7 shows a very basic architecture of Recursive Neural Network where we will input the meta information of each class/concept.

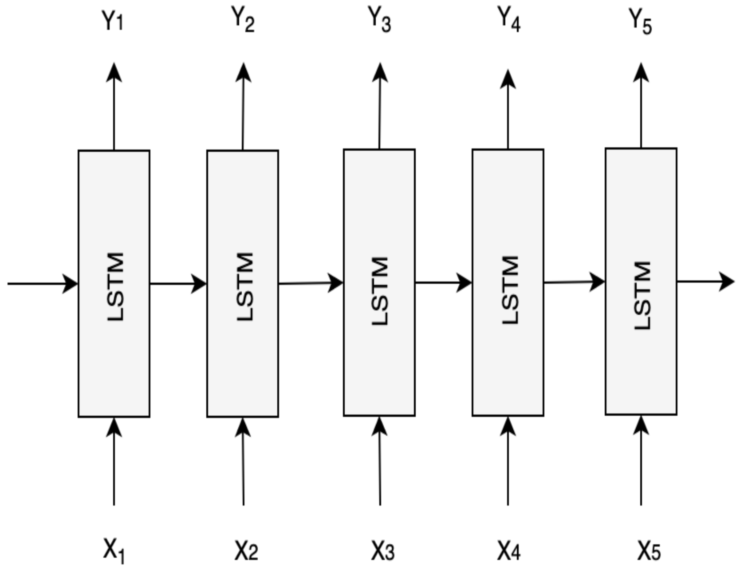


Figure 6: Basic Architecture of Recurrent Neural Network

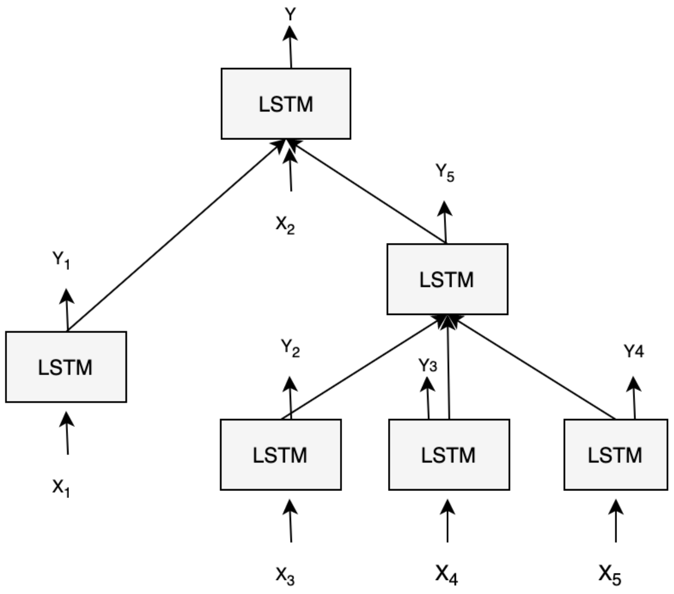


Figure 7: Basic Architecture of Recursive Neural Network

Apart from the Recursive Neural network, we have also explored the graph encoding system. There are many models developed to construct embedding from graph architecture are described below.

(Perozzi, Al-Rfou, and Skiena 2014) proposed DeepWalk, an Unsupervised learning approach, inspired from word2vec in NLP. It learns latent representations of vertices in a network with a random walk approach which encodes social relations in a continuous vector space. Graph autoencoders (GAEs) (Thomas N Kipf and Welling 2016) are unsupervised learning frameworks that encode nodes/graphs into a latent vector space and reconstruct graph data from the encoded information. GAEs are used to learn network embeddings and graph generative distributions. For network embedding, GAEs learn latent node representations through reconstructing graph structural information such as the graph adjacency matrix. For graph generation, some methods generate nodes and edges of a graph step by step while other methods output a graph all at once. Graph Convolutional Network, i.e., GCN (T N Kipf and Welling 2016) is based on the Convolution Neural Networks on non-euclidean data. The most recent one is Graph Attention Network, i.e., GAT (Veličković et al. 2017) is using masked self-attentional layers to address the shortcomings of prior methods based on graph convolutions.

In our experiment, we have used PyTorch-BigGraph (PBG) (Lerer et al. 2019). It borrows several insights from the above mentioned graph embedding techniques. It uses negative sampling and scoring techniques to embed graph nodes or entites. It uses graph partitioning to train arbitrarily large embeddings on either a single machine or in a distributed environment. The main components of PBG are as follows. The *first* one is block decomposition of the adjacency matrix into number of buckets and training on the edges from one bucket at a time. It swaps the embeddings from each partition to reduce memory usage. The *second* component is the distributed execution model that leverages on block decomposition for the large parameter matrices. The *third* component is the efficient negative sampling for nodes uniformly. The *fourth* component is PBG allows multi entity, multi-relation graphs with parameters such as edge weight and relational operator. I our approach, we have used PBG to generate embedding for each node/entity which encapsulates it's neighbor nodes. In our experiment, the neighbor nodes of any node/entity are the meta-information of the node/entity.

2.3 Datasets

To test our approach, two different datasets are used. The first one is the Anatomy dataset which consists of human and mouse anatomy information. The second one is the Library dataset which consists of vocabulary for the economic repository.

2.3.1 Anatomy

The anatomy data set (Bodenreider et al. 2005) published by OAEI. Three different files are provided in the OAEI system: source ontology, target ontology, and result or alignment file[1].

(1) The source ontology (S) is NCIt (National Cancer Institute Thesaurus), which is a reference terminology covering areas of basic and clinical science, built with the goal of facilitating translational research in cancer. It contains 3304 classes.

(2) The target ontology (T) is Adult Mouse Anatomy which contains 2744 classes.

[1]The reference file is available on OAEI: http://oaei.ontologymatching.org/2019/anatomy/index.html

```
...
<Cell>
    <Entity1 RDF:resource="http://mouse.owl#MA_0002401"/>
    <Entity2 RDF:resource="http://human.owl#NCI_C52561"/>
    <Measure RDF:datatype="xsd:float">1.0</Measure>
    <Relation>=</Relation>
</Cell>
...
```

Listing 2.1: A portion of anatomy data set alignment provided by OAEI

(3) The resulting file contains the alignments between the source and target ontology. It consists of all the same or similar concept pairs. Listing 2.1 shows a small portion of anatomy data set alignment, where “MA_0002401” is a class of Adult Mouse Anatomy and “NCI_C52561” is a class of NCIt. The tag “measure” represents the similarity score between these classes. The tag “relation” denotes the relationship between the target ontology class and the source ontology class.

2.3.2 Library

Library (“LIBRARY” 2013) dataset consists two vocabularies, the first one is STW (“STW” 2013) and the second one is TheSoz (“TheSoz” 2013). Both the vocabularies provide a knowledge base of the economic repository.

(1) The STW Thesaurus for Economics provides vocabulary on any economic subject. It contains more than 6000 standardized subject headings and 19000 additional keywords. The vocabulary was developed for indexing purposes in libraries and economic research institutions and includes technical terms used in law, sociology, or politics, and geographic names. It is maintained on a regular basis by the ZBW German National Library of Economics - Leibniz Centre for Economics and published under a CC-by-sa-nc license. It contains labels both in English and German.

```
...
<Cell>
    <Entity1 RDF:resource="http://stw.owl#26103.4"/>
    <Entity2 RDF:resource="http://thesoz.owl#10036756"/>
    <Measure RDF:datatype="xsd:float">1.0</Measure>
    <Relation>=</Relation>
</Cell>
...
```

Listing 2.2: A portion of library data set alignment provided by OAEI

(2) The Thesaurus for the Social Sciences (TheSoz) is a vocabulary that indexes documents and research information in the social sciences. It contains 12000 keywords, from which 8000 are standardized subject headings and 4000 additional keywords. TheSoz also contains labels both in English and German.

In library dataset, the output [2] structure is same as the anatomy dataset. In the following Listing 2.2 the concept “26103.4” from the “STW” ontology is similar as the concept “10036756” from the “TheSoz” ontology.

[2]The reference file is available on OAEI: http://oaei.ontologymatching.org/2013/library/index.html

Chapter 3

ONTOLOGY ALIGNMENT WITH RECURSIVE NEURAL NETWORK

In the following section, the recursive neural network approach in the OntoConnect system is described. The recursive neural network is an extension of a recurrent neural network. The main reason behind choosing the recursive neural network is it can accept graph or tree-based non-linear input instead of a sequential input. As the ontology class/concept meta-information is of arbitrary length, the recursive neural network can process the variable length input sequence via the recursive application.

3.1 Proposed Recursive Neural Network Methodology

The overall *OntoConnect* alignment process with recursive neural approach can be divided into two main processes. The first is training the *OntoConnect* model with source ontology classes/concepts. The second is predicting similar source classes/concepts for each target class/concept.

The learning/training of *OntoConnect* model is described in Algorithm 1. Our strategy is to train the recursive neural model in a stochastic manner with each meta-information of source classes/concepts. The first step is to collect the meta-information for each source class/concept. The function "getMetaInformation" is responsible for collecting the meta-information for a class from the ontology. The function "getVector" retrieves the word-vector for each source class/concept from the pre-trained FastText (Bojanowski et al. 2017). The function "trainModel" trains the recursive model and updates the weights and biases in a stochastic manner.

```
Algorithm 1: OntoConnect Learning algorithm
  Input: set of concepts in source ontology $sConcepts$
  Output: Trained OntoConnect model $onto\_model$
1 $onto\_model \leftarrow \emptyset$
2 foreach source concept $s_i \in sConcepts$ do
      /* Retrive vector from pre-trained model FastText for the source
         concept $s_i$                                                  */
3     $Vec_{s_i} \leftarrow getVector(s_i)$
      /* Retrive meta-information (parent, child, equivalent, disjoint,
         restriction) of the source concept $s_i$ from the source ontology.  */
4     $Meta_{s_i} \leftarrow getMetaInformation(s_i)$
      /* Retrive vector from pre-trained model FastText for each
         meta-information of the source concept $s_i$                  */
5     $MetaVec_{s_i} \leftarrow getVector(Meta_{s_i})$
      /* Training OntoConnect model for the source concept $s_i$ stochastically
         */
6     $onto\_model \leftarrow trainModel(Vec_{s_i}, MetaVec_{s_i}, onto\_model)$
7 return $onto\_model$
```

Algorithm 2 shows the algorithm for the prediction of the target classes/concepts. The "getMetaInformation" and "getVector" retrieve the meta-information and word-vector for a class/concept for both source and target ontology. For a target class/concept, we calculate similarity for each source ontology class/concept. The word similarity ($Word_{sim_{i,j}}$) is the cosine similarity between the target and source ontology class/concept word-vectors retrieved from the pre-trained FastText model. On the other hand, meta-similarity ($Meta_{sim_{i,j}}$) is the cosine similarity between source ontology class/concept word-vector and the predicted vector from the *OntoConnect* model for a target ontology class/concept. Next, a weighted combined similarity of the word-similarity ($Word_{sim_{i,j}}$) and meta-similarity ($Meta_{sim_{i,j}}$) is calculated for each pair of the target and source ontology classes/concepts. We are considering the source ontology class/concept as a similar class/concept with the maximum combined similarity (Cmb_{sim_i}). The correspondence list is updated with the most similar class/concept pair having the highest similarity score. In the end, it returns the correspondence list.

Algorithm 2: *OntoConnect* Prediction algorithm

```
Input: set of concepts in source ontology $sConcepts$, set of concepts in
       target ontology $tConcepts$, pre-trained $OntoConnect$ model
Output: List of Correspondences $correspondences$
1  $correspondences \leftarrow \emptyset$
2  foreach $target\ concept\ t_i \in tConcepts$ do
       /* Retrive vector from pre-trained model FastText for the target
          concept $t_i$                                                        */
3      $Vec_{t_i} \leftarrow getVector(t_i)$
       /* Retrive meta-information (parent, child, equivalent, disjoint,
          restriction) of the target concept $t_i$ from the target ontology.   */
4      $Meta_{t_i} \leftarrow getMetaInformation(t_i)$
       /* Retrive vector from pre-trained model FastText for each
          meta-information of the target concept $t_i$                         */
5      $MetaVec_{t_i} \leftarrow getVector(Meta_{t_i})$
       /* Retrive vector from pre-trained model OntoConnect from the
          meta-information of the target concept $t_i$                        */
6      $PredVec_{t_i} \leftarrow predVector(MetaVec_{t_i}, OntoConnect\ model)$
7      $Cmb_{sim_i} \leftarrow \emptyset$
8      $correspondence_i \leftarrow \emptyset$
9      foreach $source\ concept\ s_j \in sConcepts$ do
           /* Retrive vector from pre-trained model FastText for the source
              concept $s_j$                                                    */
10         $Vec_{s_j} \leftarrow getVector(s_j)$
           /* Compute cosine-similarity between $Vec_{t_i}, Vec_{s_j}$         */
11         $Word_{sim_{i,j}} \leftarrow computeSimilarity(Vec_{t_i}, Vec_{s_j})$
           /* Compute cosine-similarity between $PredVec_{t_i}, Vec_{s_j}$     */
12         $Meta_{sim_{i,j}} \leftarrow computeSimilarity(PredVec_{t_i}, Vec_{s_j})$
13         $TmpCmb_{sim_{i,j}} \leftarrow$
             $weightedHarmonicMean(Word_{sim_{i,j}}, Meta_{sim_{i,j}})$
           /* Updating $correspondence_i$ based on combined similarity         */
14         if $TmpCmb_{sim_{i,j}} > Cmb_{sim_i}$ then
15             $Cmb_{sim_i} \leftarrow TmpCmb_{sim_{i,j}}$
                 $correspondence_i \leftarrow [t_i, s_j, Cmb_{sim_i}]$
       /* Updating $correspondences$ list                                      */
16     $correspondences \leftarrow correspondences + correspondence_i$
17 return $correspondences$
```

In the following section, we discuss each step in detail for both the training/learning

of the *OntoConnect* model and predicting similar class/concept by using the trained *OntoConnect* model.
Our proposed Ontology Alignment system consists of two main tasks: the first task is unsupervised learning of the model followed by the prediction of similar classes/concepts using the trained model.

Figure 8 represents a workflow of the proposed *OntoConnect* ontology alignment system. The left side of Figure 8 shows the learning/training of the OntoConnect - Recursive Neural Network. It starts with the source ontology. The first step is data preparation. In this step, OWL API (Horridge and Bechhofer 2011) is used to extract the meta-information of each class/concept from the source ontology. In the next step of Data Preprocessing, different techniques like lemmatization, stop-word removal are applied to the labels of the extracted class/concept. After the Data Preprocessing step, pre-trained model fastText is used to generate a vector from each class/concept which will be fed to the proposed Recursive Neural Network. After the training phase, the trained model is saved for the prediction of correspondences.

The right side of Figure 8 shows the prediction phase of the OntoConnect - Recursive Neural Network. Same as the Training phase, the target ontology classes/concepts are preprocessed and the vectors are generated for each target ontology class/concept. In this phase, we have calculated both word-similarity and meta-similarity for each pair of classes/concepts of source and target ontologies. The correspondence result list is created based on the calculated similarity value.

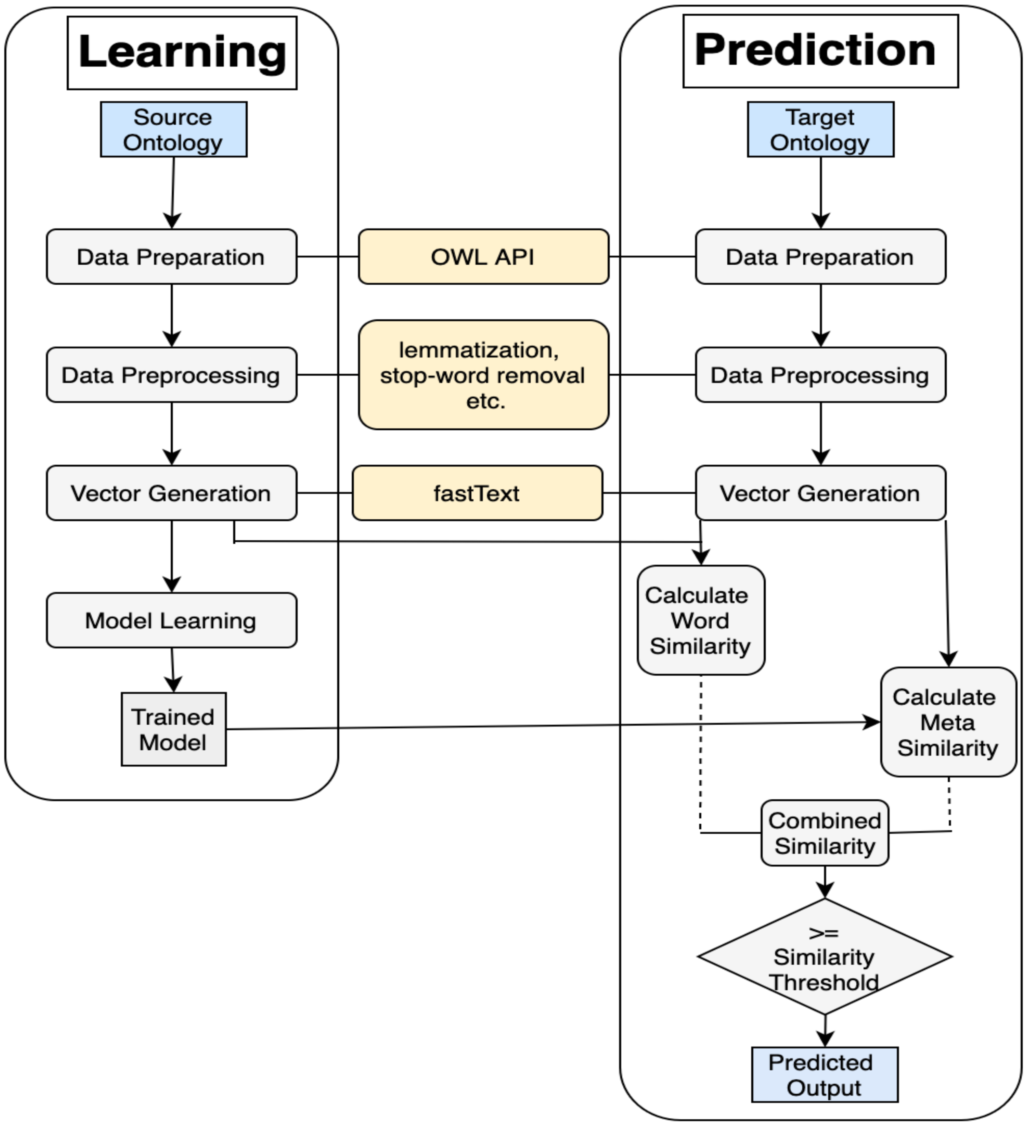


Figure 8: Project Flow of *OntoConnect* - Recursive Neural Network

3.2 Data Preparation

In the **first step**, the source and target ontology are parsed and converted into an Ontology Matching model. For parsing the source ontology, a Java API named OWL API (Horridge and Bechhofer 2011) and HermiT Reasoner (Motik, Shearer, and Horrocks 2007) are used to extract meta information of a class, such as IRI of the class (e_iri), label of the class (e_lbl), restriction of the class (rc), parent (pc), child (cc), equivalent (ec), and disjoint classes (dc) of each class/concept of the source ontology. OWL API is a high-level API to work with OWL ontology, it supports parsing, reading, reasoning, manipulating, validating, and rendering of OWL ontology files. HermiT is a reasoning engine integrated into OWL API which can determine or identify the subsumption relationships between classes/concepts.

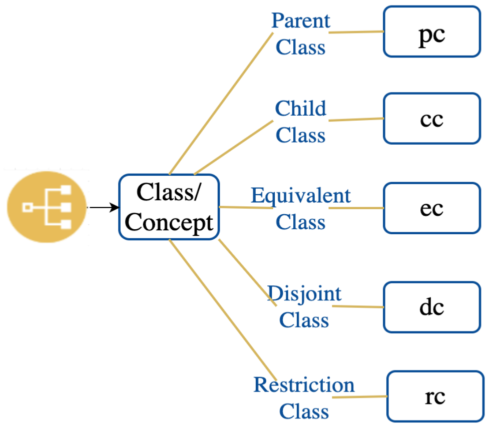


Figure 9: Meta information of a Source/Target Ontology Class/Concept

Next in the **second step**, different pre-processing techniques are applied on the class/concept labels such as Normalization, Lemmatization, Stop word removal, Tokenization, etc. Following are the pre-processing techniques that are used in the OntoConnect system.

- **Normalization**

  Normalization is to eliminate stylistic differences between strings as much as possible. This generally involves putting all characters into either upper or lower case, replacing punctuation characters with space, and standardizing word order, often by alphabetizing the words within the string. Normalization might also involve transliterating characters, not in the Latin alphabet to their closest equivalent.

- **Lemmatization**

  Lemmatization is used to eliminate grammatical differences between words due to verb tense, plurals, and other word forms by finding the root of each word in the string. However, the two words differ in their flavor. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. If confronted with the token "saw", stemming might return just "s", whereas lemmatization would attempt to return either see or saw depending on whether the use of the token was as a verb or a noun.

- **Stop words**

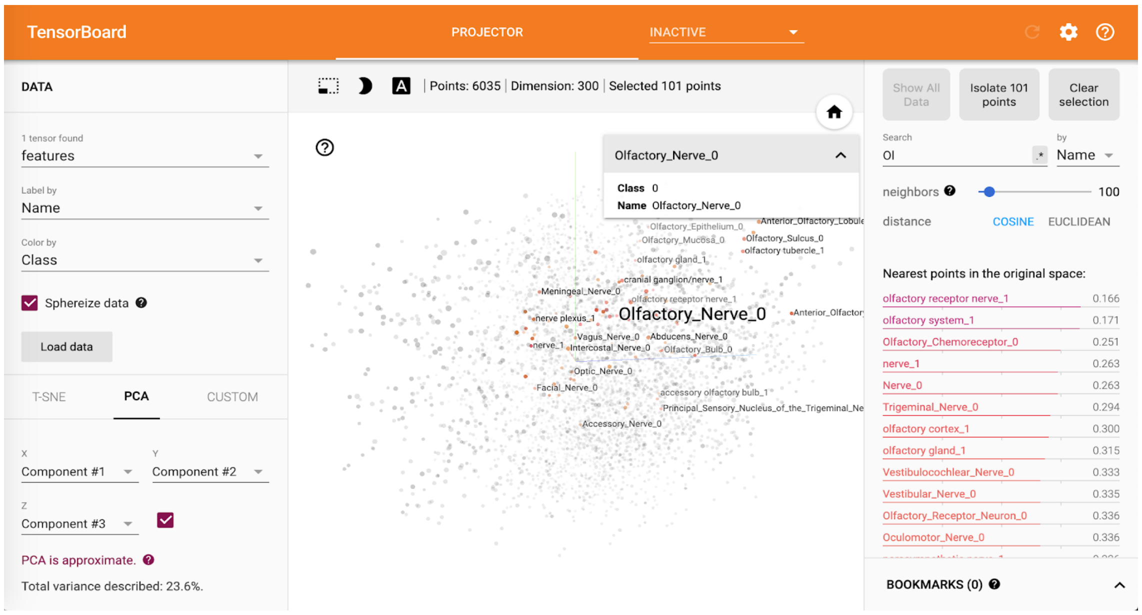
  Stop words are the most commonly used words in a language. The idea behind removing stop words from strings prior to computing their similarity is that very common words add little useful information. There are many lists of stop words available for different languages.

- **Tokenization**

  Tokenization involves splitting strings into their component words. Word boundaries vary based on implementation, but often some combination of whitespace, underscores, hyphens, slashes, and lower-to-uppercase changes (to detect camel-

Case) is used. Tokenization is useful when comparing ontologies with different naming conventions, such as underscores versus hyphens to delineate words. This is particularly important for set-based string similarity metrics.

In the **last step**, A vector form of each class/concept is generated. In the OntoConnect system, a number of pre-trained models are used. The first word embedding technique is Word2vec is a family of (Mikolov et al. 2013). Word2vec is a family of model architectures and optimizations that used to learn word embeddings from large datasets. In Word2vec, there are two novel model architectures that use to compute continuous vector representations of words from a very large corpus. The two model architectures are the continuous bag-of-words model (CBOW) and the continuous skip-gram model. In the continuous bag-of-words model, the current word is predicted from a window of surrounding context words on the other hand, in the continuous skip-gram model, the model predicts the surrounding window of context words for the current word. Apart from Word2vec, GloVe (Pennington, Socher, and Manning 2014) is another technique to obtain the word embedding. GloVe or Global Vectors for Word Representation uses training on aggregated global word-word co-occurrence statistics from a corpus. It encodes the co-occurrence probability ratio between two words. In our experiment, we have also explored BERT or Bidirectional Encoder Representations from Transformer (Devlin et al. 2018). BERT is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weights between them are dynamically calculated based upon their connection. It is designed to read or encode sentences from both directions left-to-right and right-to-left. The main difference between BERT and Word2vec is BERT can produce different word representations for the same word in different sentences. For example, given two sentences: "The man was accused of robbing a bank." and "The man went fishing by the bank of the river." In both sentences, the Word2vec produces the same word embedding for the word "bank" while BERT generates different embeddings as the context is different for the word "bank" in those two sentences. Compared to these pre-trained embedding models, another pre-trained embedding model fastText (Bojanowski et al. 2017) is used in the OntoConnect system. fastText is developed by Facebook's AI Research (FAIR) lab is used. This embedding model uses the skip-gram technique, where each word is considered as a bag of n-gram characters. It learns the representations for character n-grams and represents the words as the sum of the n-gram vectors. The main advantage of the model provided by fastText is that it can generate a meaningful vector for a word that is not present in its dictionary. In particular, we used a model trained on Wikipedia 2017, UMBC WebBase corpus, and statmt.org news data set, which generates a 300-dimension vector for each word. The main reason behind using fastText is that it can generate meaningful vectors of the ontology class/concept even when the word is not present in the model. The Figure 10 shows three-dimension representation of the ontological entities.



Points: 6035 | Dimension: 300 | Selected 101 points

1 tensor found: features

Label by: Name

Color by: Class

[✓] Sphereize data

X: Component #1 Y: Component #2

Z: Component #3 [✓]

PCA is approximate.

Total variance described: 23.6%.

Olfactory_Nerve_0

Class 0

Name Olfactory_Nerve_0

Search: Ol by: Name

neighbors: 100

distance: COSINE EUCLIDEAN

Nearest points in the original space:

olfactory receptor nerve_1	0.166
olfactory system_1	0.171
Olfactory_Chemoreceptor_0	0.251
nerve_1	0.263
Nerve_0	0.263
Trigeminal_Nerve_0	0.294
olfactory cortex_1	0.300
olfactory gland_1	0.315
Vestibulocochlear_Nerve_0	0.333
Vestibular_Nerve_0	0.335
Olfactory_Receptor_Neuron_0	0.336
Oculomotor_Nerve_0	0.336

Figure 10: Vector (mean) visual representation of all Ontological Entities

3.3 Recursive Neural Network Learning

In this step, different contexts of classes/concepts will be used to make its vector representation semantically richer. From the Ontology Matching model, we know that a class/concept can have meta-information/attributes like Parent Classes, Child Classes, Equivalent Classes, Disjoint Classes, Restrictions.

The left side of the Figure 11 shows an example of a training data which will be used later in training Recursive Neural Network. Let's assume the concept/class is "h_1". "h_1" has meta information "parent class", "child class", "equivalent class", "disjoint class" and "restriction class". The parent class of "h_1" is pc_1h_1. cc_1h_1, cc_2h_1, ec_1h_1, dc_1h_1, $rc_1'h_1$, $rc_1''h_1$ are child classes, equivalent class, disjoint class and restriction classes of h_1 respectively. The right side of the Figure 11 shows the high level representation of the learning process where the input is the meta-information of the source ontology class h_1 and the loss is calculated against the source ontology class h_1 itself.

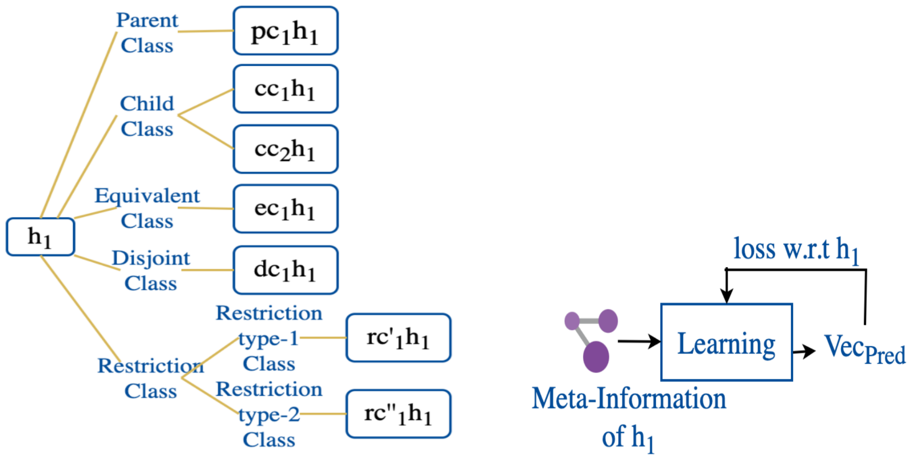


Figure 11: Meta information of a sample source ontology class and High level representation of Model learning process in Recursive Neural Approach

In our experiment, the recursive neural network (Chinea 2009) is used as an unsupervised machine learning framework for the proposed ontology alignment. The recursive neural network is an extension of a recurrent neural network that supports a non-linear neural network model. The drawback of the recurrent neural network's architecture is that it does not work well with non-linear data structures such as trees or graphs. Since ontology is structured as a tree, it is required for us to use a neural architecture that can support non-linear data. In addition to that, the recursive neural network input can be of arbitrary length and there is no need to mask or pad the input data.

In our proposed approach, long short-term memory (LSTM) is used as recursive neural network cell. Hochreiter and Schmidhuber (Hochreiter 1998) proposed long short term memory (LSTM) which can remember long sequences. Equation 3.1 shows the weight and bias calculation of an LSTM cell (Hochreiter and Schmidhuber 1997) which has three main gates: the first one is input gate i_t, the second one is forget gate f_t and the third one is output gate o_t. Apart from the gates, it has cell state c_t and hidden state h_t. In this equation, x_t is the vector input to cell at current time-step t. $W^{(i)}$, $U^{(i)}$, $b^{(i)}$ are the weights and biases for input gate. In the same way, $W^{(f)}$, $U^{(f)}$, $b^{(f)}$ are the weights and biases for forget gate and $W^{(o)}$, $U^{(o)}$, $b^{(o)}$ are the weights and biases for output gate. In our case, we used the sigmoid (σ) and *tanh* functions as activation functions. In the following equations, $\odot$ signifies the multiplication or the dot product.

$$\begin{aligned}
i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)});\\
f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)});\\
o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)});\\
u_t &= \sigma(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)});\\
c_t &= i_t \odot u_t + f_t \odot c_{t-1};\\
h_t &= o_t \odot \tanh(c_t)
\end{aligned} \tag{3.1}$$

A class/concept in the ontology can have an arbitrary number of meta information. For example, it is not necessary for a class/concept, to have a single-parent class. It may have multiple parent classes or no parent class at all. The same is true for other meta information such as child class, equivalent class, disjoint class, and restrictions. To support this arbitrary and uncertain number of child nodes for a tree node (class/concept), the system dynamically creates an LSTM cell depending on the number of child nodes. On every level, the output vectors from each LSTM cell are averaged and fed to the upper level. In the last level, the output vectors from all meta information are also averaged and used for loss calculation and back-propagation.

This dynamically generated LSTM tree model is termed a dynamic-array tree-LSTM model.

Figure 12 shows the general architecture of the Recursive neural network (dynamic-array tree-LSTM model) in the ontology alignment system. In the figure, $pc_1...pc_m$ denote the parent classes of a class/concept. Similarly, $cc_1...cc_n$, ec_1, dc_1, $rc'_1...rc'_s...rc''_1...rc''_t$ are child classes, equivalent class, disjoint class and restriction classes of a class/concept. $X(pc_1)$ is the vector representation of pc_1 obtained from pre-trained model fastText. $c(pc_1)$ is the cell state and $h(pc_1)$ is the hidden state of the LSTM cell for parent meta information. At the output level, the model generates a class/concept vector with same dimension of the input meta information vector dimension.

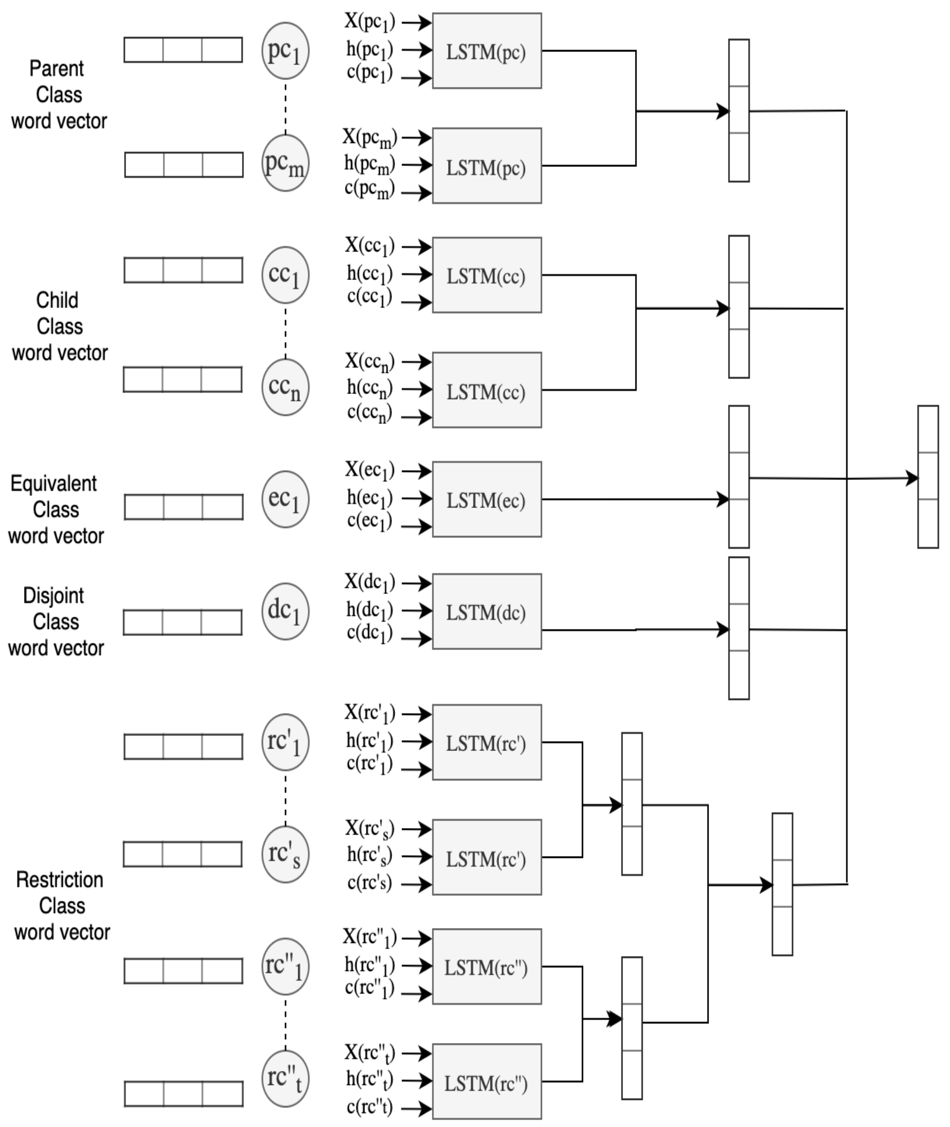


Figure 12: Ontology Matching training of Recursive Neural Network

3.4 Correspondence Prediction

The source (S) and the target (T) ontologies, both are used as inputs in model prediction. Similar to the training process, the ontologies are parsed, pre-processed and the corresponding class/concept vectors are generated.

The left part of the Figure 13 shows an example of a testing data. Let's assume the concept/class is "m_1". "m_1" has meta information "parent class", "child class", "equivalent class", "disjoint class" and "restriction class". The parent class of "m_1" is pc_1m_1. cc_1m_1, cc_2m_1, ec_1m_1, dc_1m_1, rc'_1m_1 is child class, equivalent class, disjoint class and restriction classes of m_1 respectively. The right part of the Figure 13 shows the high level representation of the predicting process where the input is the meta-information of the target ontology class m_1.

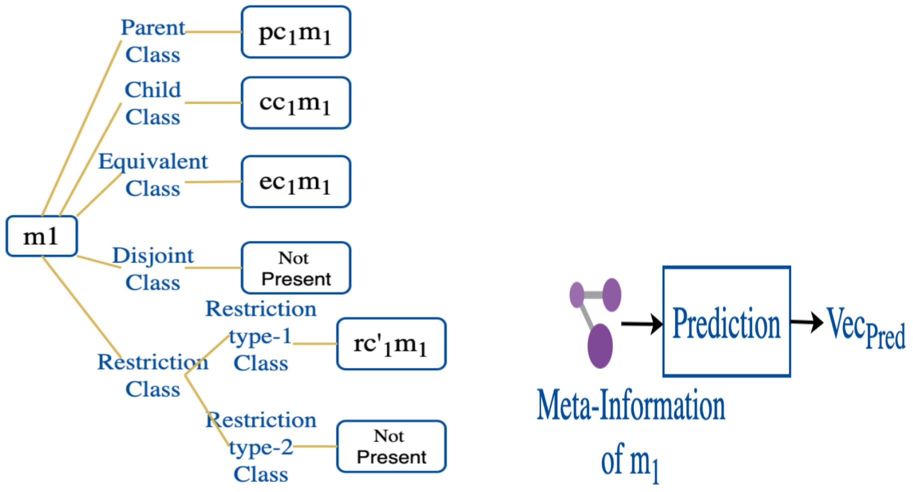


Figure 13: Meta information of a sample target ontology class and High level representation of Model predicting process in Recursive Neural Approach

The steps for the calculation of the combined similarity between the source and target ontology classes/concepts are as follows. (1) After populating the source and target class/concept vectors, the word similarity is calculated. The word similarity ($Word_{sim}$) is the cosine distance between the source and target class/concept vectors. Equation 3.2 shows the cosine similarity measurement between two vectors $\vec{s}$ and $\vec{t}$, which is the cosine of the angle projected in a multi-dimensional (d) space. In our experiment, the time complexity of this step is $O(mn)$, where m is the number of source ontology classes and n is the number of target ontology classes.

$$\begin{aligned}\cos(\overrightarrow{s}, \overrightarrow{t}) &= \frac{\overrightarrow{s} \cdot \overrightarrow{t}}{\|\overrightarrow{s}\|\|\overrightarrow{t}\|} \\ &= \frac{\sum_{i=1}^{d} \mathbf{s}_i \mathbf{t}_i}{\sqrt{\sum_{i=1}^{d} (\mathbf{s}_i)^2} \sqrt{\sum_{i=1}^{d} (\mathbf{t}_i)^2}}\end{aligned} \tag{3.2}$$

(2) In the next step, the trained ontology alignment model is used to calculate the meta similarity ($Meta_{sim}$). The input to the model is the meta-information of a target ontology class and it predicts a vector that is similar to one of the source classes. We have used the cosine similarity to measure the meta similarity as well.

(3) A combined similarity is derived from the word similarity ($Word_{sim}$) and meta similarity ($Meta_{sim}$). The combined similarity value (Cmb_{sim}) is calculated by obtaining the weighted harmonic mean of these two similarity values as shown in Equation 3.3, where α and β represent weights of word and meta similarity values respectively. In our study, a value (0.5) is fixed for both α and β to give equal weight to both word and meta similarity.

$$Cmb_{sim} = \frac{\alpha + \beta}{(\frac{\alpha}{Word_{sim}} + \frac{\beta}{Meta_{sim}})} \tag{3.3}$$

At the end of this process, we have an output with a set of alignments between source and target ontological classes/concepts with respective combined similarity values which range from 0 to 1. A threshold is considered, referred to as ‘similarity threshold’, to filter out predictions with significant similarity values only.

3.5 Experimental Study

3.5.1 Evaluation Methodology

For evaluation, we have done intrinsic evaluation. In intrinsic evaluation, automatically computed similar concepts will be compared with gold standards.

In order to evaluate the performance of the ontology matching system, we have used standard precision and recall measurement mentioned in (Crestani and Rijsbergen 1995). Given two ontologies where O is the source ontology and O' is the target ontology, an alignment between these two ontologies is a set of correspondences, i.e., $< e, e', r, n >$. Here, e is an entity from source ontology, i.e., $e \subseteq O$ and e' is an entity from target ontology, i.e., $e \subseteq O'$. r is the relationship between e and e'. In our project relationship will be equivalence, i.e., =. n the similarity value or confidence value [0..1] of the relationship r between e and e'. The output alignment of the ontology matching process is denoted by A. The gold copy of the reference alignment is denoted by R.

Given the reference alignment R, the precision of alignment A is given by equation 3.4. Here $|R \cap A|$ is the number of returned correct correspondences from the ontology matching system and $|A|$ is the number of returned correspondences from the ontology matching system.

$$P(A, R) = \frac{|R \cap A|}{|A|} \tag{3.4}$$

The recall of alignment R and A is given by equation 3.5. Here $|R \cap A|$ is the number of returned correct correspondences from the ontology matching system and $|R|$ is the number of existing correspondences in the reference alignment.

$$R(A,R) = \frac{|R \cap A|}{|R|} \tag{3.5}$$

We will also evaluate the performance of the ontology system by calculating the F-measure which is the harmonic mean of the precision and recall mentioned in the equation 3.4 and 3.5. F-measure is denoted by equation 3.6 where P(A,R), R(A,R) are the precision and recall of the ontology matching system respectively.

$$F(A,R) = \frac{2 * P(A,R) * R(A,R)}{P(A,R) + R(A,R)} \tag{3.6}$$

3.5.2 Results & Findings on Anatomy Dataset

The proposed recursive neural network was tested on Anatomy dataset ("Anatomy" 2013) with various parameters ,i.e., similarity threshold value and class/concept word vector dimension. The precision, recall, and F-measure values for each combination of parameters are reported in the following Tables 3, 4, 5 for top-k predictions of target class/concept where k=1,3,5 with 100, 200, 300 dimension class/concept word-vector respectively.

Table 3 shows the precision increases and the recall decreases with the increase of the similarity threshold. We can observe in the Table 3 with 200 dimension and 300 dimension class/concept vector the best result produced by the OntoConnect system is 80.36% for top-1 prediction with similarity threshold 0.96. We can observe that the OntoConnect system with Recursive Neural Network exhibits the same result pattern for 100, 200, and 300 dimension entity word-vector.

Table 3: Experimental results of OntoConnect Ontology Alignment system - Recursive Neural Network for varying similarity thresholds: Precision, Recall, and F-measure values for class/concept vector dimensions = 100 and number of predictions k = 1,3,5 on Anatomy dataset

		100-dimension		
Number of Prediction	Similarity Threshold	Precision	Recall	F-measure
Top-1	0.99	98.14	66.87	79.54
	0.98	97.96	67.33	79.81
	0.97	96.85	67.87	79.81
	0.96	95.66	69.14	**80.26**
	0.95	94.05	69.74	80.09
	0.94	91.11	70.54	79.52
	0.93	88.72	72.01	79.50
	0.92	85.90	72.41	78.58
	0.91	83.19	73.08	77.81
	0.90	81.12	73.75	77.26
Top-3	0.99	98.43	67.07	79.78
	0.98	98.06	67.40	79.78
	0.97	97.14	68.07	80.05
	0.96	95.93	69.34	80.50
	0.95	94.59	70.14	**80.49**
	0.94	91.72	71.01	80.05
	0.93	89.38	72.55	80.09
	0.92	86.85	73.21	79.45
	0.91	84.18	73.95	78.73
	0.90	82.44	74.95	78.52
Top-5	0.99	98.43	67.07	79.78
	0.98	98.06	67.40	79.89
	0.97	97.14	68.07	80.05
	0.96	95.93	69.34	80.50
	0.95	94.59	70.14	**80.49**
	0.94	91.72	71.01	80.05
	0.93	89.38	72.55	80.09
	0.92	86.93	73.28	79.52
	0.91	84.33	74.08	78.88
	0.90	82.59	75.08	78.66

Table 4: Experimental results of OntoConnect Ontology Alignment system - Recursive Neural Network for varying similarity thresholds: Precision, Recall, and F-measure values for class/concept vector dimensions = 200 and number of predictions k = 1,3,5 on Anatomy dataset

		200-dimension		
Number of Prediction	Similarity Threshold	Precision	Recall	F-measure
Top-1	0.99	98.33	66.93	79.65
	0.98	98.15	67.20	79.78
	0.97	97.87	67.60	79.97
	0.96	96.70	68.47	80.17
	0.95	96.00	68.47	80.20
	0.94	95.15	69.41	**80.36**
	0.93	92.90	69.94	79.80
	0.92	90.75	71.41	79.93
	0.91	88.39	72.21	79.49
	0.90	86.83	73.08	79.36
Top-3	0.99	98.43	67.00	79.73
	0.98	98.24	67.27	79.86
	0.97	97.97	67.67	80.05
	0.96	96.79	68.54	80.25
	0.95	96.18	69.00	80.36
	0.94	95.42	69.61	**80.55**
	0.93	93.17	70.14	80.03
	0.92	91.00	71.61	80.15
	0.91	88.80	72.55	79.85
	0.90	87.38	73.55	79.87
Top-5	0.99	98.43	67.00	79.73
	0.98	98.24	67.27	79.86
	0.97	97.97	67.67	80.05
	0.96	96.79	68.54	80.25
	0.95	96.18	69.00	80.36
	0.94	95.42	69.61	**80.55**
	0.93	93.17	70.14	80.03
	0.92	91.00	71.61	80.15
	0.91	88.80	72.55	79.85
	0.90	87.38	73.55	79.87

Table 5: Experimental results of OntoConnect Ontology Alignment system - Recursive Neural Network for varying similarity thresholds: Precision, Recall, and F-measure values for class/concept vector dimensions = 300 and number of predictions k = 1,3,5 on Anatomy dataset

		300-dimension		
Number of Prediction	Similarity Threshold	Precision	Recall	F-measure
Top-1	0.99	98.33	66.93	79.65
	0.98	98.24	66.93	79.62
	0.97	98.16	67.67	80.11
	0.96	97.71	68.27	**80.36**
	0.95	96.44	68.74	80.27
	0.94	95.51	69.61	80.53
	0.93	94.17	70.07	80.35
	0.92	92.93	71.14	80.59
	0.91	90.22	72.08	80.13
	0.90	88.10	72.68	79.65
Top-3	0.99	98.43	67.00	79.73
	0.98	98.33	67.00	79.70
	0.97	98.26	67.74	80.19
	0.96	97.80	68.34	80.46
	0.95	96.53	68.80	80.34
	0.94	95.69	69.74	80.68
	0.93	94.34	70.21	80.51
	0.92	93.11	71.28	**80.74**
	0.91	90.47	72.28	80.36
	0.90	88.50	73.01	80.01
Top-5	0.99	98.43	67.00	79.73
	0.98	98.33	67.00	79.70
	0.97	98.26	67.74	80.19
	0.96	97.80	68.34	80.46
	0.95	96.53	68.80	80.34
	0.94	95.69	69.74	80.68
	0.93	94.34	70.21	80.51
	0.92	93.11	71.28	**80.74**
	0.91	90.47	72.28	80.36
	0.90	88.50	73.01	79.86

3.5.3 Result Analysis on Anatomy Dataset

The average precision, recall, and F-measure across all the class/concept vector dimensions (100, 200, 300) and the number of target class/concept predictions (k=1,3,5) are presented in Figure 14. It shows the change in precision, recall, and F-measure of the ontology alignment system with different similarity values. It can be observed that the precision increases whereas the recall decrease with the increase in the similarity threshold value. This is expected as with a higher similarity threshold, the ontology alignment system returns a lower number of correspondences compared to the number of existing correspondences in the reference alignment. From the result analysis, it can be noted that with a 0.96 similarity threshold value, the proposed ontology alignment system - Recursive Neural Approach performs best with the highest average F-measure value of 80.36.

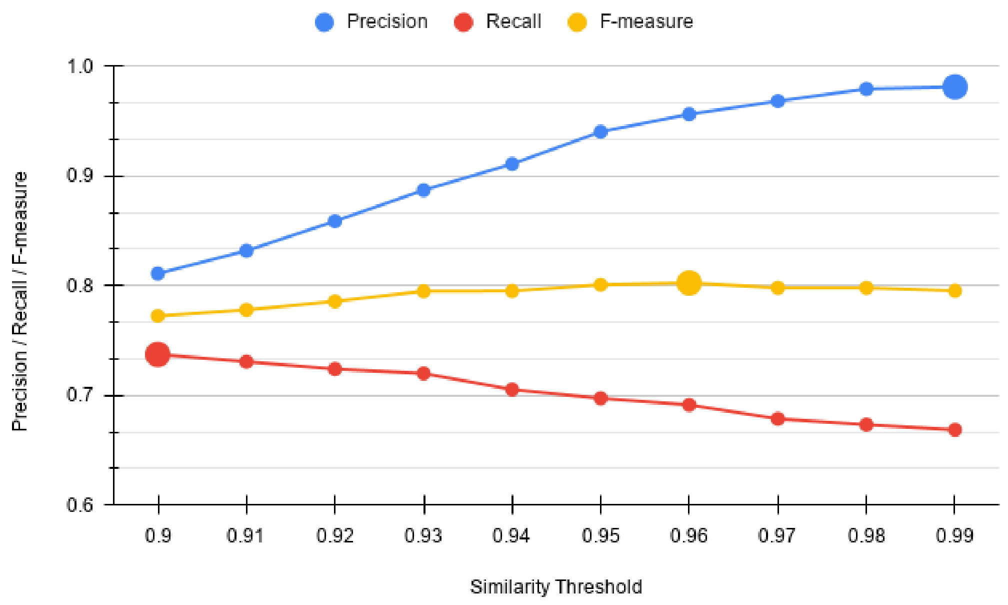


Figure 14: Change in evaluation metric values (precision, recall, F-measure) of Ontology alignment system - Recursive Neural Network with increasing similarity threshold values on Anatomy dataset

Similarly, Figure 15 shows the change in the average precision, recall, and F-measure values with different lengths of class/concept vector dimensions across all the similarity threshold values and the top-k (=1,2,3) predicted target classes/concepts. It can be observed that only a minuscule change in the performance of the ontology alignment system occurs with the increase of the class/concept vector dimensions. Thus, it can be noted that the performance of the proposed system stays almost invariant towards the change in class/concept vector length. We can also observe the change of performance (F-measure) of the OntoConnect system is very small. The highest F-measure with 100 dimension class/concept word vector is 80.26% and the highest F-measure with 200 and 300 dimension class/concept word vector is 80.36%. The difference is the only 0.1%.

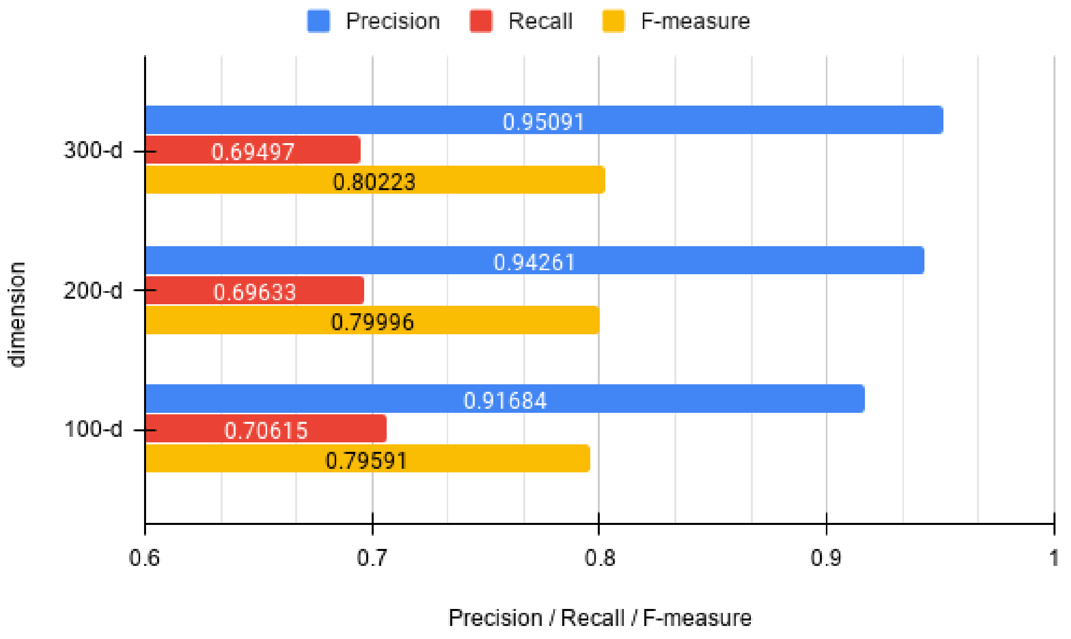


Figure 15: Performance of ontology alignment system - Recursive Neural Network for different class/concept vector dimensions (100d, 200d, 300d) on Anatomy dataset

The average precision, recall, and F-measure of the ontology alignment system with different values of k in the top-k predicted target class/concept across all the class/concept vector dimensions and the similarity threshold values are presented in Figure 16. The figure shows that there is only a minuscule change in the performance of the ontology alignment system with the increase of the number of target class/concept predictions. Therefore, our tool may be used as both an autonomous tool (i.e., without any human intervention) and as an assistive tool to help a domain expert by reducing the search space for ontology alignment in any domain.

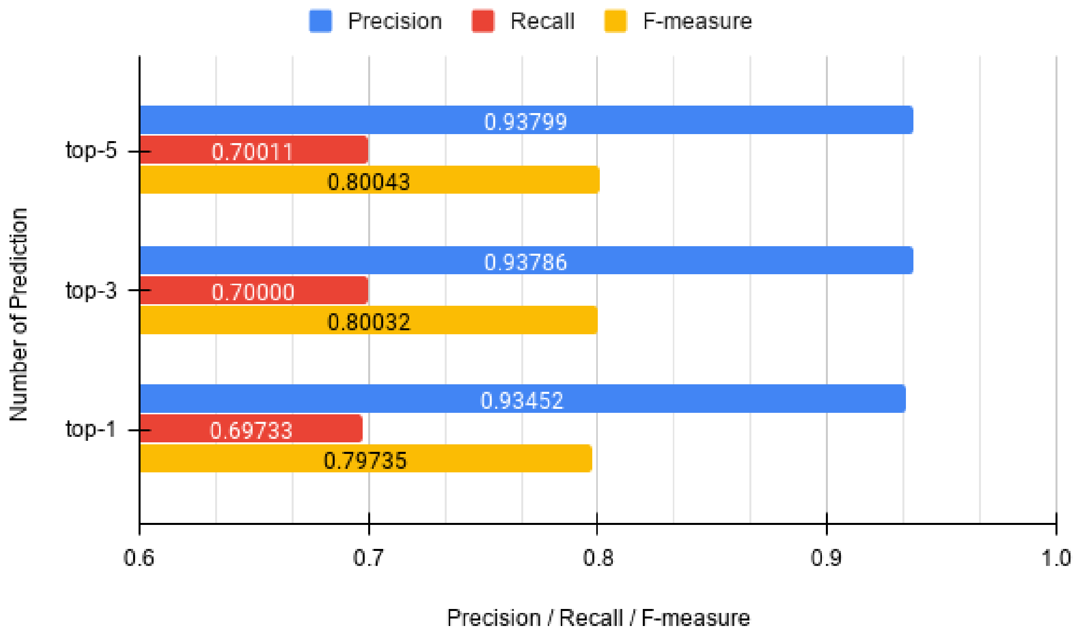


Figure 16: Performance of ontology alignment system - Recursive Neural Approach for top-k predictions with k = 1, 3, 5 on Anatomy dataset

3.5.4 Results & Findings on Library Dataset

The proposed Recursive Neural Network is also tested on Library dataset ("LIBRARY" 2013) with similar parameters i.e. similarity threshold value and class/concept word-vector dimension. Table 6, 7, 8 represent the precision, recall and F-measure of the OntoConnect Alignment System - recursive neural network for top-k predictions of target class/concept where k=1,3,5 with 100, 200, and 300 dimension class/concept word-vector respectively.

The Table 6 shows the precision increases and the recall decreases with the increase of the similarity threshold. We can observe in the Table 6 with 100 dimension class/concept vector the best result produced by the OntoConnect system is 74.8%

for top-1 prediction with similarity threshold 0.96. From the result posted by OAEI [3] we can observe that the proposed OntoConnect approach exhibits better result and comparable to the state-of-the-art tools. We can observer the same result pattern for 200 dimension entity word-vector in Table 7 and for the 300 dimension entity word-vector in Table 8.

[3]http://oaei.ontologymatching.org/2013/results/library/index.html

Table 6: Experimental results of OntoConnect Ontology Alignment system - Recursive Neural Approach for varying similarity thresholds: Precision, Recall, and F-measure values for class/concept vector dimensions = 100 and number of predictions k = 1,3,5 on Library dataset

		100-dimension		
Number of Prediction	Similarity Threshold	Precision	Recall	F-measure
Top-1	0.99	79.2	67.8	73.1
	0.98	79.2	67.9	73.1
	0.97	79.0	68.7	73.5
	0.96	78.3	71.7	**74.8**
	0.95	71.6	73.0	72.3
	0.94	61.7	73.9	67.2
	0.93	54.0	74.6	62.7
	0.92	48.3	75.4	58.9
	0.91	44.9	76.7	56.6
	0.90	42.1	77.5	54.5
Top-3	0.99	79.2	67.8	73.1
	0.98	79.2	67.9	73.1
	0.97	79.0	68.7	73.5
	0.96	78.3	71.7	74.8
	0.95	71.7	73.1	72.4
	0.94	61.9	74.2	67.5
	0.93	54.6	75.5	63.4
	0.92	49.3	76.9	60.1
	0.91	45.7	78.2	57.7
	0.90	43.0	79.1	55.7
Top-5	0.99	79.2	67.8	73.1
	0.98	79.2	67.9	73.1
	0.97	79.0	68.7	73.5
	0.96	78.3	71.7	74.8
	0.95	71.8	73.1	72.4
	0.94	62.0	74.3	67.6
	0.93	54.7	75.6	63.5
	0.92	49.3	77.0	60.1
	0.91	45.8	78.2	57.8
	0.90	43.1	79.4	55.9

Table 7: Experimental results of OntoConnect Ontology Alignment system - Recursive Neural Network for varying similarity thresholds: Precision, Recall, and F-measure values for class/concept vector dimensions = 200 and number of predictions k = 1,3,5 on Library dataset

		200-dimension		
Number of Prediction	Similarity Threshold	Precision	Recall	F-measure
Top-1	0.99	79.2	67.6	72.9
	0.98	79.2	67.8	73.1
	0.97	79.3	67.9	73.2
	0.96	79.3	68.2	73.4
	0.95	78.9	68.8	**73.5**
	0.94	77.2	69.5	73.1
	0.93	75.2	70.1	72.5
	0.92	71.2	70.8	71.0
	0.91	66.4	71.8	69.0
	0.90	61.6	72.7	66.7
Top-3	0.99	79.2	67.6	72.9
	0.98	79.2	67.8	73.1
	0.97	79.3	67.9	73.1
	0.96	79.4	68.3	73.4
	0.95	78.9	68.8	73.5
	0.94	77.2	69.5	73.1
	0.93	75.2	70.1	72.5
	0.92	71.2	70.8	71.0
	0.91	66.6	72.1	69.3
	0.90	61.9	73.1	67.0
Top-5	0.99	79.2	67.6	72.9
	0.98	79.2	67.8	73.1
	0.97	79.2	67.9	73.1
	0.96	79.3	68.2	73.4
	0.95	78.9	68.8	73.5
	0.94	77.2	69.5	73.1
	0.93	75.2	70.1	72.5
	0.92	71.2	70.8	71.0
	0.91	66.6	72.1	69.3
	0.90	61.9	73.1	67.0

Table 8: Experimental results of OntoConnect Ontology Alignment system - Recursive Neural Network for varying similarity thresholds: Precision, Recall, and F-measure values for class/concept vector dimensions = 300 and number of predictions k = 1,3,5 on Library dataset

		300-dimension		
Number of Prediction	Similarity Threshold	Precision	Recall	F-measure
Top-1	0.99	0.794	67.6	73.0
	0.98	79.3	67.8	73.1
	0.97	79.3	67.9	73.2
	0.96	79.3	68.2	73.4
	0.95	79.0	68.8	**73.5**
	0.94	77.2	69.5	73.1
	0.93	75.4	70.1	72.5
	0.92	71.2	70.8	71.0
	0.91	66.8	71.8	69.0
	0.90	62.0	72.7	66.7
Top-3	0.99	79.4	67.6	72.9
	0.98	79.3	67.8	73.1
	0.97	79.4	67.9	73.1
	0.96	79.3	68.3	73.4
	0.95	79.0	68.8	73.5
	0.94	77.2	69.5	73.1
	0.93	75.4	70.1	72.5
	0.92	71.2	70.8	71.0
	0.91	66.8	72.1	69.3
	0.90	62.5	73.1	67.0
Top-5	0.99	79.6	67.6	72.9
	0.98	79.3	67.8	73.1
	0.97	79.3	67.9	73.1
	0.96	79.3	68.2	73.4
	0.95	79.0	68.8	73.5
	0.94	77.2	69.5	73.1
	0.93	75.4	70.1	72.5
	0.92	71.2	70.8	71.0
	0.91	66.8	72.1	69.3
	0.90	62.0	73.1	67.0

3.5.5 Result Analysis on Library Dataset

The average precision, recall and F-measure across all the class/concept vector dimensions (100, 200, 300) and the number of target class/concept predictions (k=1,3,5) are presented in Figure 17. It shows the change in precision, recall, and F-measure of the ontology alignment system with different similarity values. It can be observed that the precision increases whereas the recall decrease with the increase in the similarity threshold value. From the result analysis, it can be noted that with a 0.96 similarity threshold value, the proposed ontology alignment system - Recursive Neural Approach performs best with the highest average F-measure value of 74.8%.

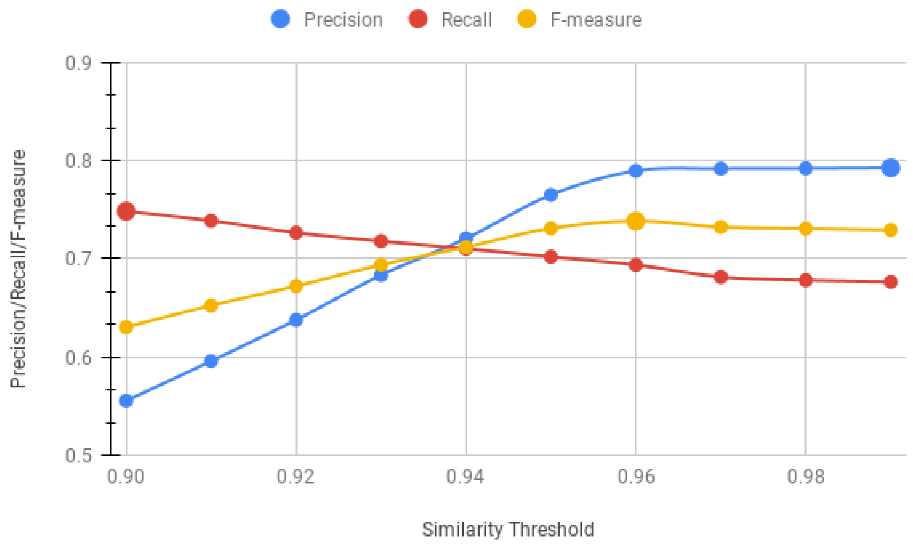


Figure 17: Change in evaluation metric values (precision, recall, F-measure) of Ontology alignment system - Recursive Neural Network with increasing similarity threshold values on Library dataset

Similarly, Figure 18 shows the change in the average precision, recall, and F-measure values with different lengths of class/concept vector dimensions across all the similarity threshold values and the top-k (=1,2,3) predicted target classes/concepts. It can be observed that only a little change in the performance of the ontology alignment system occurs with the increase of the class/concept vector dimensions. Thus, it can be noted that the performance of the proposed system stays almost invariant towards the change in class/concept vector length.

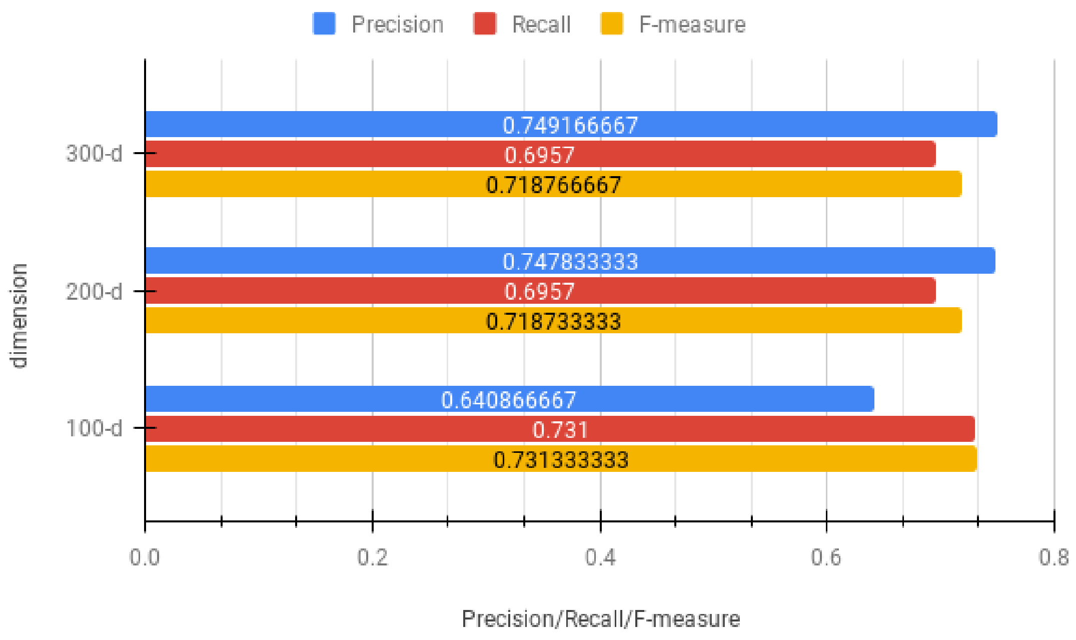


Figure 18: Performance of ontology alignment system - Recursive Neural Network for different class/concept vector dimensions (100d, 200d, 300d) on Library dataset

The average precision, recall, and F-measure of the ontology alignment system with different values of k in the top-k predicted target class/concept across all the class/concept vector dimensions and the similarity threshold values are presented in Figure 19. The figure shows that there is only a minuscule change in the performance of the ontology alignment system with the increase of the number of target class/concept predictions. Therefore, our tool may be used as both an autonomous tool (i.e., without any human intervention) or an assistive tool to help a domain expert by reducing the search space for ontology alignment in any domain.

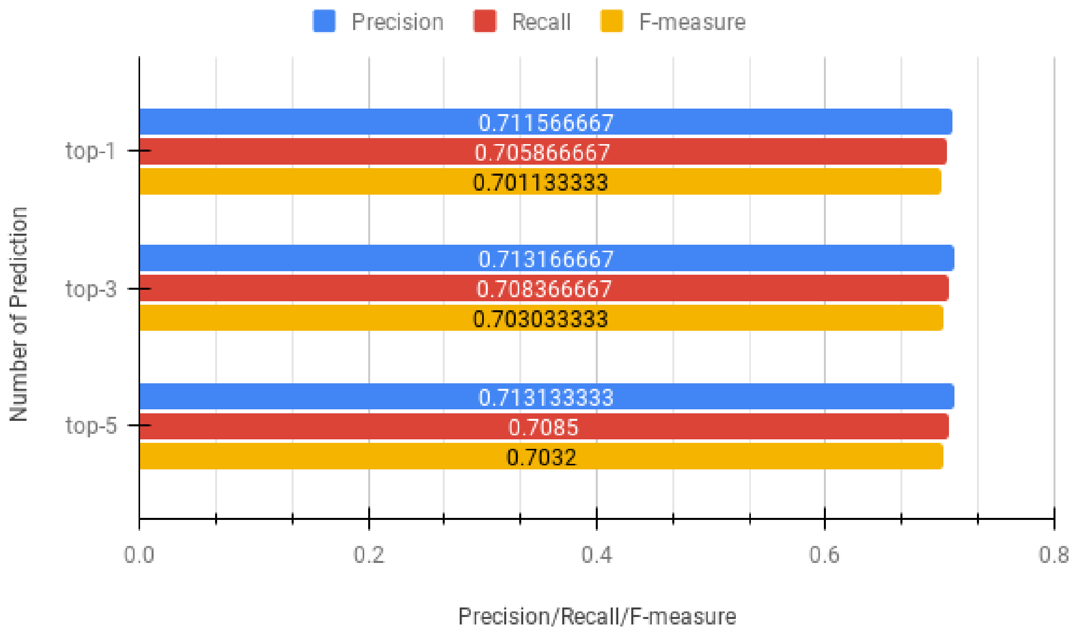


Figure 19: Performance of ontology alignment system - Recursive Neural Network for top-k predictions with k = 1, 3, 5 on Library dataset

3.6 Conclusion

In Conclusion, we can say that OntoConnect is a domain-independent ontology alignment system. With minimum or no change, we are able to train and execute the OntoConnect system on different datasets like Anatomy (Bodenreider et al. 2005) and Library ("LIBRARY" 2013). The OntoConnect system is able to do the alignment between source and target ontologies without any specific domain information and without any domain expert intervention. OntoConnect System only learned from the source ontology and predicts the similar source class/concept for each target ontology class/concept. OntoConnect also proves that meta-information or structural information of a class/concept adds more information that enhances the overall ontology alignment process.

Chapter 4

ONTOLOGY ALIGNMENT WITH GRAPH NEURAL NETWORK

In our experiment with the OntoConnect system, we have explored three different graph neural methods. The first graph neural method is graph convolution network (GCN). The graph neural network uses generalize neural learning concepts to graph-like data and enhances the embedding of each node by a message-passing scheme. In the paper (T N Kipf and Welling 2016), the authors describe the scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. The GCN layer defined in (T N Kipf and Welling 2016) as below equation 4.1. In equation 4.1, $\mathbf{x}_v^{(\ell+1)}$ is the node features of all nodes $v \in \mathcal{V}$ in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. $\mathbf{W}^{(\ell+1)}$ denotes a trainable weight matrix and $c_{w,v}$ refers to a fixed normalization coefficient for each edge.

$$\mathbf{x}_v^{(\ell+1)} = \mathbf{W}^{(\ell+1)} \sum_{w \in \mathcal{N}(v) \,\cup\, \{v\}} \frac{1}{c_{w,v}} \cdot \mathbf{x}_w^{(\ell)} \tag{4.1}$$

In graph convolution network (GCN), the weightage of the edges is defined explicitly. In the second graph neural method, i.e., graph attention network (Veličković et al. 2017) decides the weightage of the edges of the graph implicitly. It employs self-attention over the node features. In the case of GAT, the embeddings from the neighbors are aggregated together and scaled by the attention layer. Figure 20 shows a GAT layer with multi-head attention. Every neighbor i of node1 sends its own embedding of attentional coefficients, $\vec{\alpha_{1i}}$ one per each attention head α_{1i}^k. These are used to compute K separate linear combinations of neighbors' features $\vec{h_i}$. It aggregates all the combinations by concatenation or averaging to obtain the next level features of node 1, $\vec{h_1}$.

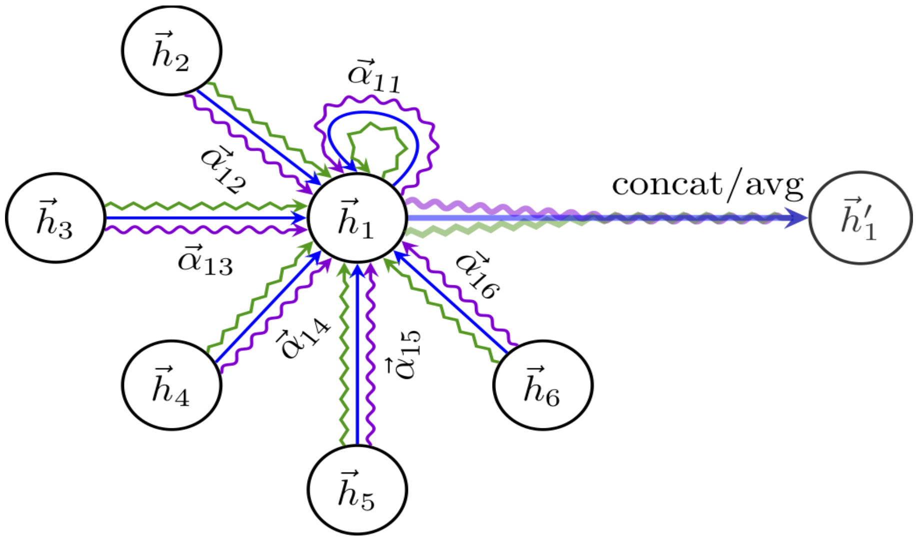


Figure 20: Graph Attention Networks Layer

In the following section, the third neural network method is described. The main working principle of the third neural network approach, i.e., graph embedding with negative sampling (Lerer et al. 2019) is to calculate the score of a node in the graph and maximizes the score difference between positive and negative edges present in the graph.

4.1 Proposed Graph Neural Network Methodology

The overall *OntoConnect* alignment process with graph neural network can be divided into two main processes. The first one is generating the embedding of the source ontology classes/concepts and the target ontology classes/concepts. The second one is calculating the similarity score between source classes/concepts for each target class/concept to form correspondences.

The generation of embedding of the source and target ontology classes/concepts is described in Algorithm 3. The strategy is to train a graph neural network model in a stochastic manner with each meta-information of source and target classes/concepts. The first step is to collect the meta-information for each source class/concept. The function "getMetaInformation" is responsible for collecting the meta-information for a class from the ontology. The function "getVector" retrieves the word-vector for each source class/concept from the pre-trained FastText (Bojanowski et al. 2017). The function "getEmbed" trains the graph neural model for each class/concept and encodes each class/concept into embedding which captures the meta information or structural information of that particular class/concept.

Algorithm 3: *OntoConnect* Graph Embedding algorithm

```
Input: Set of concepts in source and target ontology
Output: Graph Embedding of source and target ontology concepts
1 $onto\_graph\_embed\_set \leftarrow \emptyset$
2 foreach $concept\ c_i \in concepts$ do
      /* Retrive vector from pre-trained model FastText for the concept $c_i$ */
3     $Vec_{c_i} \leftarrow getVector(c_i)$
      /* Retrive meta-information (parent, child, equivalent, disjoint,
         restriction) of the concept $c_i$ from the ontology. */
4     $Meta_{c_i} \leftarrow getMetaInformation(c_i)$
      /* Retrive vector from pre-trained model FastText for each
         meta-information of the concept $c_i$ */
5     $MetaVec_{c_i} \leftarrow getVector(Meta_{c_i})$
      /* Retrieve Embedding for the concept $c_i$ stochastically */
6     $onto\_graph\_embed\_set \leftarrow getEmbed(Vec_{c_i}, MetaVec_{c_i})$
7 return $onto\_graph\_embed\_set$
```

Algorithm 4 shows the algorithm for calculating the similarity between the source and target ontology classes/concepts. The method “getVector” retrieves the word-vector for a class/concept for both source and target ontology. Now, for a target class/concept, we are calculating similarity for each source ontology class/concept. The word similarity ($Word_{sim_{i,j}}$) is the cosine similarity between the target and source ontology class/concept word-vectors retrieved from the pre-trained FastText model. On the other hand, meta-similarity ($Meta_{sim_{i,j}}$) is the cosine similarity between the target and source ontology class/concept graph-embedding from the graph neural network model. Next, a weighted combined similarity of the word-similarity ($Word_{sim_{i,j}}$) and meta-similarity ($Meta_{sim_{i,j}}$) is calculated for each pair of the target and source ontology classes/concepts. We are considering the source ontology class/concept as a similar class/concept with the maximum combined similarity (Cmb_{sim_i}). The correspondence list is updated with the most similar class/concept pair having the highest similarity score. In the end, it returns the correspondence list.

Algorithm 4: *OntoConnect* Similarity Calculation algorithm

```
Input: set of concepts in source ontology $sConcepts$, set of concepts in
       target ontology $tConcepts$
Output: List of Correspondences $correspondences$
1  $correspondences \leftarrow \emptyset$
2  foreach target concept $t_i \in tConcepts$ do
       /* Retrive vector from pre-trained model FastText for the target
          concept $t_i$ */
3      $Vec_{t_i} \leftarrow getVector(t_i)$
       /* Retrive graph embedding of the target concept $t_i$ */
4      $Embed_{t_i} \leftarrow getEmbedVec(t_i)$
5      $Cmb_{sim_i} \leftarrow \emptyset$
6      $correspondence_i \leftarrow \emptyset$
7      foreach source concept $s_j \in sConcepts$ do
           /* Retrive vector from pre-trained model FastText for the source
              concept $s_j$ */
8          $Vec_{s_j} \leftarrow getVector(s_j)$
           /* Compute cosine-similarity between $Vec_{t_i}, Vec_{s_j}$ */
9          $Word_{sim_{i,j}} \leftarrow computeSimilarity(Vec_{t_i}, Vec_{s_j})$
           /* Retrive graph embedding of the source concept $s_j$ */
10         $Embed_{s_j} \leftarrow getEmbedVec(s_j)$
           /* Compute cosine-similarity between $Embed_{t_i}, Embed_{s_j}$ */
11         $Meta_{sim_{i,j}} \leftarrow computeSimilarity(Embed_{t_i}, Embed_{s_j})$
12         $TmpCmb_{sim_{i,j}} \leftarrow$
            $weightedHarmonicMean(Word_{sim_{i,j}}, Meta_{sim_{i,j}})$
           /* Updating $correspondence_i$ based on combined similarity */
13         if $TmpCmb_{sim_{i,j}} > Cmb_{sim_i}$ then
14             $Cmb_{sim_i} \leftarrow TmpCmb_{sim_{i,j}}$
                $correspondence_i \leftarrow [t_i, s_j, Cmb_{sim_i}]$
       /* Updating $correspondences$ list */
15     $correspondences \leftarrow correspondences + correspondence_i$
16 return $correspondences$
```

Next, we will present each step in detail for both the training/learning of the OntoConnect model and predicting similar class/concept by using the graph embedding. Our proposed Ontology Alignment system consists of two main tasks: the first task is generating the embedding for each class/concept in the source and target ontology and the second task is to calculate the similarity score between the source and target ontology class/concept from the embedding and generate correspondence list.

Figure 21 represents a workflow of the proposed *OntoConnect* ontology alignment system using the Graph Neural Network.

In the Graph Neural Network, in the first step, OWL API is used to extract the meta-information of each class/concept from the source ontology and target ontology. In the next step, the same data preprocessing techniques are used on the labels of the extracted class/concept. After that, we have used a pre-trained model fastText to generate vectors from each source and target ontological classes/concepts. Each class/concept of the source and target and its meta-information are fed to the Graph Neural network which produces entity embedding for each entity. In the next phase, both word-similarity and meta-similarity are calculated for each pair of classes/concepts of source and target ontologies. Based on the similarity score, the list of correspondence is generated. In the following section, the project flow of the *OntoConnect* using the Graph Neural Network is explained in detail.

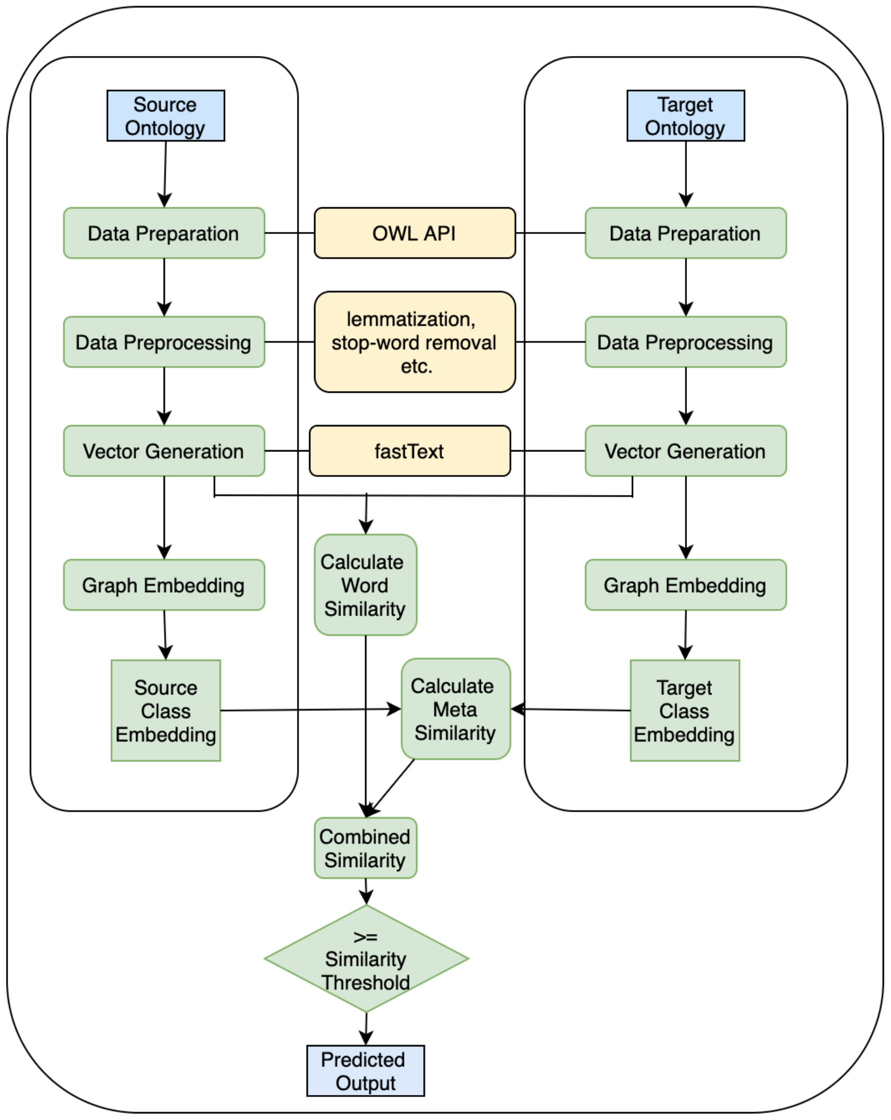


Figure 21: Project Flow of *OntoConnect* - Graph Neural Network

4.2 Data Preparation

In the Data Preparation step, the source and target ontology are parsed and converted into an Ontology Matching model. For parsing the source ontology, we have used a Java API named OWL API (Horridge and Bechhofer 2011) and HermiT

Reasoner (Motik, Shearer, and Horrocks 2007) to extract meta information of a class, such as IRI of the class (e_iri), label of the class (e_lbl), restriction of the class (rc), parent (pc), child (cc), equivalent (ec), and disjoint classes (dc) of each class/concept of the source ontology. OWL API is a high-level API to work with OWL ontology, it supports parsing, reading, reasoning, manipulating, validating, and rendering of OWL ontology files. HermiT is a reasoning engine integrated into OWL API which can determine or identify the subsumption relationships between classes/concepts.

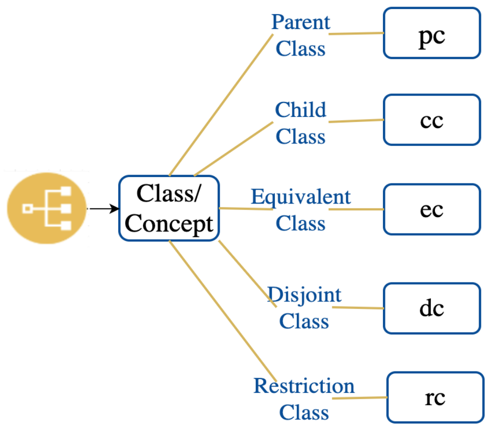


Figure 22: Meta information of a Source/Target Ontology Class/Concept

Next, different pre-processing techniques are used on the class/concept labels such as Normalization, Lemmatization, Stop word removal, Tokenization, etc. We have followed the same data pre-processing techniques. The details of different data pre-processing techniques are previously mentioned in section 3.2. After that, a vector form of each class/concept is generated. A pre-trained embedding model fastText (Bojanowski et al. 2017) developed by Facebook's AI Research (FAIR) lab is used. This embedding model uses the skip-gram technique, where each word is considered as a bag of n-gram characters. It learns the representations for character n-grams and represents the words as the sum of the n-gram vectors. The main advantage of the model provided by fastText is that it can generate a meaningful vector for a word that is not present in its dictionary. In particular, we used a model trained on Wikipedia 2017, UMBC WebBase corpus, and statmt.org news data set, which generates a 300-dimension vector for each word.

4.3 Entity Embedding

In this step, different contexts of classes/concepts will be used to make its vector representation semantically richer. From the Ontology Matching model, a class/-concept can have attributes like Parent Classes, Child Classes, Equivalent Classes, Disjoint Classes, Restrictions.

Figure 23 shows an example of a training data which will be used in training Graph Neural Network. Let's assume the concept/class is "c_1". "c_1" has meta information "parent class", "child class", "equivalent class", "disjoint class" and "restriction class". The parent class of "c_1" is "pc_1c_1". "cc_1c_1", "cc_2c_1", "ec_1c_1", "dc_1c_1", "$rc_1'c_1$", "$rc_1''c_1$" are child classes, equivalent class, disjoint class and restriction classes of "c_1" respectively.

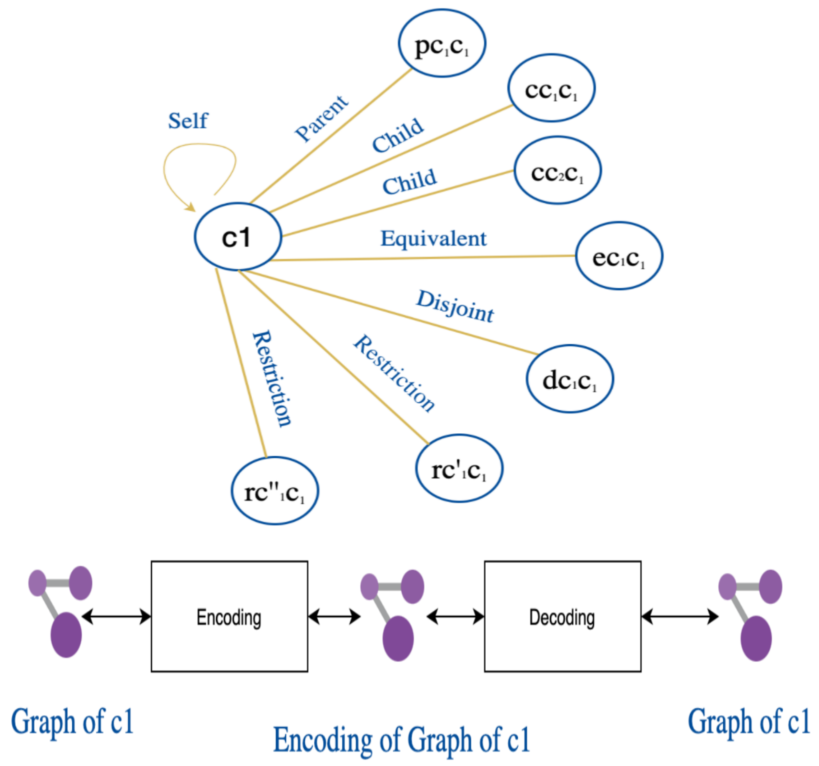


Figure 23: Graph structure of a sample ontology class and High level representation of Graph Embedding process

PyTorch-BigGraph (PBG) (Lerer et al. 2019) is a distributed system for learning graph embeddings for large graphs. It operates on graphs with vertices having multiple edges. Here the vertices are called entities and the edges are the relation between source and destination entity. A multi-relation graph is a directed graph $G = (V, R, E)$ where V are the nodes or entities, R is a set of relations, and E is a set of edges. In the graph, a generic element $e = (s, r, d)$ where s is the source node/entity, d is the destination node/entity, and r is the relation between s and d. Here, $s, d \in V$ and $r \in R$.

Equation 4.2 is the score function used in the PyTorch-BigGraph (PBG). In the equation 4.2, θ_s is the source entity, θ_d is the destination entity, and θ_r is the relation between them. $g_s(\theta_s, \theta_r)$ is the "complex-diagonal" operator between θ_s and θ_r. On the other hand, $g_d(\theta_d, \theta_r)$ is the "complex-diagonal" operator between θ_d and θ_r. PBG tries to maximize the score function $f(\theta_s, \theta_r, \theta_d)$ for any $(s, r, d) \in E$ and minimizes it for $(s, r, d) \notin E$.

$$f(\theta_s, \theta_r, \theta_d) = sim(g_s(\theta_s, \theta_r), g_d(\theta_d, \theta_r)) \tag{4.2}$$

The learning principle of the PyTorch-BigGraph (PBG) is to find embeddings for the entities so the distance between the neighbor nodes and the entity should be closer. On the other hand, the distance between non-neighbor nodes and the entity should be longer. The edges of the input graph data given to the PyTorch-BigGraph (PBG) model are treated as positive edges. It produces a set of negative edges for each positive edge. It produces negative samples for a given positive edge by a corrupted version of the entity to one side and keeping the other side intact (Bordes et al. 2013). PyTorch-BigGraph (PBG) uses different ways to sample negative edges. For our experiment, "all negative" is used to generate the negative samples. "all negative" sampling method creates a negative edge/relation between the source and the destination node. Let's say r is a positive edge between two entities s and d. For each such positive edge (s, r, d) there will be a negative edge (s', r, d) between s' and d where s' is of the same entity type of s. Besides this, there will be another negative edge (s, r, d') between s and d' where d' is of the same entity type of d.

Equation 4.3 is the loss ($\mathcal{L}$) used in the PyTorch-BigGraph (PBG). The main idea of PBG is to maximize the scores of positive edges and minimize the scores of negative edges. Here, G is a list of edges. S'_e is the set of negative edges for every positive edge. $f(e)$ is the score of a positive edge and $f(e')$ is the score of a negative edge. λ is the regularization parameter.

$$
\begin{aligned}
\mathcal{L} &= \sum_{e \in G} \sum_{e' \in S'_e} max(f(e) - f(e') + \lambda, 0) \\
S'_e &= (s', r, d) | s' \in V \cup (s, r, d') | d' \in V
\end{aligned} \tag{4.3}
$$

PBG updates the embeddings and related parameters in minibatch stochastic gradient descent (SGD). It used an Adagrad optimizer and sum the accumulated gradient over each embedding vector to reduce the memory usage on large graphs (Duchi, Hazan, and Singer 2011).

4.4 Correspondence Generation

To generate the correspondence, two similarity values are calculated. The first one is word similarity ($Word_{sim}$) and the second one is the meta similarity ($Meta_{sim}$). In our experiment, we have calculated a combined similarity (Cmb_{sim}) by the harmonic mean of the word and meta similarity.

The word similarity ($Word_{sim}$) is the cosine distance between the source and target class/concept vectors. Equation 4.4 shows the cosine similarity measurement between two vectors $\overrightarrow{s}$ and $\overrightarrow{t}$, which is the cosine of the angle projected in a multi-dimensional (d) space. In our experiment, the time complexity of this step is $O(mn)$, where m is the number of source ontology classes and n is the number of target ontology classes.

$$
\begin{aligned}
\cos(\overrightarrow{s}, \overrightarrow{t}) &= \frac{\overrightarrow{s} \cdot \overrightarrow{t}}{\|\overrightarrow{s}\| \|\overrightarrow{t}\|} \\
&= \frac{\sum_{i=1}^{d} \mathbf{s}_i \mathbf{t}_i}{\sqrt{\sum_{i=1}^{d} (\mathbf{s}_i)^2} \sqrt{\sum_{i=1}^{d} (\mathbf{t}_i)^2}}
\end{aligned} \tag{4.4}
$$

The meta similarity ($Meta_{sim}$) is the cosine distance between the source and target class/concept graph embedding. Equation 4.5 shows the cosine similarity measurement between two vectors $\overrightarrow{s_g}$ and $\overrightarrow{t_g}$, which is the cosine of the angle projected in a multi-dimensional (d) space. In our experiment, the time complexity of this step is $O(mn)$, where m is the number of source ontology classes and n is the number of target ontology classes.

$$\begin{aligned} \cos(\overrightarrow{s_g}, \overrightarrow{t_g}) &= \frac{\overrightarrow{s_g} \cdot \overrightarrow{t_g}}{\|\overrightarrow{s_g}\|\|\overrightarrow{t_g}\|} \\ &= \frac{\sum_{i=1}^{d} \mathbf{s_g}_i \mathbf{t_g}_i}{\sqrt{\sum_{i=1}^{d} (\mathbf{s_g}_i)^2}\sqrt{\sum_{i=1}^{d} (\mathbf{t_g}_i)^2}} \end{aligned} \tag{4.5}$$

4.5 Experimental Study

4.5.1 Evaluation Methodology

For evaluating the graph neural network methodology, we have used the same intrinsic evaluation method mentioned in section 3.5.1. We have followed the standard precision and recall measurement mentioned in (Crestani and Rijsbergen 1995). The result of the OntoConnect graph neural network approach on the Anatomy dataset is mentioned in the following section.

4.5.2 Results & Findings

We have tested the OntoConnect - Graph Neural Network on Anatomy ("Anatomy" 2013) dataset in the similar way. We have varied parameters, i.e., similarity threshold and class/concept vector dimension. For each combination of parameters the precision, recall and F-measure are calculated for top-k predictions of target class/concept where k=1,3,5 with 100, 200, 300 dimension word-vector. Table 9, 10, 11 tables show the outcome for the each combination of parameters.

Table 9: Experimental results of OntoConnect Ontology Alignment system - Graph Neural Network for varying similarity thresholds: Precision, Recall, and F-measure values for class/concept vector dimensions = 100 and number of predictions k = 1,3,5 on Anatomy dataset

		100-dimension		
Number of Prediction	Similarity Threshold	Precision	Recall	F-measure
Top-1	0.99	97.4	67.4	79.6
	0.98	93.5	71.0	**80.7**
	0.97	85.3	73.0	78.7
	0.96	75.1	75.2	75.1
	0.95	67.8	76.6	71.9
	0.94	61.9	77.5	68.8
	0.93	57.8	78.6	66.6
	0.92	55.3	79.2	65.1
	0.91	53.2	80.4	64.1
	0.90	51.2	81.0	62.8
Top-3	0.99	97.6	67.6	79.9
	0.98	94.0	71.4	**81.1**
	0.97	86.2	73.7	79.5
	0.96	76.4	76.5	76.6
	0.95	69.7	78.7	73.9
	0.94	63.7	79.9	70.9
	0.93	59.8	81.3	68.9
	0.92	57.4	82.2	67.6
	0.91	55.4	83.6	66.7
	0.90	53.4	84.4	65.4
Top-5	0.99	97.6	67.6	79.9
	0.98	94.0	71.4	**81.1**
	0.97	86.3	73.8	79.6
	0.96	76.6	76.7	76.7
	0.95	69.9	79.0	74.2
	0.94	64.0	80.2	71.2
	0.93	60.3	82.0	69.5
	0.92	57.9	83.0	68.2
	0.91	56.2	84.8	67.6
	0.90	54.2	85.6	66.4

Table 10: Experimental results of OntoConnect Ontology Alignment system - Graph Neural Network for varying similarity thresholds: Precision, Recall, and F-measure values for class/concept vector dimensions = 200 and number of predictions k = 1,3,5 on Anatomy dataset

		200-dimension		
Number of Prediction	Similarity Threshold	Precision	Recall	F-measure
Top-1	0.99	97.9	67.0	79.5
	0.98	96.7	69.2	80.6
	0.97	93.0	71.5	**80.8**
	0.96	88.0	73.3	80.0
	0.95	80.8	75.0	77.7
	0.94	74.6	76.4	75.5
	0.93	69.2	77.3	73.0
	0.92	64.8	78.6	71.1
	0.91	61.2	79.4	69.0
	0.90	58.1	80.1	67.3
Top-3	0.99	98.1	67.1	79.7
	0.98	96.9	69.3	80.8
	0.97	93.4	71.8	**81.2**
	0.96	88.5	73.7	80.5
	0.95	81.5	75.7	78.4
	0.94	75.8	77.6	76.7
	0.93	70.7	79.0	74.6
	0.92	66.5	80.7	72.9
	0.91	62.8	81.6	70.9
	0.90	59.8	82.5	69.4
Top-5	0.99	98.1	67.1	79.7
	0.98	96.9	69.3	80.8
	0.97	93.4	71.8	**81.2**
	0.96	88.5	73.7	80.5
	0.95	81.5	75.7	78.4
	0.94	76.0	77.9	76.9
	0.93	70.9	79.3	74.9
	0.92	66.9	81.2	73.4
	0.91	63.2	82.1	71.4
	0.90	60.4	83.2	70.0

Table 11: Experimental results of OntoConnect Ontology Alignment system - Graph Neural Network for varying similarity thresholds: Precision, Recall, and F-measure values for class/concept vector dimensions = 300 and number of predictions k = 1,3,5 on Anatomy dataset

		300-dimension		
Number of Prediction	Similarity Threshold	Precision	Recall	F-measure
Top-1	0.99	97.9	66.9	79.5
	0.98	97.3	68.3	80.3
	0.97	95.3	70.5	**81.0**
	0.96	91.6	72.1	80.7
	0.95	86.8	73.6	79.7
	0.94	81.5	75.0	78.1
	0.93	75.4	76.4	75.9
	0.92	71.3	77.4	74.2
	0.91	67.1	78.3	72.3
	0.90	64.1	78.9	70.8
Top-3	0.99	98.2	67.1	79.7
	0.98	97.6	68.5	80.5
	0.97	95.6	70.6	**81.2**
	0.96	92.0	72.4	81.0
	0.95	87.3	74.0	80.2
	0.94	82.1	75.6	78.7
	0.93	76.4	77.4	76.9
	0.92	72.5	78.7	75.5
	0.91	68.7	80.2	74.0
	0.90	65.8	81.1	72.6
Top-5	0.99	98.2	67.1	79.7
	0.98	97.6	68.5	80.5
	0.97	95.5	70.7	**81.2**
	0.96	92.0	72.4	81.0
	0.95	87.3	74.0	80.2
	0.94	82.2	75.7	78.8
	0.93	76.6	77.6	77.1
	0.92	72.8	79.0	75.8
	0.91	69.0	80.5	74.3
	0.90	66.2	81.5	73.0

4.5.3 Result Analysis

The average precision, recall and F-measure across all the class/concept vector dimensions (100, 200, 300) and the number of target class/concept predictions (k=1,3,5) are presented in Figure 24. It shows the change in precision, recall, and F-measure of the ontology alignment system with different similarity values. It can be observed that the precision increases whereas the recall decrease with the increase in the similarity threshold value. This is expected as with a higher similarity threshold, the ontology alignment system returns a lower number of correspondences compared to the number of existing correspondences in the reference alignment. From the result analysis, it can be noted that with a 0.97 similarity threshold value, the proposed ontology alignment system - Graph Neural Approach performs best with the highest average F-measure value of 81.0%.

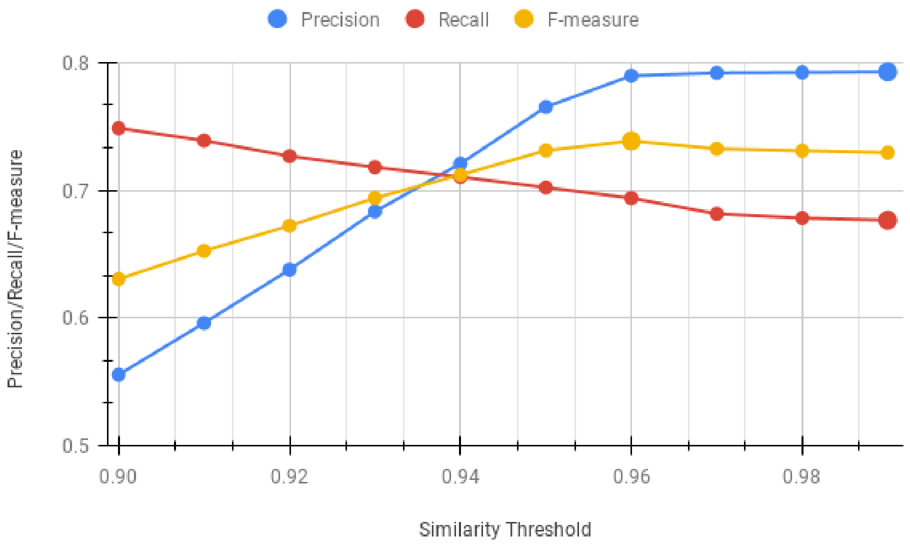


Figure 24: Change in evaluation metric values (precision, recall, F-measure) of Ontology alignment system - Graph Neural Network with increasing similarity threshold values on Anatomy dataset

Similarly, Figure 25 shows the change in the average precision, recall, and F-measure values with different lengths of class/concept vector dimensions across all the similarity threshold values and the top-k (=1,2,3) predicted target classes/concepts. It can be observed that only a minuscule change in the performance of the ontology alignment system occurs with the increase of the class/concept vector dimensions. Thus, it can be noted that the performance of the proposed system stays almost invariant towards the change in class/concept vector length.

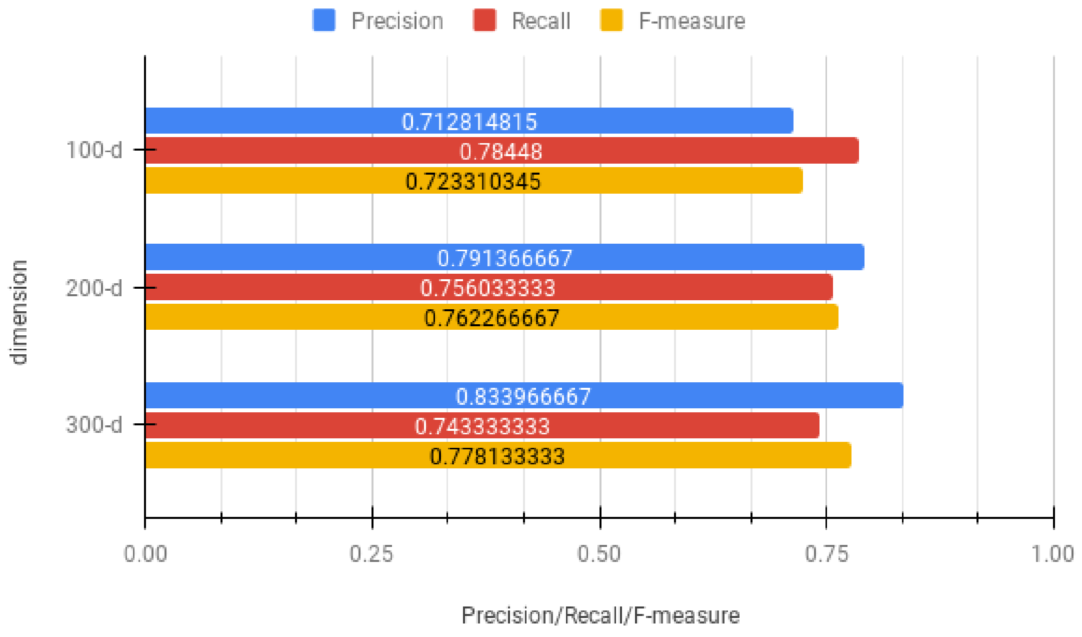


Figure 25: Performance of ontology alignment system - Graph Neural Network for different class/concept vector dimensions (100d, 200d, 300d) on Anatomy dataset

The average precision, recall, and F-measure of the ontology alignment system with different values of k in the top-k predicted target class/concept across all the class/concept vector dimensions and the similarity threshold values are presented in Figure 26. The figure shows that there is only a minuscule change in the performance of the ontology alignment system with the increase of the number of target class/concept predictions. Therefore, our tool may be used as both an autonomous tool (without any human intervention) or an assistive tool to help a domain expert by reducing the search space for ontology alignment in any domain.

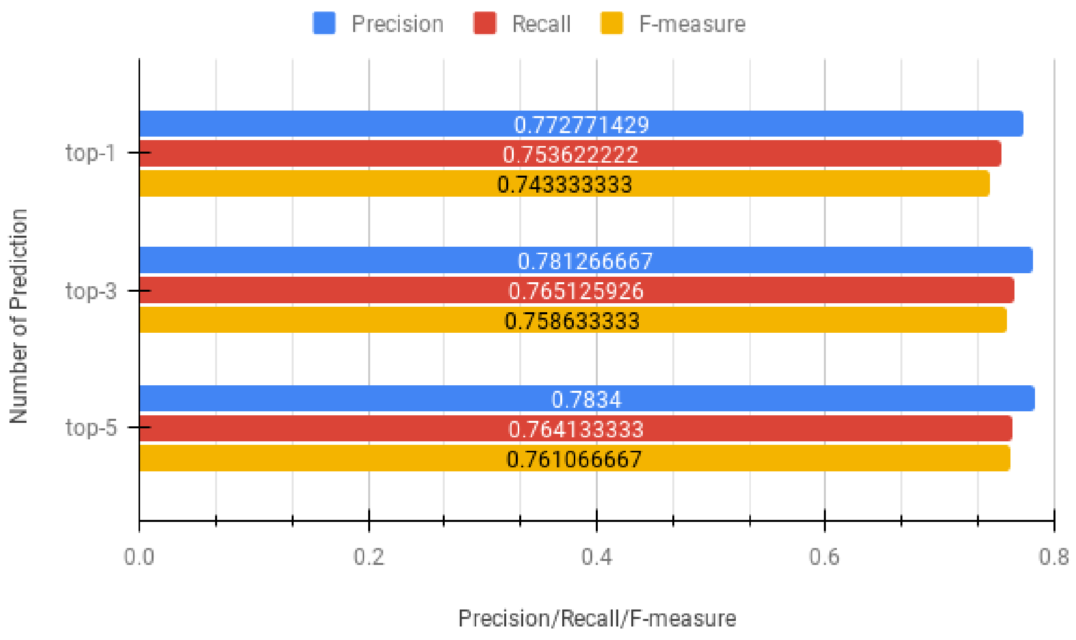


Figure 26: Performance of ontology alignment system - Graph Neural Network for top-k predictions with k = 1, 3, 5 on Anatomy dataset

4.6 Result Analysis using BERT

Both proposed recursive neural network and graph neural network is tested on Anatomy dataset using BERT pre-trained model. Table 12 shows the OntoConnect system using recursive neural network and BERT pre-trained model exhibits only 73.6% f-measure. It also shows 59.4% f-measure using the graph neural network. From the table, we can conclude that BERT is not an ideal pre-trained model for ontology alignment. As BERT focuses on the sentence and trains embedding for a word based on its position and its neighboring words. In ontology alignment, the source and the target ontology data are in an organized format, and position and bidirectional training do not add any value. For this reason, BERT performs poorly compare to fastText. We are experimenting with different versions of BERT and in our future scope, we will explore whether we can use BERT in ontology alignment in a better way or not.

		768-dimension		
Approach	**Similarity Threshold**	**Precision**	**Recall**	**F-measure**
Recursive Neural Network	0.99	77.4	70.1	73.6
Graph Neural Network	0.99	45.9	84.3	59.4

Table 12: Performance of OntoConnect on Anatomy Dataset using BERT pre-trained model

4.7 Conclusion

In Conclusion, we can say that graph embedding can be used in our proposed OntoConnect system. From the result, we can observe that graph embedding is a better choice to encapsulate the structural information or meta-information of a class/concept of both the source and the target ontology. In the graph neural network, we can also observe that the OntoConncet system is trained without any domain knowledge and without any domain expert intervention. From the result analysis, we can claim that both the recursive neural network and the graph neural network can be used on any ontology from any domain. It shows the reusability and extensibility of our proposed OntoConnect system. We can also observe that our proposed OntoConnect system can be used as an alternative to the state-of-art rule-based ontology alignment system.

Chapter 5

ABLATION STUDY

In this chapter, we present an ablation study of the OntoConnect system. The following section 5.1 shows the comparison of the word-similarity and the meta-similarity separately and also shows how the meta-similarity enhances the combined similarity between the ontological classes/concepts. Section 5.2 shows different test cases from the anatomy dataset which exhibit the effectiveness of the meta-similarity or structural information in the ontology alignment.

5.1 Comparison of similarities in OntoConnect System

From the result published by OAEI on Anatomy data [4], we can observe the base system "StringEquiv" has F-Measure 76.6%. It only considers the syntax similarity. In our ablation study, we have considered the performance of the "StringEquiv" system as word similarity. Table 13 shows the different similarities of the OntoConnect System on the Anatomy dataset. In the OntoConnect system, there are two similarities are calculated. The first similarity is the "Word Similarity" and the second similarity is the "Meta Similarity". The "Word Similarity" is the syntax similarity between class/concept of the source and target ontology and on the other hand "Meta Similarity" is the structural similarity between the concepts/classes. The OntoConnect system combines both the similarities to get combine similarity which is weighted harmonic mean to generate a correspondence list. From the table 13, we can claim that the OntoConnect

[4]http://oaei.ontologymatching.org/2020/results/anatomy/index.html

system is able to increase the 4% - 5% in F-measure of the ontology alignment process by using the meta-similarity. In table 13, we can observe that the structural information or the meta-information improves the overall performance of the ontology alignment. Both recursive neural network and graph neural network can be used to increase the overall accuracy of the ontology alignment process.

Table 13: Ablation study on results (F-measure) of OntoConnect system on Anatomy dataset using Recursive Neural Network and Graph Neural Network

		F-measure		
Approach	**Dimension**	**Word Similarity**	**Meta Similarity**	**Combined Similarity**
Recursive Neural Network with FastText on Anatomy dataset	100D	76.6	69.9	80.3
	200D	76.6	70.7	80.35
	300D	76.6	70.53	80.36
Graph Neural Network with FastText on Anatomy dataset	100D	76.6	77.0	80.7
	200D	76.6	80.5	80.8
	300D	76.6	81.0	81.0

5.2 Analysis of various testcases from Anatomy dataset

During the prediction process, we analyze different test cases to understand the effectiveness of the meta-similarity of the ontology alignment system. Table 14 shows the ranking of the actual corresponding source class in word similarity, meta similarity, and combined similarity against the target class. Word Similarity denotes how two classes/concepts are syntactically similar, on the other hand, meta similarity denotes how two classes/concepts have similar meta information such as parent class, child class, equivalent class, disjoint class, and restrictions. We present the following 14 test cases with examples. In Case-1, both the word similarity and the meta similarity rank of the actual source class/concept is 1. In the given example, “Pancreatic_Endocrine_Secretion” is most similar to “endocrine pancreas secretion” both syntactically and structurally which leads to correct prediction in combined similarity as well. Case-2,3,4 show that though the word similarity rank of the actual source class/concept is high, however, the meta similarity rank is low thus for Case-3,4, the ontology alignment system fails to detect the correct similar source class/concept. Case-8,9,10 show the effectiveness of the ontology system where high meta similarity rank compensates the low rank of word similarity which results in predicting correct similar source class/concept. Case-5,6,7 show both the word similarity and the meta similarity ranks are low which depicts that the ontology alignment system does not compensate the word similarity low rank with the meta similarity and fails to predict correctly similar source class/concept. Case-11,12,13,14 show the high meta similarity rank is not enough to balance the word similarity low rank and results in incorrect final prediction.

Table 14: Different case studies in the prediction of the proposed ontology alignment system with the ranking of the actual source class/concept based on word similarity, meta similarity, and combined similarity for class/concept vector dimensions 300, number of predictions k=1, and similarity threshold 0.96

Case-No	Class/Concept	Word Similarity Rank	Meta Similarity Rank	Combined Similarity Rank
Case-1	endocrine pancreas secretion (Target Class/Concept)	1	1	1
	Pancreatic_Endocrine_Secretion (Source Class/Concept)			
Case-2	right lung terminal bronchiole (Target Class/Concept)	1	3	1
	Right_Lung_Terminal _Bronchiole (Source Class/Concept)			
Case-3	intrahepatic part of left hepatic duct (Target Class/Concept)	1	5	3
	Intrahepatic_Portion_of_the_ Left_Hepatic_Duct (Source Class/Concept)			
Case-4	small intestine muscularis mucosa (Target Class/Concept)	1	4	2
	Small_Intestinal_Muscularis _Mucosa (Source Class/Concept)			
Case-5	tongue skeletal muscle (Target Class/Concept)	2	5	2
	Tongue_Muscle (Source Class/Concept)			

Table 15: Continuation of Table 14

Case-No	Class/Concept	Word Similarity Rank	Meta Similarity Rank	Combined Similarity Rank
Case-6	caudal auricular vein (Target Class/Concept)	2	4	3
	Posterior_Auricular_Vein (Source Class/Concept)			
Case-7	retina inner nuclear layer (Target Class/Concept)	2	2	2
	Inner_Nuclear_Layer (Source Class/Concept)			
Case-8	foot interosseus muscle (Target Class/Concept)	2	1	1
	Foot_Interosseous_Muscle (Source Class/Concept)			
Case-9	pancreas secretion (Target Class/Concept)	2	1	1
	Pancreatic_Secretion (Source Class/Concept)			
Case-10	esophagus muscularis mucosa (Target Class/Concept)	2	1	1
	Esophageal_Muscularis_Mucosa (Source Class/Concept)			
Case-11	mammary gland epithelium (Target Class/Concept)	3	2	2
	Mammary_Epithelium (Source Class/Concept)			
Case-12	fourth ventricle (Target Class/Concept)	4	2	2
	Fourth_Ventricle_of_the_Brain (Source Class/Concept)			
Case-13	proximal convoluted tubule (Target Class/Concept)	3	2	3
	Proximal_Convoluted_Tube (Source Class/Concept)			
Case-14	mammary gland epithelium (Target Class/Concept)	3	2	2
	Mammary_Epithelium (Source Class/Concept)			

Chapter 6

COMPARISON WITH STATE-OF-THE-ART ALGORITHMS

The results of OntoConnect, an Ontology alignment system, in the Ontology Alignment Evaluation Initiative (OAEI) 2020 campaign are reported in this chapter. OntoConnect is a domain-independent schema alignment system that combines syntactic similarity and structural similarity between classes/concepts to align the classes/concepts from the source and target ontologies. This chapter describes the participation of OntoConnect at OAEI 2020 and discusses its methodology and results on the Anatomy dataset.

Table 16 shows the comparison between OntoConnect system with Other Matcher systems who participated in OAEI ("OAEI" 2020) Anatomy dataset. OAEI is an ontology alignment evaluation initiative that campaigns aiming at evaluating ontology matching technologies.

6.1 Comparative Result analysis of the state-of-the-art systems

We have tested OntoConnect on the Anatomy (**anatomy**) data set published by OAEI with different parameters such as input vector dimension and similarity threshold. Three different files are provided in the OAEI System: source ontology, target ontology, and result or alignment file. Standard evaluation metrics, i.e., precision, recall, and F-measure are used. The OntoConnect system yields satisfactory results with a precision of 99.6%, recall of 66.5%, and F-measure of 79.7% for a similarity threshold of 0.99 with the 100-dimension input vector.

Table 16: Performance comparison between OntoConnect with Other Matcher systems in OAEI-2020 Anatomy dataset

Matcher	Precision	Recall	F-Measure	Runtime	Size
AML	0.956	0.927	0.941	29	1471
Lily	0.901	0.902	0.901	706	1517
LogMapBio	0.885	0.902	0.893	1005	1544
LogMap	0.918	0.846	0.88	7	1397
Wiktionary	0.956	0.753	0.842	65	1194
ALIN	0.986	0.72	0.832	1182	1107
LogMapLite	0.962	0.728	0.828	2	1147
OntoConnect	**0.996**	**0.665**	**0.797**	**248**	**1012**
ATBox	0.987	0.671	0.799	192	1030
ALOD2Vec	0.83	0.768	0.798	236	1403
StringEquiv	0.997	0.622	0.766	-	946
DESKMatcher	0.472	0.623	0.537	391	2002

From the table, we can observe that the state of art tools such as AML (Faria et al. 2013), LogMap (Jiménez-Ruiz and Grau 2011) are producing high F-measure on the Anatomy dataset. Most of these state of art tools are rule-based and use external knowledge sources for the ontology alignment process. On the other hand, the OntoConnect tool does not use any domain knowledge or any other external knowledge. The main aim of the OntoConnect system was to eliminate or reduce the domain expert intervention.

From the table 16, we can claim that though our proposed OntoConnect system did not surpass the other tools in terms of F-measure the performance of the OntoConnect system is comparable with the other state-of-the-art tools.

6.2 Analysis of the results

The main goal of OntoConnect is to address questions such as, (i) can ontology alignment be done independently of domain information? (ii) Can ontology alignment be achieved by using only the meta-information and structural information of ontologies? (iii) Can ontology alignment be achieved using unsupervised machine learning instead of the traditional rule-based approaches? The OntoConnect tool is able to address all the above questions and moreover, it performs well compared to some of the state-of-the-art systems in OAEI 2020. The main strength of the tool is that a domain-independent approach is performed by achieving the mentioned goals.

Besides the strengths of the tool, there is a number of potential improvements to be realized for OntoConnect. The main weakness of the OntoConnect tool is the complex architecture of the system, as it has two different components of different languages, i.e., java and python. It was difficult to incorporate any OAEI evaluation wrapper because of the complex architecture of the tool. We have used Docker to execute the system on the HOBBIT (Röder, Kuchelev, and Ngonga Ngomo 2019) platform but there is still room for improving the system architecture so that the tool can be easily executed. The second problem is the size of the project. We have used the pre-trained model fastText in the system and the default dimension of the fastText output vector is 300. The high dimension of the vector causes an increase in the size of the tool. In future work, we would like to explore different procedures such as the autoencoder approach to reduce the dimension to minimize the size of the tool.

6.3 Conclusion

In this study, the OntoConnect tool is presented with a generic and domain-independent approach to align multiple ontologies that eliminate cumbersome and error-prone manual work. A non-linear neural network is used for feature extraction from the source ontology and is independent of the domain knowledge. Participating in this campaign for the first time allowed us to see how the OntoConnect system was performing compared to the other tools. It was seen that our tool had a high precision among the tools without any domain knowledge and without depending on any vocabularies. But both recall and F1 have room to improve. Even though OntoConnect has a reasonable runtime, we would like to decrease the execution time for better performance. We have seen that our tool is comparable to the current state-of-the-art domain-specific approaches and we would like to participate in other tracks next year to see the results in different domains.

Chapter 7

ONTOCONNECT ARCHITECTURE

This section describes the architecture of the OntoConnect system. Our approach follows a microservices architecture, consisting of different components that work together. The main motivation behind using microservices was to isolate different tasks and use some of the existing modules within our project. This allowed the use of different programming languages for different purposes based on their applicability. We divided our project into two different microservices:

1. A Java microservice, which allowed the use of the OWL API and HermiT Reasoner.
2. A Python microservice, which allowed us to use the FastText model to perform complex text analysis, develop the neural network model, and calculate the similarity between the source and target ontologies.

Each microservice was dockerized, making it modular, portable, as well as isolating the environments so as to run on any operating system. Figure 27 shows a high-level system architecture of OntoConnect.

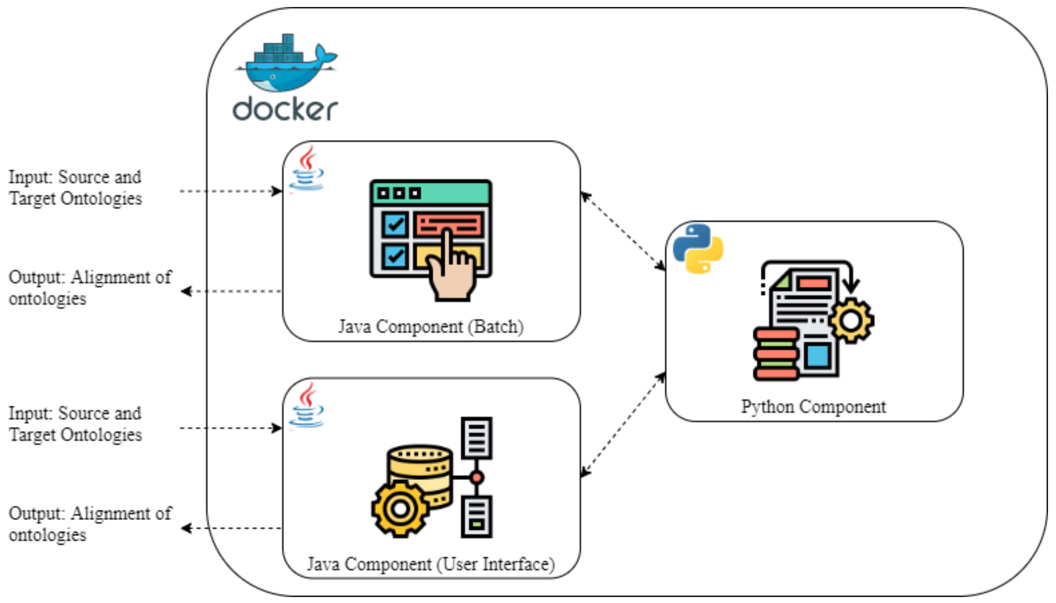

docker
Input: Source and Target Ontologies
Output: Alignment of ontologies
Java Component (Batch)
Input: Source and Target Ontologies
Output: Alignment of ontologies
Java Component (User Interface)
Python Component

Figure 27: Ontology Alignment System Architecture

7.1 OntoConnect Components

The first component is the Java microservice, which reads and parses the ontologies passed to it by the angular microservice. The Java microservice performs many different tasks, the first of which is information extraction. First, we extract information from these ontologies using the OWL API and HermiT Reasoner. In this process, we have considered the following information about a class: Label of the class, IRI of the class, Parent classes of the class, Child classes of the class, Equivalent classes of the class, Disjoint classes of the class, and Restriction (such as "part of"). Finally, all the information is written into JSON files, which are used by the next microservice, i.e., the Python microservice for the linking between the source and target ontology.

The second component is the Python microservice, which is responsible for the network training and alignment process. First, it modifies the labels for each class, object property, and relations by using simple techniques such as stop words removal, stemming and lemmatization, expanding abbreviations, and then, creates a label dictionary. Using this dictionary, it then generated 300D vectors for each ontological entity of the source and target ontology using a pre-trained FastText model. It is used for training the recursive neural network with the source ontology entities. Once completed, the trained neural network predicts a similar source ontology entity for each target ontology entity. The output containing a set of alignments is written to an RDF file. The output contains a set of alignments ($< e, e', r, n >$). Here the "e" is the class from the source ontology, "e'" is the class from the target ontology, "r" is the relation between "e" and "e'". In this project, we have only considered equivalence ("=") relation. n is the similarity score between "e" and "e'". Once this process is finished, the Python microservice generates the RDF file.

7.2 OntoConnect Software Specification

The OntoConnect system was tested on a Unix system with 16 GB RAM and 100GB of disk space. The major constraint here was with memory, as FastText data is loaded into memory at once before it can be used. Java 1.8, python 3.6, and related modules/packages were used to develop the entire application. PyTorch v1.7 was used to develop the neural network, FastText was used for word vector generation, Apache Jena was used for ontology file parsing, and tools like Apache Tomcat and Maven were used for compiling, building, and serving parts of the code. Finally, Docker v2.0.0 was used to isolate and containerize the different services, so as to establish a consistent running environment regardless of the host operating systems and different existing versions of programming languages (Java and Python) on the host. Docker-compose was used to orchestrate the communication and interaction between the different microservices, and it also made it easier for end-users to start the application with a single command, without worrying about the finer details within the application.

7.3 Adaptations made for the evaluation

We have tried to test the OntoConnect system on Semantic Evaluation At Large Scale (SEALS) (Wrigley, García-Castro, and Nixon 2012) platform, however, were not able to run the system as SEALS only provides a wrapper for java-specific tools only. Other frameworks such as MELT (Hertling, Portisch, and Paulheim 2019) was also tried for the evaluation of the OntoConnect System, however, MELT provides an evaluation wrapper for either java-only tools or python-only tools. It does not support tools that have both java and python components in one. OntoConnect system uses both the java and python components. Hobbit platform (Röder, Kuchelev, and Ngonga Ngomo 2019) permits dockerized tool which is independent of the type of the programming language of the tool. For this reason, the dockerized approach is used to build the OntoConnect System and we could successfully test and evaluate it on the Hobbit platform.

Chapter 8

FUTURE WORK

In the future, we aim to complete the following tasks to enhance the OntoConnect System.

First, the ontology tool *OntoConnect* will be tested on different ontologies from different domains. We have planned to test on Large Bio Medical ontology ("LargeBIO" 2020). The main purpose of choosing Large Bio Medical ontology is that it contains a large number of individuals and the structure of this ontology is complex compared to other ontologies. The experiment on Large Bio Medical ontology will help analyze the efficiency and robustness of the *OntoConnect* tool for both large and complex datasets.

Second, we will address the complexities due to the existential restrictions present in the Large Bio Medical ontology. One such existential restriction is shown in Figure 28 which depicts the complex relation of class/-concept "Rat Benign Basal Cell Tumor" is sub-class-of (ObjectSomeValuesFrom ("EO_Disease_Has_Associated_EO_Anatomy", "Rat Forestomach"). To address this issue, we will follow the process defined by (Chen et al. 2020) to convert the OWL ontology described $\mathcal{O}$ into a graph $\mathcal{G}$ in RDF form. where the existential restriction of a class/concept "Rat Benign Basal Cell Tumor" in Figure 28 will be transformed into the following four triples:

(i) <nci:Rat Benign Basal Cell Tumor, rdfs:subClassOf, _:x>,

(ii) <_:x, rdf:type, owl:Restriction>,

(iii) <_:x, ObjectSomeValuesFrom, EO_Disease_Has_Associated_EO_Anatomy>,

and

(iv) <_:x, owl:onProperty, Rat Forestomach>.

A random graph walk will be applied to extract corpus for a class/concept which includes the structural and lexical information of the class/concept.

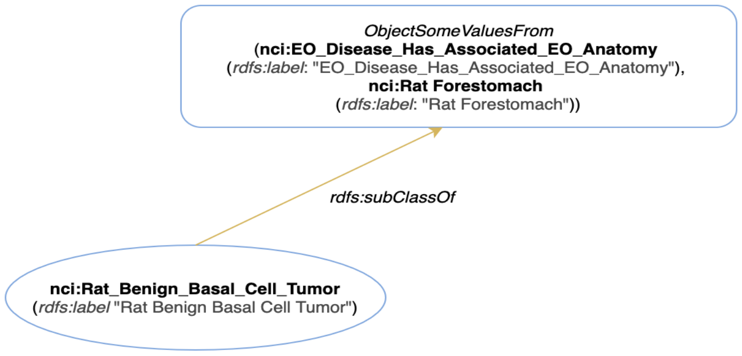

ObjectSomeValuesFrom
(nci:EO_Disease_Has_Associated_EO_Anatomy
(*rdfs:label*: "EO_Disease_Has_Associated_EO_Anatomy"),
nci:Rat Forestomach
(*rdfs:label*: "Rat Forestomach"))
rdfs:subClassOf
nci:Rat_Benign_Basal_Cell_Tumor
(*rdfs:label* "Rat Benign Basal Cell Tumor")

Figure 28: Fragments of National Cancer Institute Thesaurus - NCI (Large Bio Medical Ontology)

Third, refinement of the word-vectors for each entity will be performed. In our approach, the learned word vectors from the pre-trained FastText and BERT model reflect the word frequency instead of meaning which is undesirable for the ontology alignment. To overcome this, the authors exploit the synonyms and antonyms of a class/concept to learn semantic word representations which are more suitable candidates for ontology alignment. The authors extract antonym and synonym relations similar to approach taken by Kolyvakis et al. (Kolyvakis, Kalousis, and Kiritsis 2018) from different thesaurus such as Wordnet (Oram 2001), PPDB 2.0 (Pavlick et al. 2015), etc. The main idea is if two entities are synonyms then their vectors should be close in the vector space model and if they are antonyms then they should be away from each other.

Chapter 9

CONTRIBUTIONS & CONCLUSION

9.1 Contributions

The research objective of this study is to perform domain-independent ontology mapping without the need for human intervention by exploiting meta information of the ontological classes. Existing ontology alignment approaches customize their techniques using domain knowledge thereby limiting their use only for specific domains. The novelty of our approach is in the use of an unsupervised neural network approach with a pre-trained word embedding model that comprises words from the English language making it generic and can be used with any data sets without human intervention. Experimental results show that our approach, in spite of being domain-independent, gets close to current domain-specific approaches in terms of precision and accuracy. The main contributions of this dissertation can be summarized as follows:

(i) *OntoConnect* system that generates a model based on a given source ontology in any domain and a pre-trained word embedding model that comprises vector representation for all words in the English language.

(ii) Using the trained model and given a target ontology for alignment with a specific source ontology, the *OntoConnect* system predicts the alignments between ontological concepts by computing the word similarity (syntactic) between all concepts, meta similarity (structural) between all classes/concepts, and a combined similarity that is based on both word and meta similarities.

(iii) A comprehensive evaluation, in terms of precision, recall, and F-measure of our approach is presented using Anatomy dataset along with a reference ground truth provided by the Ontology Alignment Evaluation Initiative (OAEI) and comparison with other state-of-the-art systems.

(iv) Detailed analysis and hyperparameter tuning using different similarity thresholds and vector dimensions are performed in order to obtain the best results that are close to that of state-of-the-art domain-specific approaches.

9.2 Conclusion

Given the availability of huge hierarchical domain-specific data, automated ontology mapping is still a complex problem. There are many proposed approaches available, but most of them require human intervention and are heavily domain-specific. In this study, we have presented two unsupervised learning techniques. The first one is a generic and domain-independent approach to align multiple ontologies that eliminate cumbersome and error-prone manual work. A non-linear neural network is used for feature extraction from the source ontology and is independent of the domain knowledge. The implemented novel ontology alignment method is based on an unsupervised machine learning technique that incorporates a combination of word similarity and meta similarity thereby taking advantage of both syntactic and structural information about ontological concepts. The experiments show that it can produce high precision results without any domain expert intervention. On the other hand, we have also explored the graph embedding technique. The experiments also show that the graph embedding technique can be applied to find entity embedding which captures its meta-information in an unsupervised way. From our experiment, we can observe both the techniques are yielding high accuracy which is comparable with the current state-of-the-art domain-specific approaches.

REFERENCES

Aumueller, David, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. 2005. "Schema and ontology matching with COMA++." In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data,* 906–908.

Bodenreider, Olivier, Terry F Hayamizu, Martin Ringwald, Sherri De Coronado, and Songmao Zhang. 2005. "Of mice and men: Aligning mouse and human anatomies." In *AMIA Annual Symposium Proceedings,* 2005:61. American Medical Informatics Association.

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. "Enriching word vectors with subword information." *Transactions of the Association for Computational Linguistics* 5:135–146.

Bordes, Antoine, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. "Translating embeddings for modeling multi-relational data." In *Neural Information Processing Systems (NIPS),* 1–9.

Bühmann, Lorenz, Jens Lehmann, Patrick Westphal, and Simon Bin. 2018. "DL-learner structured machine learning on semantic web data." In *Companion Proceedings of the The Web Conference 2018,* 467–471.

Chen, Jiaoyan, Pan Hu, Ernesto Jimenez-Ruiz, Ole Magnus Holter, Denvar Antonyrajah, and Ian Horrocks. 2020. "OWL2Vec*: Embedding of OWL Ontologies." *arXiv preprint arXiv:2009.14654.*

Chen, Jiaoyan, Ernesto Jiménez-Ruiz, Ian Horrocks, Denvar Antonyrajah, Ali Hadian, and Jaehun Lee. 2021. "Augmenting Ontology Alignment by Semantic Embedding and Distant Supervision."

Chinea, Alejandro. 2009. "Understanding the principles of recursive neural networks: a generative approach to tackle model complexity." In *International Conference on Artificial Neural Networks,* 952–963. Springer.

Crestani, Fabio, and Cornelis J van Rijsbergen. 1995. "Information retrieval by logical imaging." *Journal of Documentation* 51 (1): 3–17.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805.*

Doan, AnHai, Alon Halevy, and Zachary Ives. 2012. *Principles of data integration.* Elsevier.

Doan, AnHai, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. 2003. “Learning to match ontologies on the semantic web.” *The VLDB journal* 12 (4): 303–319.

Duchi, John, Elad Hazan, and Yoram Singer. 2011. “Adaptive subgradient methods for online learning and stochastic optimization.” *Journal of machine learning research* 12 (7).

Faria, Daniel, Catia Pesquita, Emanuel Santos, Matteo Palmonari, Isabel F Cruz, and Francisco M Couto. 2013. “The agreementmakerlight ontology matching system.” In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems",* 527–541. Springer.

Giunchiglia, Fausto, Pavel Shvaiko, and Mikalai Yatskevich. 2004. “S-Match: an algorithm and an implementation of semantic matching.” In *European semantic web symposium,* 61–75. Springer.

Gracia, Jorge, Jordi Bernad, and Eduardo Mena. 2011. “Ontology matching with CIDER: evaluation report for OAEI 2011.” *Ontology Matching,* 126.

Hertling, Sven, Jan Portisch, and Heiko Paulheim. 2019. “Melt-matching evaluation toolkit.” In *International Conference on Semantic Systems,* 231–245. Springer, Cham.

Hochreiter, Sepp. 1998. “The vanishing gradient problem during learning recurrent neural nets and problem solutions.” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (02): 107–116.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. “Long short-term memory.” *Neural computation* 9 (8): 1735–1780.

Horridge, Matthew, and Sean Bechhofer. 2011. “The owl api: A java api for owl ontologies.” *Semantic web* 2 (1): 11–21.

“OAEI.” 2020. *http://oaei.ontologymatching.org/.*

“LIBRARY.” 2013. *http://oaei.ontologymatching.org/2013/library/.*

“Anatomy.” 2013. *http://oaei.ontologymatching.org/2019/anatomy/index.html.*

“FMA.” 2020. *http://si.washington.edu/projects/fma.*

"LargeBIO." 2020. *http://www.cs.ox.ac.uk/isg/projects/SEALS/oaei/.*

"NCI." 2020. *https://ncit.nci.nih.gov/ncitbrowser/.*

"TheSoz." 2013. *https://www.gesis.org/en/home.*

"SNOMED." 2020. *https://www.snomed.org/.*

"STW." 2013. *https://zbw.eu/stw/version/latest/about.*

Hu, Wei, Jianfeng Chen, and Yuzhong Qu. 2011. "A self-training approach for resolving object coreference on the semantic web." In *Proceedings of the 20th international conference on World wide web,* 87–96.

Jiang, Jay J, and David W Conrath. 1997. "Semantic similarity based on corpus statistics and lexical taxonomy." *arXiv preprint cmp-lg/9709008.*

Jiménez-Ruiz, Ernesto, and Bernardo Cuenca Grau. 2011. "Logmap: Logic-based and scalable ontology matching." In *International Semantic Web Conference,* 273–288. Springer.

Kipf, T N, and Max Welling. 2016. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907.*

Kipf, Thomas N, and Max Welling. 2016. "Variational graph auto-encoders." *arXiv preprint arXiv:1611.07308.*

Kolyvakis, Prodromos, Alexandros Kalousis, and Dimitris Kiritsis. 2018. "Deepalignment: Unsupervised ontology matching with refined word vectors." In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers),* 787–798.

Lenzerini, Maurizio. 2002. "Data integration: A theoretical perspective." In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems,* 233–246.

Lerer, Adam, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. "Pytorch-biggraph: A large-scale graph embedding system." *arXiv preprint arXiv:1903.12287.*

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

Lin, Dekang, et al. 1998. “An information-theoretic definition of similarity.” In *Icml,* 98:296–304. 1998.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. “Efficient estimation of word representations in vector space.” *arXiv preprint arXiv:1301.3781.*

Motik, Boris, Rob Shearer, and Ian Horrocks. 2007. “Optimized reasoning in description logics using hypertableaux.” In *International Conference on Automated Deduction,* 67–83. Springer.

Ngo, DuyHoa, and Zohra Bellahsene. 2012. “YAM++: a multi-strategy based approach for ontology matching task.” In *International Conference on Knowledge Engineering and Knowledge Management,* 421–425. Springer.

Ngomo, Axel-Cyrille Ngonga, and Sören Auer. 2011. “Limes-a time-efficient approach for large-scale link discovery on the web of data.” *integration* 15 (3).

Oram, Peter. 2001. “WordNet: An electronic lexical database. Christiane Fellbaum (Ed.). Cambridge, MA: MIT Press, 1998. Pp. 423.” *Applied Psycholinguistics* 22 (1): 131.

Pavlick, Ellie, Pushpendre Rastogi, Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2015. “PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification.” In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers),* 425–430.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning. 2014. “Glove: Global vectors for word representation.” In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP),* 1532–1543.

Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. 2014. “Deepwalk: Online learning of social representations.” In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining,* 701–710.

Pesquita, Catia, Cosmin Stroe, Isabel F Cruz, and Francisco M Couto. 2010. “BLOOMS on AgreementMaker: Results for OAEI 2010.” *Ontology Matching,* 134.

Resnik, Philip. 1995. “Using information content to evaluate semantic similarity in a taxonomy.” *arXiv preprint cmp-lg/9511007.*

Röder, Michael, Denis Kuchelev, and Axel-Cyrille Ngonga Ngomo. 2019. "HOBBIT: A platform for benchmarking Big Linked Data." *Data Science,* no. Preprint, 1–21.

Serafini, Luciano, Paolo Bouquet, Bernardo Magnini, and Stefano Zanobini. 2003. "An algorithm for matching contextualized schemas via SAT."

Shvaiko, Pavel, and Jérôme Euzenat. 2011. "Ontology matching: state of the art and future challenges." *IEEE Transactions on knowledge and data engineering* 25 (1): 158–176.

Suchanek, Fabian M, Serge Abiteboul, and Pierre Senellart. 2011. "Paris: Probabilistic alignment of relations, instances, and schema." *arXiv preprint arXiv:1111.7164.*

Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. "Graph attention networks." *arXiv preprint arXiv:1710.10903.*

Wrigley, Stuart N, Raúl García-Castro, and Lyndon Nixon. 2012. "Semantic evaluation at large scale (SEALS)." In *Proceedings of the 21st International Conference on World Wide Web,* 299–302.

Wu, Zhibiao, and Martha Palmer. 1994. "Verb semantics and lexical selection." *arXiv preprint cmp-lg/9406033.*

Xin, Luna Dong. 2015. *Big Data Integration (Synthesis Lectures on Data Management).*

APPENDIX A

ONTOCONNECT LINKS

A.1 Link to the system and parameters file

The OntoConnect code is available on GitHub: https://github.com/dbpedia/linking.

A.2 Link to the set of provided alignments

The OntoConnect result is published on http://oaei.ontologymatching.org/2020/results/anatomy/index.html . The result is also available on GitHub: https://github.com/dbpedia/linking/wiki/Result

A.3 Publications

1. Chakraborty, J., Bansal, S., Yaman, B., Virgili, L., & Konar, K. (2021). Ontoconnect: Unsupervised ontology alignment with recursive neural network. In Proceedings of the 36th ACM/SIGAPP Symposium on Applied Computing, SAC (pp. 22-26).
2. Hu, Y., Chang, S. Y., Hsu, K. R., Chakraborty, J., & Bansal, S. K. (2021, January). Ontology-based Document Recommendation System using Topic Modeling. In 2021 IEEE 15th International Conference on Semantic Computing (ICSC) (pp. 449-454). IEEE.
3. Dileep, A. K., Mishra, A., Mehta, R., Uppal, S., Chakraborty, J., & Bansal, S. K. (2021, January). Template-based Question Answering analysis on the LC-QuAD2. 0 Dataset. In 2021 IEEE 15th International Conference on Semantic Computing (ICSC) (pp. 443-448). IEEE.
4. Chakraborty, J., Yaman, B., Virgili, L., Konar, K., & Bansal, S. K. (2020). OntoConnect: Results for OAEI 2020. Ontology Matching, 204.
5. Chakraborty, J., Thopugunta, G., & Bansal, S. (2018, January). Data extraction and integration for scholar recommendation system. In 2018 IEEE 12th International Conference on Semantic Computing (ICSC) (pp. 397-402). IEEE.
6. Chakraborty, J., Padki, A., & Bansal, S. K. (2017, March). Semantic etl—state-of-the-art and open research challenges. In 2017 IEEE 11th International Conference on Semantic Computing (ICSC) (pp. 413-418). IEEE.
7. Entity Linking, 2019, Google Summer Of Code. (https://summerofcode.withgoogle.com/archive/2019/projects/4837547543363584/)