DBpedia Type and Entity Detection Using Word Embeddings and N-gram Models

by

Hanqing Zhou

Thesis submitted in partial fulfillment of the requirements for the
Master of Computer Science degree

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Nowadays, knowledge bases are used more and more in Semantic Web tasks, such as knowledge acquisition (Hellmann et al., 2013), disambiguation (Garcia et al., 2009) and named entity corpus construction (Hahm et al., 2014), to name a few. DBpedia is playing a central role on the linked open data cloud; therefore, the quality of this knowledge base is becoming a central point of focus. However, there are some issues with the quality of DBpedia. In particular, DBpedia suffers from three major types of problems: a) invalid types for entities, b) missing types for entities, and c) invalid entities in the resources' description. In order to enhance the quality of DBpedia, it is important to detect these invalid types and resources, as well as complete missing types.

The three main goals of this thesis are: a) invalid entity type detection in order to solve the problem of invalid DBpedia types for entities, b) automatic detection of the types of entities in order to solve the problem of missing DBpedia types for entities, and c) invalid entity detection in order to solve the problem of invalid entities in the resource description of a DBpedia entity. We compare several methods for the detection of invalid types, automatic typing of entities, and invalid entities detection in the resource descriptions. In particular, we compare different classification and clustering algorithms based on various sets of features: entity embedding features (Skip-gram and CBOW models) and traditional n-gram features. We present evaluation results for 358 DBpedia classes extracted from the DBpedia ontology.

The main contribution of this work consists of the development of automatic invalid type detection, automatic entity typing, and automatic invalid entity detection methods using clustering and classification. Our results show that entity embedding models usually perform better than n-gram models, especially the Skip-gram embedding model.

Acknowledgement

First and foremost, I offer my sincere gratitude and appreciation to my supervisors, Dr. Amal Zouaq and Dr. Diana Inkpen, for their eminent supervision, creative ideas, and endless support. This work would not have been possible without their help.

Thanks to the NLP group meetings and the accompanied Tamale seminars for their broad knowledge and cutting-edge research ideas. Also, special thanks to my lab colleagues and all of the NLP group members for their help, suggestions and support.

Finally, I would like to thank my parents for their support and encouragement during my time in the Master's program.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

The Semantic Web is defined by Berners-Lee et al. (2001) as an extension of the current Web in which information is given a well-defined meaning, in order to allow computers and people to cooperate better than before. Linked Data is about creating typed links between data from different sources by using the current Web. More precisely, it is a "Web of Data" in Resource Description Format (RDF) (Bizer et al., 2009a).

Knowledge bases represent the backbone of the Semantic Web, and they also represent Web resources (Kabir et al., 2014). Knowledge bases are being created by integration of resources like Wikipedia and Linked Open Data (Syed and Finin, 2011; König et al., 2013) and have turned into a crystallization point for the emerging Web of Data (Hellmann et al., 2009; Moran, 2012).

There are several popular knowledge bases such as DBpedia（Auer et al., 2007), Yago (Fabian et al., 2007) or Wikidata (Vrandečić and Krötzsch, 2014). The DBpedia knowledge base is a rich knowledge base, that uses the DBpedia extraction framework to extract structured information from Wikipedia. It contains several properties, such as title and abstract from Wikipedia in around a hundred different languages (Mendes et al., 2012; Lehmann et al., 2015). The YAGO knowledge base is automatically extracted from Wikipedia and WordNet using combined rule-based and heuristic methods. Each Wikipedia article becomes an entity and each Wikipedia category becomes a type in the YAGO knowledge base (Fabian et al., 2007; Suchanek et al., 2007; Suchanek et al., 2008). Wikidata knowledge base is a multilingual "Wikipedia for data", it provides a common resource of data for Wikipedia, which aims to manage the factual information of the popular online encyclopedia (Vrandečić and Krötzsch, 2014; Erxleben et al., 2014).

Knowledge bases have various applications on the Semantic Web, such as entity linking (Shen et al., 2012), which links entities in-text with Wikipedia and the WordNet knowledge base (Krötzsch et al., 2007; Fellbaum, 2010). Another important application is leveraging the rich semantic knowledge embedded in Wikipedia. Question answering on the semantic web (Yih et al., 2016) uses semantic similarity from the Freebase knowledge base (Bollacker et al., 2007; Bollacker et al., 2008; Yao and Van Durme, 2014) and reaches high accuracy. Furthermore, semantic similarity measurement is another application for knowledge bases; For example, (Li et al., 2012) combines

knowledge bases with neural networks to simulate human process of similarity perception to measure semantic similarities.

Among the main knowledge bases on the Semantic Web, the DBpedia knowledge base represents structured information from Wikipedia, describes millions of entities, and it is available in more than a hundred languages (Morsey et al., 2012). DBpedia uses a unique identifier to name each entity (i.e., Wikipedia page) and associates it with an RDF description that can be accessed on the Web via the URI dbr: <http://dbpedia.org/resource/> (Lehmann et al., 2015; Auer et al., 2007). Similarly, DBpedia is based on an ontology, represented in the namespace dbo:< http://dbpedia.org/ontology/>, that defines various classes that are used to type available entities (resources). DBpedia refers to and is referenced by several datasets on the LOD and it is used for tasks as diverse as semantic annotation (Zhang et al., 2013), knowledge extraction (Alani et al; 2003), and information retrieval (Keong and Anthony, 2013). As shown in figure 1.1[1](Abele et al., 2017), DBpedia is a central data hub on the Semantic Web, and more specifically on the Linked Open Data (LOD) cloud.

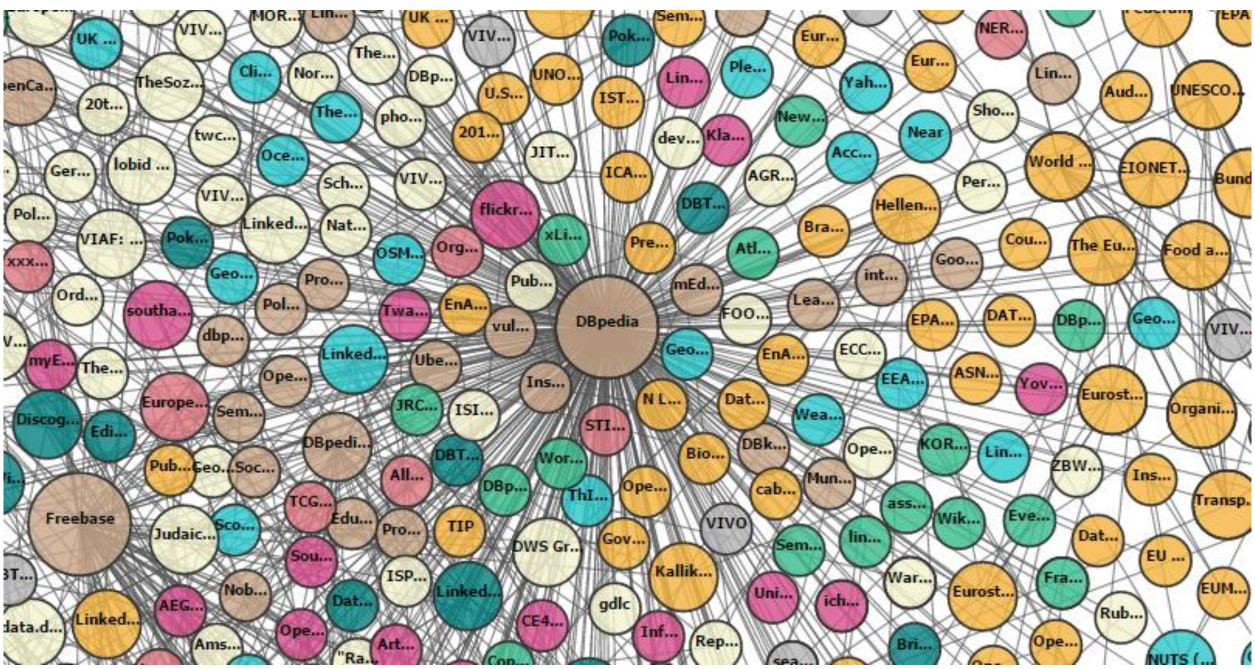

Figure 1. 1. Part of Linked Open Data diagram (Abele et al., 2017)

Given the automatic extraction framework of DBpedia along with its dynamic nature, several inconsistencies can exist in DBpedia (Töpper et al., 2012; Sheng et al., 2012, Fleischhacker et al.,

[1] Linked Open Data Diagram retrieved from http://lod-cloud.net/

2014; Font et al., 2015). One important quality problem is related to DBpedia types (classes). In fact, there exists invalid DBpedia types for some entities. For example, the DBpedia entity "dbr:Xanthine [2] " belongs to three DBpedia types: "dbo:ChemicalSubstance [3] " "dbo:ChemicalCompound," and "dbo:Airport," though "dbo:Airport" is an invalid type. Furthermore, given the size of DBpedia, type information is sometimes unavailable for some entities. Several entities are still un-typed, or not typed with all the relevant classes from the DBpedia ontology. For instance, one example of the missing type problem is the absence of the types "dbo:Politician" and "dbo:President" for the entity "dbr:Donald_Trump." The only available types are "dbo:Person" and "dbo:Agent" which are not incorrect, but they are not specific enough. As well, another problem is that some of the available ontology classes still do not have any instances, such as "dbo:ArtisticGenre," "dbo;TeamSport," and "dbo:SkiResort." Finally, invalid DBpedia entities may exist in the RDF description of an entity due to erroneous triples (Wienand and Paulheim, 2014). Consequently, some entities are not correctly linked to other entities. In terms of invalid entity in a resource description, the DBpedia entity "dbr:Earthquake" is described with several triples, where the triple <dbr:Vanilla_Ice dbo:genre dbr:Earthquake> links "dbr:Earthquake" to "dbr:Vanilla_Ice." We can see that this is an erroneous fact in the description of the entity "dbr:Earthquake." Thus "dbr:Vanilla_Ice" is an invalid entity in the resource description of "dbr:Earthquake."

Identifying these invalid types and entities manually is unfeasible. Similarly, the automatic enrichment and update of DBpedia with new type statements (through *rdf:type*) is thus becoming an important research avenue (Kliegr and Zamazal, 2014; Kliegr, 2015). In this thesis, we rely on vector-based representations such as word embeddings and entity embeddings (Hu et al., 2015; Chen et al., 2016) to reach these objectives.

Word embeddings are vector representations of word(s) (Mikolov et al., 2013a; Mikolov et al., 2013b; Mikolov et al., 2013c) that have several applications in Natural Language Processing tasks, such as entity recognition (Seok et al., 2016), information retrieval (Ganguly et al., 2015), and question answering (Zhou et al., 2015). Entity embeddings, as defined in this thesis, are similar to word embeddings (Mikolov et al., 2013a; Mikolov et al., 2013b; Mikolov et al., 2013c) but they

[2] http://dbpedia.org/resource/Xanthine
[3] http://dbpedia.org/ontology/ChemicalSubstance

differ in that they are based on vectors that describe *entity URIs* instead of words (Hu et al., 2015; Chen et al., 2016).

In this thesis, we aim at addressing the following research questions:

RQ1: Can entity embeddings help detect the relevant types of an entity?

RQ2: How do entity embeddings compare with traditional n-gram models for type identification?

RQ3: Can entity embeddings help detect the relevant entities in the RDF description of a DBpedia resource?

RQ4: How do entity embeddings compare with traditional n-gram models for invalid entity detection?

1.1 Motivation and Contributions

Our primary motivation is to contribute to the improvement of the quality DBpedia by automatically typing entities and identifying erroneous types and entities. Our secondary motivation for this thesis consists of building and experimenting with different entity embedding models for the automatic type identification, automatic typing, and automatic entity identification tasks. The third motivation is to compare entity embedding models with various n-gram models for type and entity identification.

In this thesis, we have the following contributions:

- We build different n-gram models from DBpedia entities' abstracts.
- We use machine learning techniques coupled with entity embeddings models to detect invalid types, complete missing types, and detect invalid entities to enhance the quality of DBpedia.
- We find new entities for each DBpedia type, find new types for each DBpedia entity, and find invalid triples based on the whole DBpedia knowledge base.
- We compare the performance of entity embedding models with traditional n-gram models in terms of DBpedia invalid type detection and invalid DBpedia entity detection.

1.2 Outline

In chapter 1, we discuss the motivation of this work, our goals and contributions. Chapter 2 presents background information about Machine Learning and Data Mining techniques that are used in this thesis. It also discusses the related work about the Semantic Web, Open Linked Data, DBpedia, entity embeddings, and n-gram models. In chapter 3, we discuss our work for collecting datasets, building entity embedding and n-gram models. Then, in Chapter 4, we propose different clustering and classification methods to detect invalid DBpedia types, and complete missing types for 358 types from the DBpedia ontology. Parts of this chapter was published in (Zhou et al., 2017). In Chapter 5, we present our approach to detect invalid DBpedia resources using our own entity embedding and n-gram models. In chapter 6, we conclude our work and contributions, and we discuss possible directions of work for the future.

Chapter 2: Background and Related Work

This chapter starts with a description of the context of this work, including a description of the Semantic Web and Linked Open Data knowledge bases. We then introduce some of the techniques used to achieve our goals including word embeddings, entity embeddings, n-gram models and different machine learning algorithms. Finally, this chapter describes the state of the art in the two main challenges tackled in this thesis: DBpedia Entity Type Detection (Chapter 4) and Invalid Entity Detection in Resource Descriptions (Chapter 5).

2.1 The Semantic Web and Linked Open Data

The emergence of the Linked Open Data cloud (LOD) has resulted in an openly available "Web of Data" which contains billions of RDF triples (Jain et al., 2010; Jain et al., 2012). In fact, the LOD is the current concretization of the Semantic Web; the linked data cloud is used to represent data models on the Semantic Web (Rodriguez 2009; Van Hooland et al., 2012). Knowledge bases can be used to store different structured or unstructured information (Fagerberg et al., 2012). For every Wikipedia page in the context of the Semantic Web, there is a Uniform Resource Identifier (URI) that has been created to identify each DBpedia entity (Mendes et al., 2012). Resources are real-world things that are described on the semantic Web, they are meaningful to machines with descriptions in common data formats (Debattista et al., 2016). Knowledge bases consist of definitional, descriptive, and relevant information for each resource. They can be considered as an encyclopedia for resources, and these resources are sets of interrelated facts in the form of triples <subject, predicate, object> generally in the Resource Description Framework (RDF) format (Zheng et al., 2010). There are many applications of knowledge bases, such as city planning (Simmie and Lever, 2002), recommender systems (Zhang et al., 2016), and teaching (Paulien et al., 2001) to name a few.

Knowledge bases on the Semantic Web such as DBpedia (Bizer et al., 2009b), Yago (Suchanek et al., 2007) and Wikidata (Tanon et al., 2016) for instance, often rely on an ontology. An ontology is defined as a conceptual structure that provides a key to machine-processable data. It also serves

as metadata schema to provide a controlled vocabulary of concepts (Maedche and Staab, 2001; Staab and Studer, 2009). The Web Ontology Language (OWL) is used in the Semantic Web to enable efficient representation of ontologies, which is amenable to decision procedures and reasoning (Shadbolt et al., 2006).

As previously indicated, one of the central resources on the Linked Open Data is DBpedia. It is a multilingual cross-domain knowledge base that contains RDF descriptions of concepts in about a hundred languages (Mendes et al., 2012). The quality and evolution of this knowledge base is currently a hot topic (Mayfield and Finin, 2012; Paulheim and Bizer, 2014; Wienand and Paulheim, 2014; Font et al., 2017).

In order to enhance the quality of DBpedia, the first two goals of this thesis are: building a system based on machine learning techniques that can detect invalid or irrelevant types, and complete missing types automatically. The third goal of this thesis is to build a system based on machine learning techniques that can identify invalid entities automatically.

2.2 Word and Entity Embeddings

Among the models explored in the thesis are word embedding and entity embedding models. Word embeddings are vector representations of word(s) (Mikolov et et al., 2013a) and have been successfully applied to several natural language processing (NLP) tasks.

The first successful word embedding model was the Neural Probabilistic Language Model (Bengio et al., 2003; Bengio et al., 2006), which learned distributed representations of each word and probability functions for word sequences. One of the most popular word embedding tools is Word2vec (Mikolov et al., 2013a; Mikolov et al., 2013b; Mikolov et al., 2013c; Goldberg and Levy, 2014). We use one of the variations of Word2vec, which is called Wiki2vec[4]. It can generate DBpedia entities based on Wikipedia hyperlinks. Our goal is to learn vector representations of DBpedia entities *(entity embeddings*) by using the information available in the corresponding Wikipedia pages or abstracts. The main difference is that the vectors describe entity URIs rather than words, which allow us to compute a semantic distance between entities.

There are two major word embedding models in Word2vec (Mikolov et al., 2013b; Mikolov et al., 2013c): the Continuous Bag-of-Words (CBOW) model and the Skip-gram model.

[4] https://github.com/idio/wiki2vec

The CBOW model is related to the feedforward Neural Net Language Model (NNLM) (Bengio et al, 2003) whose architecture consists of four layers: an input layer, a projection layer, a hidden layer, and an output layer. The CBOW model is similar to the feedforward Neural Net Language Model (NNLM) but it removes the non-linear hidden layer for all words. The objective of the CBOW model is to predict a target word based on context words (a word window) in the same sentence.

The difference between the Continuous Bag-of-Words (CBOW) and standard Bag-of-Words (BOW) models is that CBOW uses continuous distributed representations of the context (Mikolov et al., 2013b; Mikolov et al., 2013c). The architecture of the continuous Bag-of-Words model is shown in Figure 2.1.

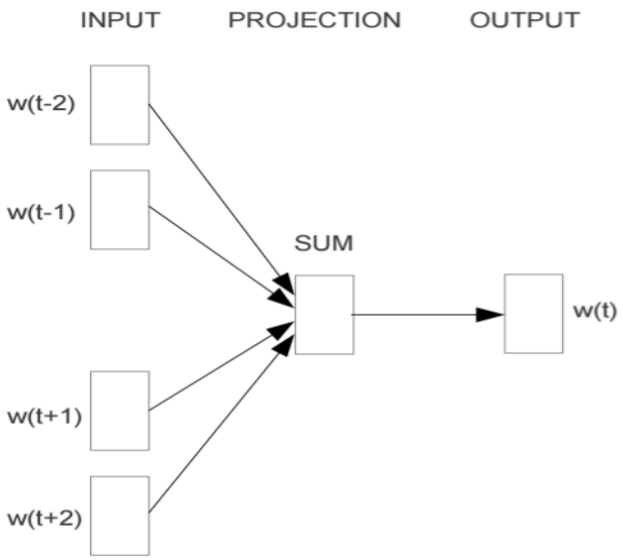

Figure 2. 1. Architecture of Continuous Bag-of-Words model (Mikolov et al., 2013b)

The architecture of the Skip-gram model is similar to that of CBOW, but it predicts the context words given a target word in the same sentence, instead of predicting the current word based on the surrounding words (Mikolov et al., 2013b; Mikolov et al., 2013c). The architecture of the Skip-gram model is shown in Figure 2.2. The Skip-gram model uses a certain skip distance *k* to allow a total of *k* or less skips for the construction of n-gram models (Guthrie et al., 2006).

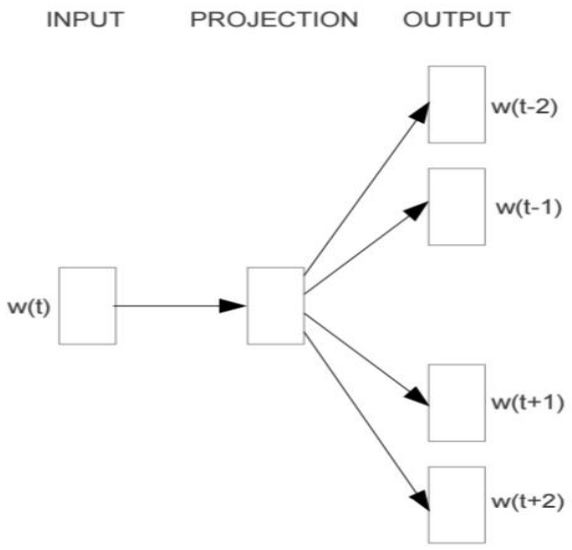

Figure 2. 2. Architecture of Continuous Skip-gram model (Mikolov et al., 2013b)

The literature behind word embeddings describes more ways to induce word representations, inferred from n-gram language modeling rather than continuous language models (Collobert et al., 2011).

2.3 N-gram Models

Generally, language models compute probabilities assigned to sequences of words based on their frequency. An n-gram model, where the *n* represents the number of words to consider in a sequence, assigns probabilities to sequences of words (Roark et al., 2007; Jurafsky and Martin, 2007). N-gram models have been shown to be effective and efficient in several natural language processing tasks. One of our objectives is to compare n-gram models to the entity embedding models in the experiments for invalid type identification and invalid entity detection. For n-gram models, we consider unigrams, bigrams, and trigrams as well as the set of these n-grams together. Our n-gram models are built based on the collection of the DBpedia entities' abstracts. We rely on the measure TF-IDF to weight the n-grams. We also perform Random Forest feature selection (Rogers and Gunn, 2006; Han et al., 2006; Pan and Shen, 2009; Genuer et al., 2010; Genuer et al., 2015) in order to reduce the number of features to limit the dimensions of our vectors to be 1000.

2.4 Machine Learning Techniques

In this section, we present some background information for our experiments and the methods for evaluating our experiments. We exploit several machine learning algorithms namely three clustering algorithms and six classification algorithms.

In the evaluation part, we report the formulas for Precision, Recall, F-score, Accuracy, and Area Under the Curve (AUC) scores.

2.4.1 Clustering

Clustering is the process of partitioning a set of data object into clusters (Han et al., 2012). Clustering algorithms are unsupervised algorithms. In this thesis, we experiment with K-means (Mitchell, 1997; Krishna and Murty; 1999), Mean Shift (Derpanis, 2005), and Birch (Zhang et al., 1996).

The aim of K-means is to divide M points into K clusters in N dimensions to minimize the within-cluster sum of the square error (Hartigan and Wong, 1979). The method used by K-means is detailed by Han et al. (2012):

(1) Choose *k* objects arbitrarily as the initial cluster centers;

(2) Repeat the following steps (3) and (4) until there is no change;

(3) Reassign each object to the cluster based on similarity measure;

(4) Update cluster means with the mean value of all the objects from each cluster.

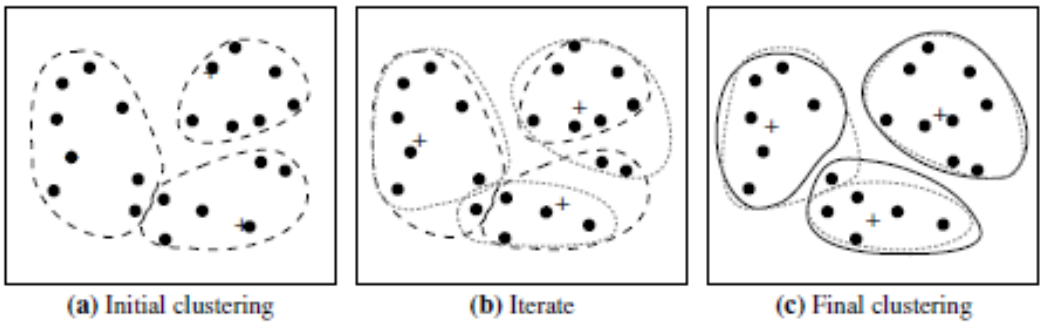

Figure 2. 3. Clustering process (Han et al., 2012)

As shown in the figure 2.3 (a), initially the algorithm selects center points in each cluster as the initial cluster centers. Then the center points change with each iteration as shown in figure 2.3 (b). Finally, in figure 2.3 (c), the center points are stable.

The idea of Mean Shift is to treat the points in the *d*-dimensional feature space as a Probability Density Function (PDF). It performs a gradient ascent procedure to locate the maxima on the local estimated density until convergence (Derpanis, 2005).

The BIRCH clustering algorithm stands for *Balanced Iterative Reducing and Clustering using Hierarchies*, and it uses clustering feature trees to represent cluster hierarchies. This algorithm is generally suitable for large amounts of data (Han et al., 2012; Zhang et al., 1996).

In clustering, detecting outliers is an important issue. There are three major categories of outliers, as mentioned by Han et al. (2012), namely, a global outlier if a data object deviates significantly from the rest of the data objects; a contextual outlier if a data object deviates significantly with respect to a specific context of the object; and collective outliers if several data objects have different characteristics than others, as shown below (figure 2.4).

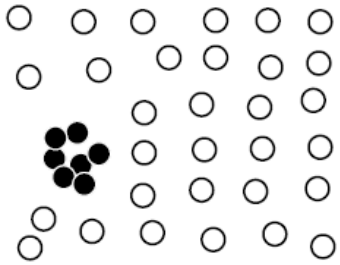

Figure 2. 4. Collective outlier example (Han et al., 2012)

The black objects are collective outliers as shown in figure 2.4, since they have a much higher density than the rest of the data objects.

2.4.2 Classification

Classification is a data analysis technique that builds models to describe each data class. Classification algorithms are supervised learning algorithms (Han et al., 2012). In this work, we relied mainly on seven classification algorithms: Decision Tree (Brown and Myles, 2010), Extra Tree (Geurts et al., 2006), Random Forest (Breiman, 2001), K Nearest Neighbor (KNN) (Guo et al., 2003), Naïve Bayes (Zhang, 2004), and Support Vector Machine (SVM) (Bennett and Bredensteiner, 2000). These algorithms are based on three main paradigms: tree-based algorithms,

ensemble algorithms, and probabilistic algorithms. The tree-based algorithms: Decision Tree Extra Tree, and Random Forest, can be used to track the decision process. Random Forest is also an ensemble algorithm, it can be used to overcome the overfitting problem that decision tree faces. The probabilistic algorithm, Naïve Bayes tends to work well on most of textual data.

In the following paragraphs, we explain the principle of each of the algorithms used in our experiments.

The first tree structure classification algorithm is Decision Tree. Every internal node represents the test on an attribute and each external node (leaf) holds the class label (Apte and Weiss, 1997; Brown and Myles, 2010; Han et al., 2012).

Extra Tree Classifier is one of a tree structure classifiers which implements a meta-estimator to fit several randomized decision trees; then it uses averaging techniques to overcome the over-fitting problem on the training data (Geurts et al., 2006). Extra Tree should perform better than a regular Decision Tree, since Extra Tree has the estimator to generate results that can fit with several decision trees and average the results to overcome the over-fitting problem that a Decision Tree can have.

Another tree-based meta-classifier is Random Forest, which unlike decision tree, is also an ensemble learning method and uses tree predictors. With the increasing number of trees in the forest, the error rate will reach a limit and become stable to overcome the overfitting problem of Decision Tree; this makes Random Forest suitable for high-dimensional data (Breiman, 2001; Chen and Ishwaran, 2012).

The Lazy Learner classifier we use is the K Nearest Neighbor (KNN). It only performs generalization to classify tuples by measuring similarity to the training tuples that are stored beforehand. As a Lazy Learner, KNN first stores training tuples; when it sees the test tuples, it looks for the *k* training tuples that are the closest to the test tuples. The *k* training tuples are the *k*-nearest neighbors of the test tuples (Han et al., 2012; Guo et al., 2003).

The Naïve Bayes classifier is a statistical classifier that is based on the Bayes theorem to compute probabilities of classes, and of features' associated to each class. Naïve Bayes is the simplest of Bayesian networks and the attributes are considered independent of each other (Zhang, 2004) given the class (see figure 2.5).

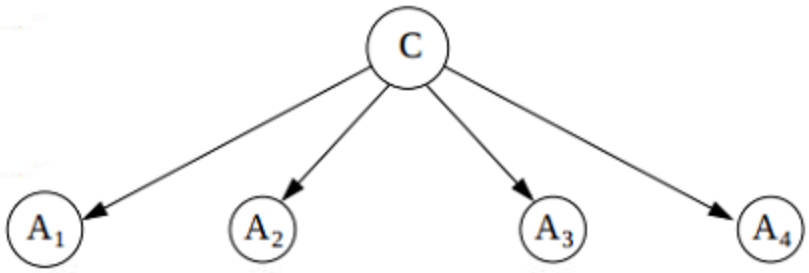

Figure 2. 5. Example of Naïve Bayes (Zhang, 2004)

Support Vector machine (SVM) is a very robust classifier used to classify both linear and nonlinear data. It works as follows: it first transfers the training dataset into a higher dimension via a nonlinear mapping (a kernel). The second step is to search for the linear optimal separating hyperplane within the new dimension by using support vectors and margins (Han et al., 2012; Bennett and Bredensteiner, 2000). As shown in figure 2.6, the margins can be small or big; with increasing margins, the accuracy of results will increase too.

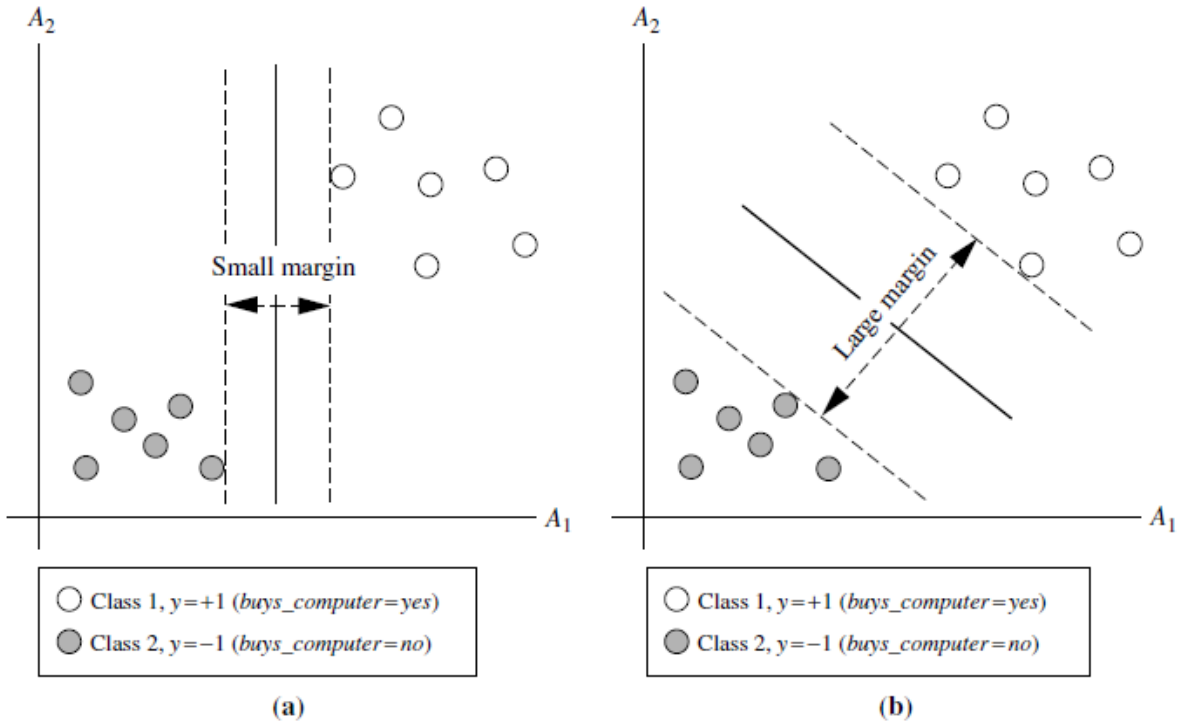

Figure 2. 6. SVM example with small and large margins (Han et al., 2012)

2.5. Related Work

The previous sections described some general background information. This section focuses on the related work for DBpedia and Linked Data quality assessment and enhancement, and automatic type detection.

2.5.1. DBpedia and Linked Data Quality Assessment and Enhancement

Several approaches in the literature aim to enhance the quality of DBpedia and Linked Data. Paulheim (2017) proposed a survey about approaches for knowledge graph refinements, such as methods for detecting invalid DBpedia types, DBpedia invalid relations, and invalid DBpedia knowledge graph interlinks, along with the evaluation results. The authors identified two main problems in DBpedia knowledge base namely the completeness problem and the correctness problems of the DBpedia knowledge base, and they also pointed out several approaches for the DBpedia knowledge base refinements. The survey by Färber et al. (2015) analyzed the most five popular linked data knowledge graphs: DBpedia, Freebase, OpenCyc, Wikipedia, and YAGO, and found these knowledge graphs have the following problems: accuracy, trustworthiness, and consistency. Furthermore, the survey also proposed a framework to find the most suitable knowledge graph for given settings.

Similarly, (Zaveri et al., 2013) propose a user-driven quality evaluation of DBpedia which assesses the quality of DBpedia by both manual and semi-automatic processes. The evaluation of DBpedia relies on the creation of a quality problem taxonomy which is used during crowdsourcing quality assessment. The quality problem taxonomy has fours dimensions: accuracy, relevancy, representational-consistency, and interlinking. Based on this evaluation, (Zaveri et al., 2013) identified around 222,298 incorrect triples and concluded that 11.93% of the tested DBpedia triples were having quality issues.

Another approach is the test-driven quality assessment by Kontokostas et al. (2014) that is inspired by test-driven software development to explore data quality problems. The authors propose a pattern-based approach to exploit the RDF data model for the data quality test of RDF knowledge bases. It contains two sources of inputs: the first input is from community feedback that can be used to generate data quality patterns, in the form of tuples (V, S), where V is a set of typed pattern variables

and S is a SPARQL query template. Based on the evaluation results, the study identified 32,293 test cases, and was able to solve data quality issues in an effective and efficient way.

Other approaches focused on particular aspects of the DBpedia knowledge base. For example, Font et al. (2015) performed an in-depth evaluation of domain knowledge representation in DBpedia. The paper confirmed the completeness problems of domain knowledge representation in DBpedia and the necessity of automatic methods for knowledge base completion.

Finally, some approaches relied on logical reasoning to detect inconsistencies in knowledge graphs. A rule-based approach to check and handle inconsistencies in DBpedia was proposed by Sheng et al. (2012), it uses rule-based reasoning with MapReduce. Five different types of inconsistencies can be detected by the approach: undefined class/properties, incompatible ranges of data type properties, inconsistencies in taxonomical links, and invalid entity definitions.

Similarly, Töpper et al. (2012) proposed a DBpedia ontology enrichment approach for inconsistency detection. The approach uses statistical methods to enrich the DBpedia ontology by identifying and resolving inconsistencies in DBpedia dataset based on the improved DBpedia ontology. The improved DBpedia ontology is free of syntactic, logical, and semantic errors. Based on the evaluation results, the system processed 3.11 million instances, and 50 thousand of them have been detected as inconsistent instances. Likewise, Lehmann and Buhmann (2010) proposed a tool called ORE (Ontology Repairing and Enrichment) for repairing and enriching knowledge bases. It uses machine learning algorithms to detect ontology modeling problems, and can guide users to solve the problems. When the tool is applied to DBpedia, it can detect and guide the user to solve the following problems in DBpedia: incorrect property ranges in DBpedia and DBpedia incompatibility problems with external ontologies.

2.5.2. Automatic Type Detection

There are several cutting-edge related works for automatic type detection (Töpper et al., 2012; Gangemi et al., 2012; Paulheim and Bizer, 2013; Paulheim and Bizer, 2014; Roder et al., 2015; Haidar-Ahmad et al., 2016; Van Erp and Vossen, 2017).

Some works rely on the detection of syntactic patterns from definitions to extract type information. For example the winners of the Open Knowledge Extraction competition (Haidar-Ahmad et al., 2016) extract entity types using SPARQL patterns on the dependency parses of natural language definitions and align the extracted textual types (e.g. “oke:Church”) to corresponding classes in

the DBpedia ontology (e.g. “oke:Church” rdfs:subClassOf “dbo:Place”) using disambiguation techniques.

Another approach for automatic typing is CETUS – a baseline approach for type extraction by Roder et al. (2015), which consists of pattern extraction from DBpedia abstracts and create local type hierarchies based on the extracted types. The final step maps the structured types to the DOLCE + DnS ontology[5] classes. Another automatic DBpedia entity typing approach is proposed by Gangemi et al. (2012), which presents an algorithm and a tool called Tipalo. It can interpret the DBpedia entity’s natural language definition from the Wikipedia page abstract to identify the most appropriate types for the DBpedia entity. The flowchart of Tipalo is shown in figure 2.7.

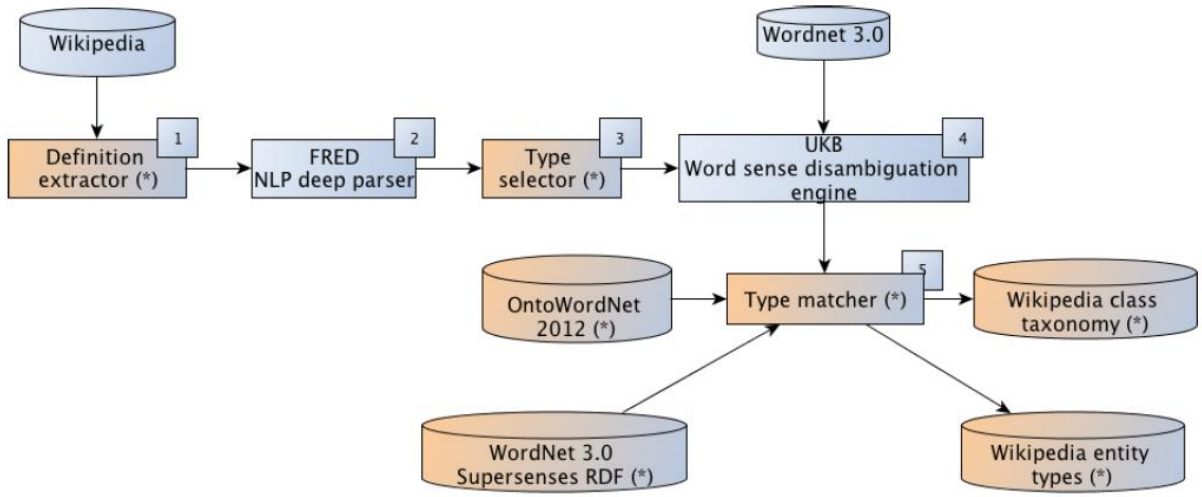

Figure 2. 7. Pipeline of Tipalo (Gangemi et al., 2012)

As shown in the pipeline, there are five major steps for Tipalo. The first step is to extract the contents from Wikipedia page abstracts, the second step is to parse the Wikipedia page abstracts using FRED[6], a machine reader for Semantic Web that parses natural langue text into Linked Data. The third step is to select the types and their relationships from the WordNet OWL graph, the fourth step is about disambiguation and the last step is to produce rdf:type statements based on the results from previous steps. The authors tested Tipalo on 800 randomly selected DBpedia entities,

[5] http://stlab.istc.cnr.it/stlab/WikipediaOntology/
[6] http://wit.istc.cnr.it/stlab-tools/fred

and obtained an F-score of 0.75 based on their manually annotated golden standard of Wikipedia entity types.

Other approaches rely on statistical methods on the DBpedia knowledge base rather than syntactic patterns for automatic type extraction. One approach to link types and instances in linked data is called SDType (Statistical Distribution Typing) by Paulheim and Bizer (2013) and Paulheim and Bizer (2014). It uses a weighted voting approach that exploits links between resources as indicators for types. More specifically, it uses statistical distributions for each ingoing and outgoing links to an instance as an indicator for predicting the instances types. SDType evaluated 3.6 million DBpedia instances and 359 DBpedia types, and led to the linking of an average of 5.6 types for each DBpedia instance. It also led to an average of 38 thousand instances per DBpedia type. Furthermore, the SDType approach successfully added a lot of type statements to DBpedia resources with 3.4 new million types (21% increase) to the DBpedia 3.9 release.

The most recent approach and the most similar to our work is (Van Erp and Vossen, 2017) which combines word embeddings with external information for entity typing. The vector models are used to assign vectors to words based on word embeddings using Word2vec. When testing with the YAGO knowledge base, the authors tested11.682 thousand entities and 70% of these entities were typed correctly.

For the automatic typing part of entities, most of the related works presented above rely on SPARQL patterns, statistical distributions, and knowledge extraction from Wikipedia abstracts for automatic typing. In contrast, our method uses entity embedding and n-gram models learned on Wikipedia coupled with clustering and classification algorithms. The most similar work appears to be (Van Erp and Vossen, 2017) but they combine word embeddings with external information instead of entity embeddings for entity typing. Furthermore, they tested with the YAGO knowledge base instead of DBpedia knowledge base.

2.5.3. Outlier Detection in Linked Data

Several approaches for detecting invalid or faulty knowledge in DBpedia were proposed in the state of the art (Fleischhacker et al., 2014; Paulheim and Bizer, 2014; Debattista et al., 2016), especially by detecting data points that differ significantly from their neighbors. For example, cross-checked outlier detection techniques were used to detect errors in numerical linked data in Fleischhacker et al. (2014). The authors proposed a two-step approach to detect errors in numerical

linked data, such as the population in a city, country, or continent. The first step is to apply outlier detection to the property values from the repository to split the data into relevant subsets. For example, subsets of "*population"* are generated as subpopulations, and outlier detection is applied to these subpopulations. The second step is to exploit the "*owl:sameAs"* links of the entities, for the purpose of collecting values from other repositories too. It performs a second outlier detection for these values. This is why the method is called a *Cross-Checked* Outlier Detection technique.

SDValidate, proposed by Paulheim and Bizer (2014), is another outlier detection algorithm, which aims at detecting faulty statements by using a technique based on statistical distributions in a manner similar to SDType. The approach consists of three steps: the first step is to compute the relative predicate frequency to describe the frequency of the combined predicate and object for each statement. The second step is to find confidence score for the selected statements, which uses the statistical distribution to assign a score to each of the selected statements based on their properties. More specifically, for each property, a vector is assigned to the predicate's subject and object. Then the cosine similarity of two vectors is computed, and stored as the confidence score for the statements. Finally, a threshold τ of 0.15 is used to test whether the statement can be categorized as a faulty statement or not. In Paulheim and Bizer (2014), SDValidate detected 13,000 erroneous statements in DBpedia.

Another outlier detection approach to enhance quality of linked data is by Debattista et al. (2016), which uses distance-based outlier detection technique to identify potentially incorrect RDF statements. Based on experiments with DBpedia, the approach reaches 0.76 for precision, 0.31 for recall, and 0.43 for F-score using data taken from DBpedia as a gold standard.

In contrast to most of the state of the art, in this thesis, we are interested in investigating how entity embeddings can contribute to detect invalid types and resources, and complete missing types for un-typed DBpedia resources. The main difference between our method and these related works is that our work uses clustering and classification algorithms instead of cross-checked outlier detection, distance-based outlier detection, or statistical distribution to detect invalid DBpedia resources and statements.

Chapter 3: Research Methodology

This chapter presents the process used for gathering our dataset from DBpedia, the methodology followed to build the entity embedding and n-gram models from Wikipedia pages' abstracts, and the generation of our clustering and classification models.

There were 760 DBpedia types available in the DBpedia ontology[7] at the date of our test in March 2017. In this version of the DBpedia ontology and knowledge base, only 461 classes have instances (through *rdf:type*). Among these 461 types, we ignore the classes with less than 20 instances (199), which leads to 358 classes for our experiments. Examples of some of the 199 classes include "ArtisticGenre," "TeamSport," and "SkiResort". The details of the DBpedia Ontology are shown in Appendix B.

3.1 General Architecture

This section describes the general architecture of the various modules involved in our experiments (See fig. 3.1).

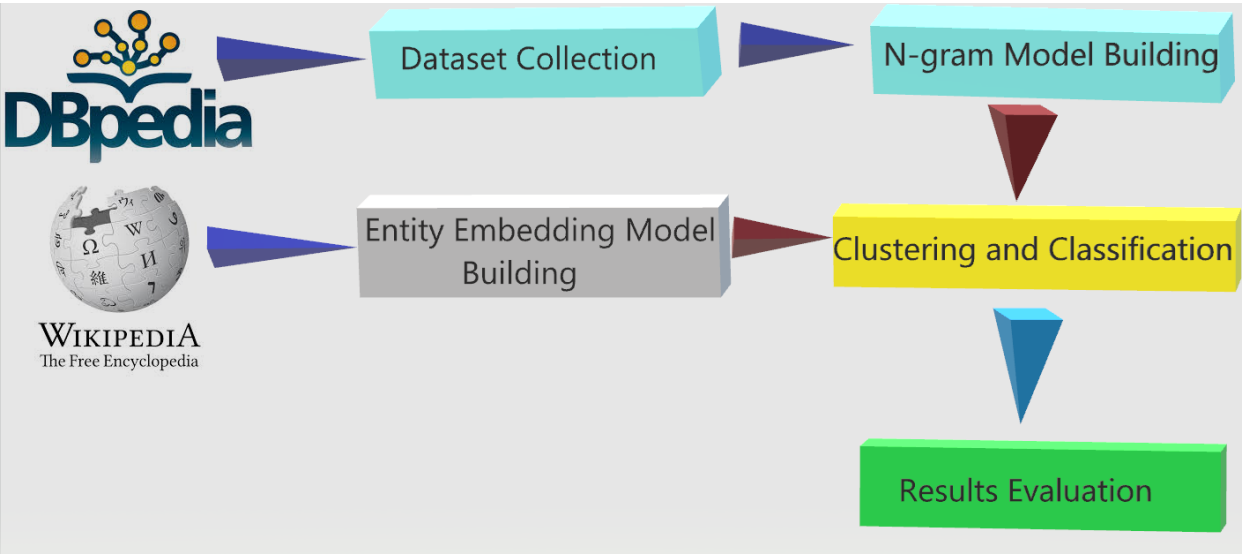

Figure 3. 1. Flowchart of the DBpedia invalid type and entity detection pipeline

[7] http://mappings.dbpedia.org/server/ontology/classes/

The first step is the development of the entity embedding models that are built with Wiki2ec with input from the Wikipedia dumps[8]. The second step is the development of n-gram models that are extracted with the Weka n-gram tokenizer[9] from DBpedia entities' abstracts. The third step is to prepare three datasets from DBpedia related to each of our tasks. The fourth step involves the clustering and classification experiments, which rely on the feature vectors produced in step 1 and 3. We compare three clustering and seven classification algorithms (including a baseline algorithm). The last step is to evaluate the generated results and discuss the performance of each clustering and classification algorithm.

3.2 Entity Embedding Model Building

The process of building our DBpedia entity embedding model is shown in Figure 3.2.

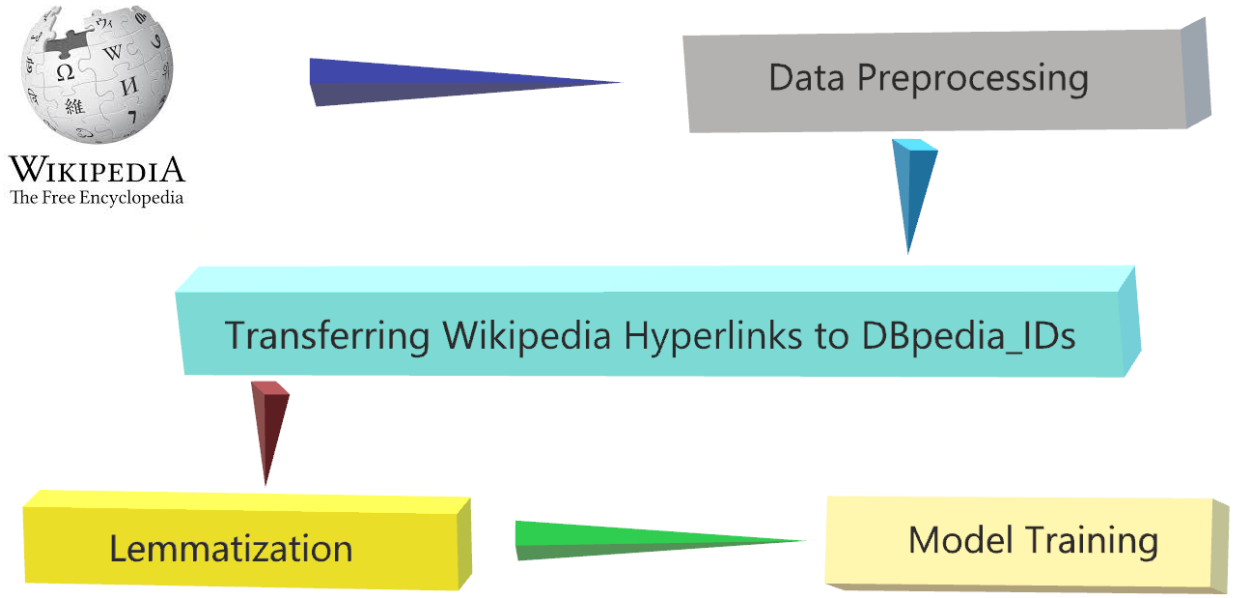

Figure 3. 2. The process of building the entity embedding model

Initially, we relied on one pre-built model, English Skip-gram model. We used pre-built models from wiki2vec for the "English Wikipedia (Feb 2015), without stemming, with a Skip-gram model. We noticed that many entities of the pre-built models did not have corresponding vectors. To solve

[8] https://dumps.wikimedia.org/enwiki/
[9] http://weka.sourceforge.net/doc.dev/weka/core/tokenizers/NGramTokenizer.html

this problem, we used both the Skip-gram (Mikolov et al., 2013b; Mikolov et al., 2013c) and CBOW (Mikolov et al., 2013b; Mikolov et al., 2013c) architectures for building our models.

In order to increase the coverage, we trained our model with the following parameters: a minimum number of occurrences of 5, a vector dimension of 100 to limit the size of the entity embedding model, and a window size set to 5, that describes the maximum distance between the current and predicted word within a sentence.

In this phase, the Word2vec tool (Mikolov et al, 2013a; Mikolov et al, 2013b; Mikolov et al, 2013c) was envisaged to compute vector representations of words. Given that we were interested in entities and not words, we needed a way to obtain vector representations of these entities (resources). For this, we used Wiki2vec, which is built on top of Word2vec, and allows to build a DBpedia model from Wikipedia. In fact, we can exploit the fact that each Wikipedia page is represented by a DBpedia resource. For example, the Wikipedia page "https://en.wikipedia.org/wiki/Airport" is directly mapped to the DBpedia resource "http://dbpedia.org/resource/Airport." Wiki2Vec replaces Wikipedia hyperlinks in Wikipedia pages by their corresponding DBpedia URIs. Next we run Word2vec on the modified corpus to train the DBpedia entity embedding models.

The detailed process is as follows. Our first step is to process Wikipedia Dumps (in English)[10] to extract the plain text, and to eliminate tags, figures, tables, etc. Hyperlinks that represent Wikipedia pages are identified by a specific "DBPEDIA_ID/" (e.g., "DBPEDIA_ID/Barack_Obama" replaces the hyperlink "https://en.wikipedia.org/wiki/Barack_Obama"). After that, lemmatization is performed using NLTK[11] and this allows us to build one vector representation for the words that share the same lemma, for example Child and Children have the same vector. The final step is to build the model with Wiki2vec.

Once the training process is finished, we obtain a continuous vector representation of single words and DBpedia entities. These vectors are used to compute similarities between entities and types. For example, in the Wiki2vec model, the similarity between "DBPEDIA_ID/Bill_Clinton" and "DBEPDIA_ID/President" can be computed. The obtained entity vectors also represent features for the classification and clustering tasks and are associated to the types that are already represented in DBpedia, when applicable. For example, the vector related to "dbr:Bill_Clinton" is

[10] https://dumps.wikimedia.org/enwiki/
[11] http://www.nltk.org/

associated to the type “dbo:President.” To give a better idea about this notion of distance, a visualization plot based on t-SNE[12] is provided in Figure 3.3.

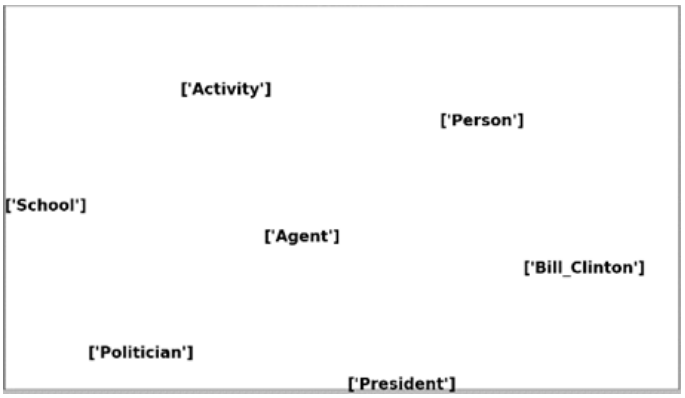

Figure 3. 3. Visualization of DBpedia types for the entity Bill_Clinton

The 2-D plot is based on a 100-dimension vector representation and uses PCA (Principle Component Analysis) (Abdi and Williams, 2010) for dimensionality reduction. For example, “dbr:Bill_Clinton” has four related DBpedia types: “dbo:Agent,” “dbo:President,” “dbo:Politician,” and “dbo:Person.” Two unrelated types, “dbo:School,” and “dbo:Activity” are added to the figure. Based on the plot shown above, we can observe that the types “dbo:Agent,” “dbo:President,” “dbo:Politician,” and “dbo:Person” are closer to the entity “dbr:Bill_Clinton” than the other two unrelated types.

Furthermore, our trained entity embedding models store both entities and words; entities (i.e., DBpedia resources) are distinguished from words using their “DBPEDIA_ID/”. DBpedia entities might be represented by compound words such as “Barack_Obama.[13]” For single words, the dataset usually contains both DBpedia entities and words. For example, both “DBPEDIA_ID/Poetry” and “Poetry” is represented in our model. We only use DBpedia entity “DBPEDIA_ID/Poetry” in our tasks, since we are only interested in DBpedia entities.

[12] https://lvdmaaten.github.io/tsne/
[13] http://dbpedia.org/page/Barack_Obama

3.3 Datasets Preparation

In this section, we present the process of datasets extraction, first for detecting invalid DBpedia types and complete missing types (training dataset and test dataset), then for detecting invalid DBpedia entities.

3.3.1. Dataset for Entity Type Detection

The first datasets consist of the training and test datasets for the automatic detection of DBpedia types. For each of the 358 DBpedia types, we retrieve entities with the specified type through DBpedia public Querying on the Semantic Web (SPARQL) endpoint[14].

For example, the following query was run to find instances of the class (type) *Airport*:

[14] http://dbpedia.org/sparql

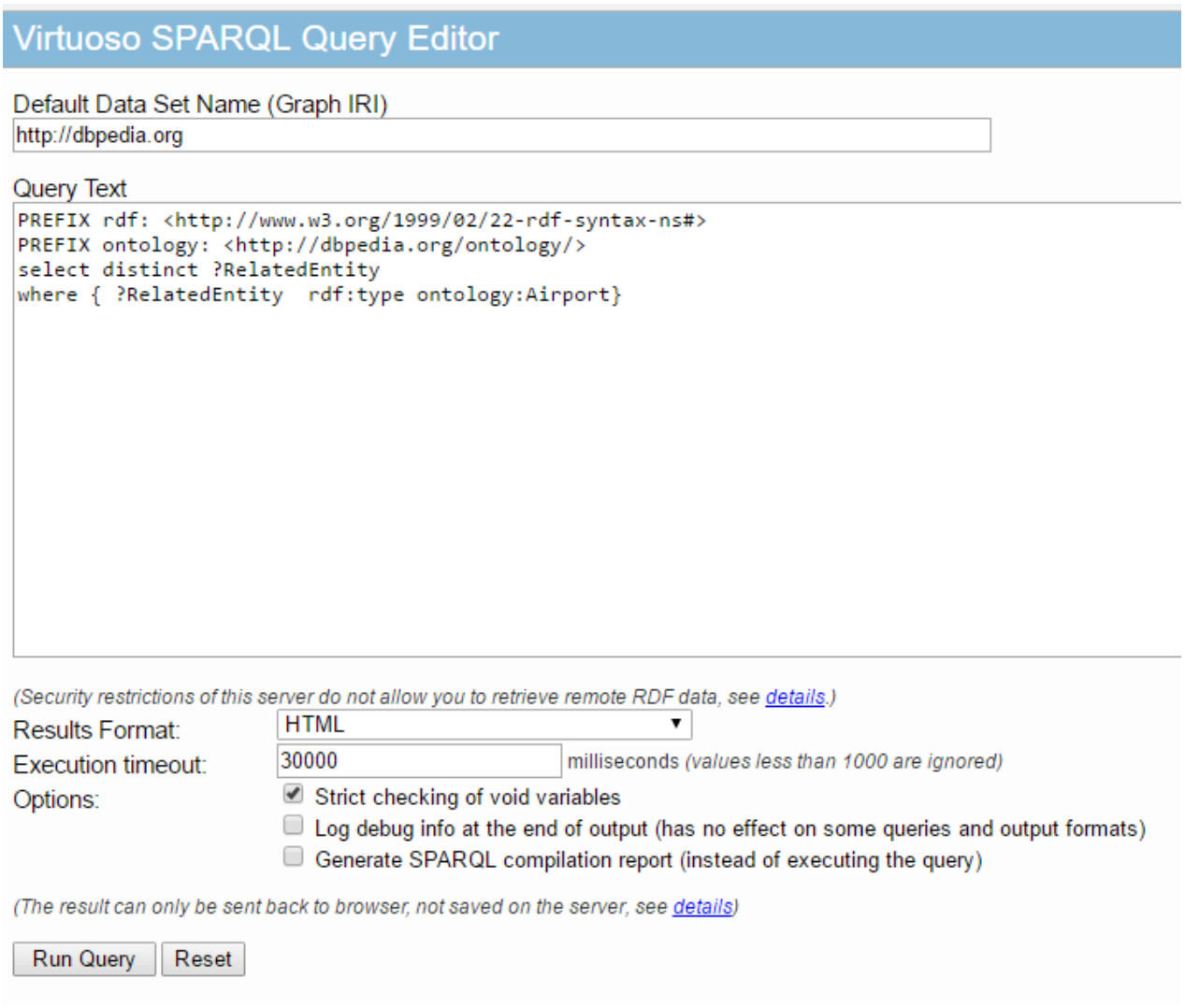

Figure 3. 4. Sample DBpedia SPARQL query-Airport

A sample of the retrieved entities is shown in Figure 3.5.

RelatedEntity
http://dbpedia.org/resource/Airport
http://dbpedia.org/resource/Xanthine
http://dbpedia.org/resource/Aksu_Airport
http://dbpedia.org/resource/Along_Airport
http://dbpedia.org/resource/Anadolu_Airport
http://dbpedia.org/resource/Brody_(air_base)
http://dbpedia.org/resource/Khatanga_Airport
http://dbpedia.org/resource/MacGregor_Airport
http://dbpedia.org/resource/North_Lake_(Red_River_County,_Texas)
http://dbpedia.org/resource/Oktyabrsky_Airport
http://dbpedia.org/resource/United_States_Navy_Facility,_Barbados
http://dbpedia.org/resource/Wakulla_County_Airport
http://dbpedia.org/resource/Wakulla_Springs
http://dbpedia.org/resource/Abendroth_Peak
http://dbpedia.org/resource/Aboisso_Airport
http://dbpedia.org/resource/Anduki_Airfield
http://dbpedia.org/resource/Anvik_Airport

Figure 3. 5. DBpedia SPARQL query result-Airport

To build the training dataset, we select at most 2000 entities for each DBpedia type using the process described above, then test the availability of each of these entities in our trained Word2vec entity embedding model. If the entity is available in our trained entity embedding model, we select the entity for the dataset, otherwise the entity is ignored. These entities are tagged as positive examples of that DBpedia type. Negative examples are chosen from a random selection of instances from all the remaining types, except the test DBpedia types. Given the variations in the number of instances for some DBpedia classes (some classes have few instances, while others may have more than a thousand entities), the number of negative entities depends on the corresponding number of positive entities in order to build a balanced dataset. Duplicates of positive examples are removed from the negative examples, in order to eliminate mis-clustering and mis-classification problems. For example, if “dbr:Bill_Clinton” is selected for the (positive) DBpedia type “dbo:President,” then the entity will not be selected as a negative example. This means that if an entity is selected as a positive one, we eliminate any mention of this entity in the negative examples. This can happen if the negative examples are taken from super-classes of the considered type, such as Person and President.

The process of building the test dataset is different from the training dataset, as we randomly select around 5 to 6 entities for each type from all of the 358 DBpedia types. In total, we obtained 2,111 entities in our test dataset, distributed among these various types. Each entity has an average of 4 to 5 types. We stored entities and all their related types. For example, we might have "dbr:Bill_Clinton" with types "dbo:President," "dbo:Agent," "dbo:Politician," and "dbo:Person;". We make sure there are no common entities in the training and test dataset. These train and test datasets are available to download for the research community. [15].

3.3.2. Dataset for Invalid Entity Detection

The second dataset for the task of detecting invalid DBpedia resources is collected separately from the datasets of detecting invalid DBpedia types. In preparation for the dataset, we randomly select 2573 entities through DBpedia Querying on the Semantic Web (SPARQL) endpoint that are available in our entity embedding models. For each of the selected DBpedia entities, we need all of their corresponding DBpedia resources.

For example, the DBpedia resource "dbr:Barack_Obama" is described in the RDF format by a set of <Subject Predicate Object> triples where the Subject is "dbr:Barack_Obama," and the Predicate are the predicates that related to the current Subject, such as "dbo:birthPlace." The Object contain the related resources that may or may not be invalid As show in figure 3.6[16], the related resource (Object) through the Predicate "dbo:almaMater" is "dbr:Occidental_College." The related resources through the Predicate "dbo:birthPlace" are "dbr:Hawaii," "dbr:Illinois" and "dbr:Kapiolani_Medical_Center_for_Women_and_Children."

These related resources are tagged as positive examples of that DBpedia entity "dbr:Barack_Obama." Negative examples are chosen from a random selection of entities from all of the remaining entities, except the entities that share the same class. For example, the "dbr:Hawaii" has five classes "dbo: Place," "dbo: Location," "dbo: City," "dbo: PopulatedPlace," and "dbo: Settlement." We do not select any entities (resources) that are associated with these classes, such as "dbr:New_York," "dbr:California," and "dbr:Chicago" etc. The number of negative entities depends on the corresponding number of positive entities in order to build a balanced dataset.

[15] http://www.site.uottawa.ca/~diana/resources/kesw17/
[16] Results retrieved from http://dbpedia.org/page/Barack_Obama

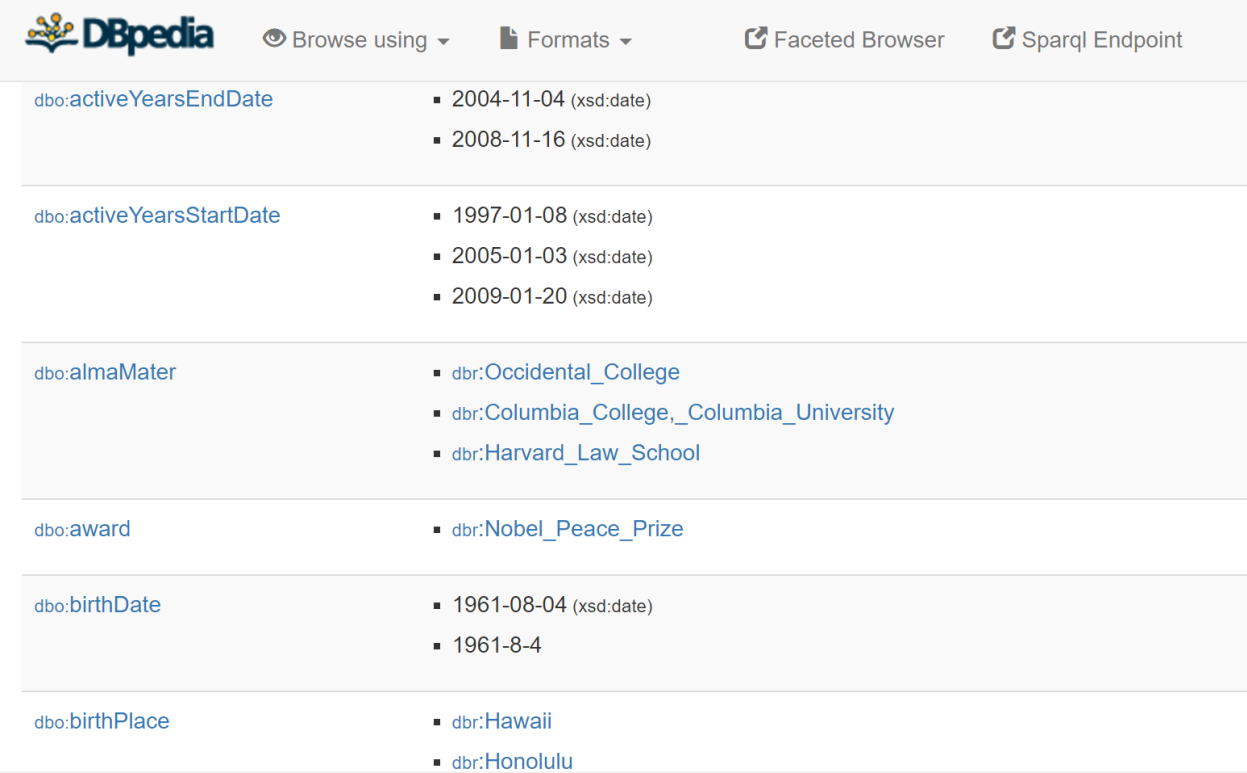

Figure 3. 6. DBpedia page-Barack_Obama

The following query was run to find related DBpedia resources of the resource Barack_Obama (figure 3.7)

Figure 3. 7. Sample DBpedia SPARQL query-Barack_Obama

The results for the related DBpedia resources for “dbr:Barack_Obama” are shown in figure 3.8.

dbr
http://dbpedia.org/resource/2008
http://dbpedia.org/resource/Confirmations_of_Barack_Obama's_Cabinet
http://dbpedia.org/resource/List_of_bills_sponsored_by_Barack_Obama
http://dbpedia.org/resource/Climate_change_policy_of_the_United_States
http://dbpedia.org/resource/List_of_things_named_after_Barack_Obama
http://dbpedia.org/resource/The_United_States_Senate
http://dbpedia.org/resource/International_media_reaction_to_the_United_States_presidential_election
http://dbpedia.org/resource/Hawaii
http://dbpedia.org/resource/Honolulu
http://dbpedia.org/resource/Kapiolani_Medical_Center_for_Women_and_Children
http://dbpedia.org/resource/Occidental_College
http://dbpedia.org/resource/Columbia_College,_Columbia_University
http://dbpedia.org/resource/Harvard_Law_School
http://dbpedia.org/resource/Nobel_Peace_Prize
http://dbpedia.org/resource/Democratic_Party_(United_States)

Figure 3. 8. DBpedia SPARQL query result-Barack_Obama

The experiments on the training datasets helps us find the most suitable machine learning algorithms for our task, and the experiments on the test dataset validates the performance of different machine learning algorithms. Here, we focus on the 358 types that contain at least 20 entities. The rest of DBpedia types are ignored.

Table 3.1 summarizes the size of the three datasets.

Dataset	Training data set for DBpedia invalid type detection	Test dataset for DBpedia invalid type detection	Dataset for DBpedia invalid entity detection
Number of entities	360,843	2,111	108,936[17]

Table 3. 1. Summary of the number of instances in the datasets

[17] This number is the total number of entities included in both subjects and objects. There are 2,573 subject entities.

3.4 N-gram Model Building

The n-gram models are extracted using the Weka n-gram tokenizer. The first step is to collect the DBpedia entities' abstracts through DBpedia public Querying on the Semantic Web (SPARQL) endpoint, then save these entities' abstracts into output files. The second step is to learn the n-gram models using Weka on the output files by varying the length of the word sequences (*n*) and using TF-IDF to weight the n-grams. We perform Random Forest feature selection[18] (Rogers and Gunn, 2006; Pan and Shen, 2009; Genuer et al., 2010; Genuer et al., 2015) in order to reduce the number of features to 1000. This number was determined after a set of empirical experiments.

Other traditional feature selection techniques have been performed and the resulting models have applied to the task of DBepdia Entity Detection. including Principal Component Analysis (PCA) (Abdi and Williams, 2010), Linear Discriminant Analysis (LDA) (Webb, 2002) and Pearson Coefficient (Kornbrot, 2005). Based on the results of these feature selection techniques, Random Forest feature selection gets the best performance among all of the tested feature selection techniques. The original dimensions are 36,842 for the uni-gram model, 62,637 for the bi-gram model, 81,830 for the tri-gram model, and 181,309 for the n-gram model; with the Random Forest feature selection technique, the dimension of n-gram models decreased to 1,000.

3.5 Evaluation Metrics

Here we describe the metrics used to evaluate the results. The evaluation measures include Precision, Recall, F-Score, Accuracy, and Area Under the Curve (AUC) (for the classification experiments). Finally, the student-t test measure is described.

The evaluation metrics are calculated based on a confusion matrix (Table 3.2), which is a table that shows the classification or clustering results. The following measures are defined as follows (Han et al, 2012):

- True Positives (TP): These refers to the positive examples that were correctly clustered/labeled by the clustering tool or classifier.
- True Negatives (TN): These refers to the negative examples that were correctly clustered/labeled by the clustering tool or classifier.

[18] A random forest consists of a number of decision trees. Every node in the decision trees is a condition on a single feature, designed to split the dataset into two sets, so that similar response values end up in the same set. The feature selection measure is based on which locally optimal condition is chosen and it is called impurity.

- False Positives (FP): These refers to the negative examples that were incorrectly clustered/labeled as positive.
- False Negatives (FN): These refers to the positive examples that were incorrectly clustered/labeled as negative.
- Positive (P): These refers to total number of positive examples.
- Negative (N): these refer to total number of negative examples.

	Predicted Class/Cluster			
Actual Class/Cluster		Positive	Negative	Total
	Positive	TP	FN	P
	Negative	FP	TN	N
	Total	P	N	P+N

Table 3. 2. Confusion Matrix

Precision is a measure of exactness, and calculates the percentage of the examples labeled as positive that are actual positive examples.

$$precision = \frac{TP}{TP + FP} \qquad (3.1)$$

Recall is t a measure of completeness, and calculates the percentage of the examples labeled as positive among all examples of the class.

$$recall = \frac{TP}{TP + FN} \qquad (3.2)$$

F-Score is a combination of Precision and Recall, F-Score can be seen as a harmonic mean of Precision and Recall.

$$F\text{-}score = \frac{2 \times precision \times recall}{precision + recall} \qquad (3.3)$$

Accuracy is the percentage rate of correctly classified examples.

$$accuracy = \frac{TP + TN}{P + N} \qquad (3.4)$$

AUC compares the performance of the classifiers not only over the entire range of cross distribution, but also the error costs, it represents the expected performance of the classifier, which is the average of the pessimistic and optimistic results (Ling et al., 2003; Fawcett, 2006).

Finally, a **Student-t test** is used to test the significance level of the difference between two models.

Chapter 4: DBpedia Entity Type Detection

4.1 Experiment Introduction and Setup

This part of the thesis is based on our paper (Zhou et al., 2017) published in the Knowledge Engineering and Semantic Web conference. This phase consists of two parts: clustering and classification. Both parts use Scikit-Learn[19] (Pedregosa et al., 2011) on the prepared dataset for each DBpedia type. We perform a binary clustering and a binary classification that represent two main categories for each type of interest: The Type category and the NOT Type category (The positive class/cluster and the negative class/cluster). For example, we want to classify examples as instances of *Airport* and instances of NOT *Airport* using the vectors as features.

In terms of clustering, all of the entities are clustered with the following standard algorithms: K-means, Mean Shift, and Birch. Clustering is performed by computing the Euclidean distance between vectors. The number of clusters is set to two, one represents the positive cluster for the type of interest, and the other one is the negative cluster.

As a reminder, for each DBpedia type, the related DBpedia entities are selected using the predicate *rdf:type* through DBpedia Querying on the Semantic Web (SPARQL) endpoint for the training and test datasets. These entities are considered as examples of the positive class, the same number of negative entities is then selected from DBpedia except the entities from the positive class. Because the produced clusters are not labelled, the cluster with a greater number of positive entities is considered the positive cluster and vice versa.

In the classification part, several classification algorithms are tested. We experimented with Decision Tree, Extra Tree, Random Forest, K Nearest Neighbor (KNN), Naive Bayes and Support Vector Machine (SVM). We wanted to test tree-based algorithms, such as Decision Tree, since the trees could be helpful to track the decision process. For ensemble methods, the Random Forest was used. Naive Bayes classifiers are based on Bayes theorem and they tend to work well on text data and more generally on homogeneous features. Support Vector Machine classifier has often robust performance.

[19] http://scikit-learn.org/stable/

In the evaluation section, we present clustering and classification results with different entity embedding and n-gram models, and we compare results with different clustering and classification algorithms. In addition, results on both training and test dataset are shown. The experiments on the training dataset helps us find the most appropriate clustering and classification algorithms for our task. Then we build models on the whole training dataset and we apply them on the held-out test set, on which we report our final results.

4.2 Clustering Evaluation

The clustering evaluation consists of two subsections, the first subsection contains evaluation results with n-gram models and the second subsection contains evaluation results with entity embedding models.

4.2.1 Clustering with N-gram Models

4.2.1.1 Results on the training dataset

Based the result on all of the 358 DBpedia types, Random Forest gets optimal results among all of the feature selection techniques we used. The original uni-gram, bi-gram, tri-gram and n-gram models have dimension around 30,000. After performing feature selection with Random Forest, the dimension becomes 1,000.

Table 4.1 to 4.4 show the results of the 358 DBpedia types on the training dataset with four different n-gram models: uni-grams, bi-grams, tri-grams, and n-grams with n=1~3.

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.819	0.657	0.669	66.3%
Mean Shift	0.357	0.431	0.367	27.4%
Birch	0.881	0.745	0.726	70.2%

Table 4. 1. Summary of the average clustering results with the uni-gram model on the training dataset

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.816	0.655	0.664	65.9%
Mean Shift	0.361	0.429	0.365	27.5%
Birch	0.800	0.759	0.730	69.8%

Table 4. 2. Summary of the average clustering results with bi-gram model on the training dataset

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.818	0.661	0.670	66.4%
Mean Shift	0.346	0.433	0.367	26.7%
Birch	0.810	0.743	0.721	69.8%

Table 4. 3. Summary of the average clustering results with the tri-gram model on the training dataset

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.897	0.671	0.731	74.4%
Mean Shift	0.407	0.391	0.345	28.8%
Birch	0.858	0.773	0.774	76.2%

Table 4. 4. Summary of the average clustering results with the n-gram model on the training dataset

Based on the results of the uni-gram, bi-gram, tri-gram and n-gram models, the uni-gram, bi-gram and tri-gram models have similar performance. When we look at the detailed results of these three models, the uni-gram model performs slightly better than the bi-gram model, and the bi-gram model also performs slightly better than the tri-gram model. According to the results of table 4.1 to 4.4, the n-gram model is significantly different than the other three models; it performs with around 10% better than the other three models in terms of F-score. We report the improvements as differences (percentage points).

K-means and Birch have similar performance, and they usually perform better than Mean Shift. According to the clustering results with the n-gram model (table 4.4), K-means and Birch perform with around 40% better than Mean Shift in terms of F-score.

Based on the above results for the entity embedding and n-gram models, Skip-gram model has the best performance among all of the models followed by the N-gram model.

4.2.1.2 Results on the test dataset

Tables 4.5 to 4.8 show the average clustering results with different n-gram models on the test dataset.

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.505	0.940	0.657	50.9%
Mean Shift	0.528	0.737	0.607	52.9%
Birch	0.507	0.933	0.654	50.8%

Table 4. 5. Summary of the average clustering results with the uni-gram model on the test dataset

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.503	0.937	0.654	50.5%
Mean Shift	0.516	0.718	0.589	51.0%
Birch	0.505	0.933	0.653	50.5%

Table 4. 6. Summary of the average clustering results with the bi-gram model on the test dataset

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.513	0.930	0.655	51.1%
Mean Shift	0.530	0.734	0.602	52.5%
Birch	0.514	0.929	0.655	51.2%

Table 4. 7. Summary of the average clustering results with the tri-gram model on the test dataset

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.507	0.935	0.656	50.9%
Mean Shift	0.525	0.732	0.597	51.6%
Birch	0.510	0.945	0.662	51.6%

Table 4. 8. Summary of the average clustering results with the n-gram model on the test dataset

Based on the averaged clustering results for the four n-gram models on the test dataset, different from the results on the training dataset, uni-gram, bi-gram, tri-gram and n-gram models have very close performance, there is no significant difference between them.

4.2.2 Clustering with Entity Embedding Models

In the evaluation section, we present the clustering results both on the training dataset via cross-validation (section 4.2.2.1) and the held-out test dataset (section 4.2.2.2).

4.2.2.1 Results on the training dataset

This subsection shows the average results by using training dataset with different entity embedding models.

There are two word embedding models we build for the experiments. Table 4.9 and table 4.10 show the average results for clustering on the 358 DBpedia types with Skip-gram and CBOW entity embedding models with the DBpedia knowledge base gold standard. We consider the cluster with more positive instance as the positive cluster and vice versa. We report the metrics for the positive class only.

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.937	0.950	0.941	94.2%
Mean Shift	0.560	0.991	0.713	71.3%
Birch	0.941	0.945	0.937	94.0%

Table 4. 9. Summary of the average clustering results with Skip-gram entity embedding model on the training dataset

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.826	0.653	0.669	66.7%
Mean Shift	0.360	0.431	0.368	27.6%
Birch	0.815	0.753	0.733	70.8%

Table 4. 10. Summary of the average clustering results with CBOW entity embedding model on the training dataset

Based on the results of the two entity embedding models, the performance of Skip-gram is usually better than CBOW. Skip-gram performs with around 26% better than CBOW in terms of F-Score. For Accuracy, the difference is even bigger; Skip-gram performs with around 32% better than CBOW.

Furthermore, K-means has similar performance as Birch, and their performance is much better than that of Mean Shift. With the results of the Skip-gram model, K-means and Birch perform with around 23% better than Mean Shift in terms of F-score. When switching to the CBOW model, the different becomes even bigger (0.7 compared to 0.3).

The Skip-gram entity embedding model has the best performance among all of the entity embedding and n-gram models based on the test dataset, followed by the CBOW entity embedding model.

4.2.2.2 Results on the test dataset

We perform clustering on the test as an independent step from the training set. In fact there was no need to separate clustering in both test and train and that we did only to follow the same structure that is required for classification.

Similar to the experiments on the training dataset, the experiments on the test dataset were also performed with K-means, Mean Shift, and Birch clustering algorithms with different entity embedding and n-gram models.

The table 4.11 shows the average clustering results on the 358 DBpedia types with the test dataset using the Skip-grams entity embedding model.

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.820	0.935	0.856	83.0%
Mean Shift	0.657	0.951	0.756	69.1%
Birch	0.837	0.932	0.861	83.7%

Table 4. 11. Summary of the average clustering results with Skip-gram entity embedding model on the test dataset

Table 4.12 shows the average clustering results on the 358 DBpedia types with the test dataset using the Continuous Bag-Of-Words (CBOW) entity embedding model.

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.543	0.924	0.667	53.8%
Mean Shift	0.533	0.785	0.627	53.9%
Birch	0.535	0.916	0.660	52.8%

Table 4. 12. Summary of the average clustering results with CBOW entity embedding model on the test dataset

Based on the clustering results on entity embedding models (table 4.11 and 4.12), the results of Skip-gram model are with around 17% better than that of CBOW model in terms of F-Score, when comparing the results in terms of Accuracy, the difference is even bigger. The Skip-gram model is with around 55% better than CBOW model.

In terms of clustering results of the skip-gram entity embedding model, K-means and Birch are with around 11% better than Mean Shift in terms of F-Score, and they are with 14% better than Mean Shift in terms of Accuracy. For the clustering results of the CBOW entity embedding model and the uni-gram, bi-gram, tri-gram and n-gram n-gram models, the three clustering algorithms have similar performance, and there is no significant difference between the three clustering algorithms using these models.

The tables in appendix A show the detailed results of 10 typical DBpedia types with K-means clustering algorithms on the Skip-gram and n-gram models.

4.3 Classification Evaluation

This section of the thesis contains the evaluation results using classification algorithms. In this work, we used six classification algorithms: Decision Tree, Extra Tree, K Nearest Neighbor (KNN), Random Forest, Naïve Bayes and Support Vector Machine (SVM). Also, a baseline algorithm was used for a comparison / lower bound (named DummyClassifier in Scikit-learn).

4.3.1 Classification with N-gram Models

This section shows the classification results with various traditional N-gram models. It contains two subsections, the first subsection shows classification results on the training dataset, and the second subsection shows the classification results on the test dataset.

4.3.1.1 Results on the Training dataset

Tables 4.13 to 4.16 show the average classification results with the uni-gram, bi-gram, tri-gram and n-gram models on the training dataset.

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.498	0.496	0.485	49.7%	0.500
Decision Tree	0.907	0.874	0.884	89.4%	0.898
Extra Tree	0.933	0.879	0.900	91.0%	0.955
KNN	0.933	0.640	0.737	80.1%	0.896
Random Forest	0.932	0.858	0.886	90.1%	0.951
Naïve Bayes	0.835	0.927	0.871	86.9%	0.876
SVM	0.961	0.837	0.884	90.6%	0.960

Table 4. 13. Summary of the average classification results with uni-gram model on the training dataset

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.497	0.497	0.485	49.5%	0.801
Decision Tree	0.904	0.871	0.881	89.0%	0.895
Extra Tree	0.935	0.876	0.898	91.0%	0.957
KNN	0.932	0.639	0.737	80.2%	0.900
Random Forest	0.934	0.859	0.888	90.2%	0.956
Naïve Bayes	0.836	0.928	0.872	86.9%	0.874
SVM	0.956	0.833	0.879	90.2%	0.955

Table 4. 14. Summary of the average classification results with bi-gram model on the training dataset

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.501	0.501	0.489	49.9%	0.501
Decision Tree	0.906	0.872	0.883	89.3%	0.897
Extra Tree	0.939	0.881	0.903	91.4%	0.959
KNN	0.934	0.640	0.737	80.2%	0.897
Random Forest	0.938	0.860	0.891	90.5%	0.957
Naïve Bayes	0.840	0.930	0.875	87.2%	0.880
SVM	0.953	0.832	0.878	90.1%	0.950

Table 4. 15. Summary of the average classification results with tri-gram model on the training dataset

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.504	0.501	0.490	50.2%	0.499
Decision Tree	0.933	0.921	0.923	93.0%	0.933
Extra Tree	0.955	0.926	0.936	94.1%	0.976
KNN	0.954	0.738	0.815	85.2%	0.931
Random Forest	0.955	0.911	0.928	93.7%	0.977
Naïve Bayes	0.855	0.939	0.887	88.4%	0.894
SVM	0.964	0.895	0.921	93.6%	0.976

Table 4. 16. Summary of the average classification results with n-gram model on the training dataset

Based on these classification results, the n-gram model has the best performance in terms of both F-score and Accuracy among all of the n-gram models, followed by uni-gram, bi-gram and tri-gram models. Extra Tree and SVM have the best results among all the classification algorithms.

4.3.1.2 Results on the test dataset

Table 4.17 to table 4.20 show the average classification results with the uni-gram, bi-gram, tri-gram and n-gram models on the test dataset.

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.499	0.493	0.487	50.0%	0.492
Decision Tree	0.940	0.923	0.926	92.9%	0.936
Extra Tree	0.962	0.916	0.932	93.8%	0.974
KNN	0.838	0.601	0.635	72.0%	0.845
Random Forest	0.959	0.896	0.918	92.8%	0.967
Naïve Bayes	0.835	0.921	0.866	85.7%	0.872
SVM	0.982	0.874	0.912	92.7%	0.970

Table 4. 17. Summary of the average classification results with uni-gram model on the test dataset

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.487	0.499	0.485	49.3%	0.496
Decision Tree	0.921	0.853	0.876	88.8%	0.896
Extra Tree	0.948	0.851	0.886	90.1%	0.952
KNN	0.774	0.344	0.430	64.2%	0.801
Random Forest	0.949	0.833	0.874	89.3%	0.948
Naïve Bayes	0.783	0.933	0.842	82.2%	0.823
SVM	0.968	0.778	0.842	87.9%	0.953

Table 4. 18. Summary of the average classification results with bi-gram model on the test dataset

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.500	0.507	0.495	50.1%	0.494
Decision Tree	0.912	0.789	0.831	84.9%	0.860
Extra Tree	0.920	0.815	0.847	86.3%	0.915
KNN	0.824	0.420	0.501	66.3%	0.791
Random Forest	0.919	0.800	0.836	85.0%	0.916
Naïve Bayes	0.759	0.919	0.819	79.1%	0.794
SVM	0.941	0.697	0.766	81.7%	0.913

Table 4. 19. *Summary of the average classification results with tri-gram model on the test dataset*

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.501	0.511	0.498	50.2%	0.502
Decision Tree	0.957	0.922	0.934	93.8%	0.943
Extra Tree	0.970	0.906	0.930	93.7%	0.974
KNN	0.864	0.497	0.852	70.4%	0.827
Random Forest	0.964	0.898	0.921	93.0%	0.968
Naïve Bayes	0.828	0.937	0.871	85.7%	0.859
SVM	0.987	0.962	0.907	92.4%	0.965

Table 4. 20. Summary of the average classification results with n-gram model on the test dataset

Similar to the results on training dataset, according to the classification based on the uni-gram, bi-gram, and tri-gram and n-gram model, the n-gram model gets the best results among all of the n-gram models, followed by the uni-gram model, then by bi-gram and tri-gram models in terms of F-score and Accuracy. However, according to the classification based on the uni-gram, bi-gram, and tri-gram and n-gram model, the difference between the n-gram model and the uni-gram model is not significant based on the results of all classification algorithms in terms of F-score and Accuracy.

4.3.2 Classification with Entity Embedding Models

This section consists of two subsections, the first subsection contains the evaluation results on the training dataset, and the second subsection contains the evaluation results on the test dataset. The training dataset can help us to find the best models approach, and the test dataset is a good use case for the identification of the type of new DBpedia resources.

4.3.2.1 Results on the training dataset

This subsection contains the results by using training dataset with different entity embedding and n-gram models.

Table 4.21 and table 4.22 show the average classification results on the 358 DBpedia types using Skip-gram and CBOW the entity embedding models on the training dataset.

We report results by 10-fold cross-validation on the training data.

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.499	0.497	0.487	49.8%	0.502
Decision Tree	0.881	0.897	0.883	88.8%	0.888
Extra Tree	0.962	0.950	0.954	95.8%	0.988
KNN	0.869	0.995	0.922	91.8%	0.976
Random Forest	0.958	0.935	0.944	94.9%	0.986
Naïve Bayes	0.967	0.955	0.959	96.4%	0.991
SVM	0.958	0.986	0.970	97.3%	0.995

Table 4. 21. Summary of the average classification results with Skip-gram entity embedding model on the training dataset

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.496	0.500	0.486	49.7%	0.503
Decision Tree	0.908	0.896	0.881	89.0%	0.896
Extra Tree	0.937	0.897	0.899	91.1%	0.955
KNN	0.941	0.641	0.739	80.2%	0.898
Random Forest	0.936	0.857	0.887	90.1%	0.955
Naïve Bayes	0.837	0.925	0.871	87.1%	0.877
SVM	0.959	0.836	0.883	90.3%	0.955

Table 4. 22. Summary of the average classification results with CBOW entity embedding model on the training dataset

Based on the classification results on entity embedding models on training dataset (table 4.21 and table 22), the Skip-gram model still performs better than the CBOW model, but the difference between them is quite small; the performance of Skip-gram model is only around 7% better than CBOW model in terms of F-score and Accuracy. The baseline algorithm gets the F-score and Accuracy around 0.5 for both models, which is the same as a random choice. For the results with Skip-gram entity embeddings, Extra Tree, Random Forest, Naïve Bayes, and SVM get close performance, and they usually perform better than other classifiers. Their F-Score and Accuracy are more than or close to 0.95. When switching to the CBOW model, the performance of these classification algorithms drops with around 10% or more, except for the baseline algorithm.

Based on the overall results of entity embedding and n-gram models, tree-based algorithms and SVM over-perform other ones. Compared to the results of the Skip-gram model, Naïve Bayes does not perform as well on the n-gram models: the F-Score, Accuracy and AUC are around 0.85 for the n-gram models compared to 0.95 or higher on the Skip-gram models. Extra Tree and SVM have the best results among all the classification algorithms on both training and test datasets. In terms of the Skip-gram entity model, Support Vector Machine performs slightly better than Extra Tree in terms of F-score, while the difference is within 2%. When switching to the n-gram model, Extra Tree performs slightly better than Support Vector Machine in terms of F-score, the difference is within 1%. However, the two algorithms have almost the same Accuracy and AUC.

Furthermore, like the results in the clustering part, the Skip-gram entity embedding model still has the best performance among all of the entity embedding and uni-gram, bi-gram, tri-gram and n-gram models. Skip-gram model has performance slightly better than the n-gram model in our classification experiments and the difference is much smaller than in the clustering ones. For example, with the Extra Tree classification algorithm, the difference between the two models is only around 3% in terms of F-Score and Accuracy. Using the Support Vector Machine classification algorithm on the test dataset, the difference between the two models is around 8% and 6% in terms of F-Score and Accuracy

4.3.2.2 Results on the test dataset

This subsection shows classification results on the test dataset. Tables 4.15 and 4.16 show the average classification results on the 358 DBpedia types by using the Skip-gram and CBOW entity embedding models respectively.

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.506	0.507	0.495	50.5%	0.503
Decision Tree	0.900	0.892	0.889	89.1%	0.891
Extra Tree	0.972	0.931	0.946	95.2%	0.986
KNN	0.886	0.984	0.927	91.9%	0.975
Random Forest	0.961	0.922	0.938	94.5%	0.983
Naïve Bayes	0.974	0.925	0.943	95.1%	0.986
SVM	0.966	0.967	0.962	96.5%	0.991

Table 4. 23. *Summary of the average classification results with Skip-gram entity embedding model on the test dataset*

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.507	0.505	0.495	50.3%	0.506
Decision Tree	0.812	0.836	0.814	81.4%	0.814
Extra Tree	0.920	0.850	0.875	88.6%	0.947
KNN	0.818	0.948	0.868	85.0%	0.950
Random Forest	0.917	0.948	0.872	88.3%	0.941
Naïve Bayes	0.671	0.778	0.678	66.5%	0.798
SVM	0.926	0.890	0.897	90.4%	0.962

Table 4. 24. Summary of the average classification results with CBOW entity embedding model on the test dataset

Based on the classification results on the 358 DBpedia types with the Skip-gram and the CBOW entity embedding models on the test dataset (table 4.23 and 4.24), the Skip-gram model still performs better than the CBOW model; the Skip-gram model is with around 8% better than CBOW model in terms of F-Score and Accuracy. While for the results in terms of Area Under the Curve (AUC), Skip-gram model performs with 5% better than CBOW model. Similar to the classification results on the training dataset, Extra Tree, Random Forest, Naïve Bayes, and SVM have high quality results, and can be used to detect most DBpedia types correctly. When switching from the Skip-gram model to the CBOW model, the overall results drop around 10%. One interesting point is that when switching from the Skip-gram model to the CBOW model, the results of Naïve Bayes drop significantly, from 0.95 to 0.68 in terms of F-Score, and from 0.96 to 0.66 in terms of Accuracy. The difference between them is around 27% for F-Score, and 30% for Accuracy.

4.5 Student-t Test

Several student-t tests were applied on the precision, recall, f-score, and accuracy results of the Skip-gram entity embedding and the n-gram model, using the best classification algorithm Support Vector machine based on both the training and test dataset with one-tailed hypothesis. The details of Student-t test results based on precision, recall, f-score, and accuracy results of the Skip-gram entity embedding, and n-gram traditional n-gram models, are shown in table 4.23.

For student-t test on the training dataset, the results are *not* statistically significant for precision results. However, recall, f-score and accuracy results are statistically significant at $p < 0.01$.

For student-t test on the test dataset, precision results are statistically significant at $p < 0.01$ as well as recall and F-score. However, accuracy results are *not* statistically significant at $p < 0.01$, $p<0.05$ or $p<0.1$.

Dataset	Results	Significance level	t value	p value	Student-t Test Results
Training dataset	Precision	0.01	-1.022	0.153	*not* significant
	Recall	0.01	13.436	< 0.00001	significant at p < 0.01
	F-score	0.01	7.793	< 0.00001	significant at p < 0.01
	Accuracy	0.01	7.722	< 0.00001	significant at p < 0.01
Test dataset	Precision	0.01	-6.006	<0.00001	significant at p < 0.10
	Recall	0.01	9.059	< 0.00001	significant at p < 0.01
	F-score	0.01	5.970	< 0.00001	significant at p < 0.01
	Accuracy	0.01	0.109	0.457	*not* significant

Table 4. 25. Student-t tests on Precision, Recall, F-score, and Accuracy results based on the training and test datasets

4.6 Synthesis and Discussion

Overall, the Skip-gram model obtains the best results among all of the entity embedding and the traditional models followed by n-gram and uni-gram models. Continuous Bag-Of-Words has a similar performance with bi-gram and tri-gram models. The difference between results obtained using the Skip-grams model and the n-gram models is statistically significant in terms of recall, f-score and accuracy results on the training dataset. Furthermore, based on the precision, recall and F-score results on the test dataset, the difference between results obtained using the Skip-gram model and the n-gram models is also statistically significant.

Random Forest, Extra Tree and Support Vector Machine were among the top classifiers with good performance on all of the entity embedding and n-gram models. The Random Forest feature selection algorithm helped to decrease the dimension of vector for n-gram models.

Finally, the Support Vector Machine obtains the best results among all of the clustering and classification results. The classification results are usually better than clustering results regardless of which entity embedding or n-gram model is being used.

Chapter 5: DBpedia Invalid Entity Detection in Resources Description

This chapter describes the task of DBpedia invalid entity detection in resource description, including the methods and the experimental setups for this task, and detailed evaluation results for this task.

The task of DBpedia invalid entity detection in resource description is to detect invalid DBpedia entities in the DBpedia resource descriptions. There are some examples of invalid entities. For instance, consider the entity "dbr:Cake" where the resource description contains triples related to types of Food or ingredients. In the "dbr:Cake" description, we found one invalid entity "dbr:Ernest_Dichter." "dbr:Ernest_Dichter" is a psychologist and thus should be instead related to "dbo:Person" and "dbo:Agent" for instance. Similarly, in "dbr:Farmer," where entities are about "dbo:Agriculture," we identified "dbr:Iowa_State_University" as an outlier, as it has the type "dbo:University" and is related to "dbo:School," "dbo:Education" for instance.

At the time of our experiments, there were several outliers and invalid facts in some resources descriptions. However, DBpedia is a knowledge base that is updated and cleaned on a regular basis. Some of the invalid entities detected during our experiments were removed from DBpedia resources description, such as the invalid entity "dbr:Vanilla_Ice" which was removed from the "dbr:Earthquake" resource description. Based on our latest experiments, most (90%) of the DBpedia entities do not currently have any invalid entities Thus, to be able to evaluate the interest of our approach, we built an artificial dataset by adding noisy/invalid triples in randomly selected entities. The objective of this task is to detect whether the clustering and classification algorithms can detect this external noise. For instance, in the "dbr:Cake" resource description, the external noise entities such as "dbr:FreeBSD," and "dbr:Hydrazine," and "dbr:Johns_Hopkins_University" etc. have been indeed detected as invalid entities. The original entities from the "Cake" resource description, such as "dbr:Cupcake," "dbr:Butter," and "dbr:Sprinkles" etc. have been detected as valid entities. Similarly, our approach has been able to discriminate, for the "dbr:Farmer" resource description, between invalid entities, such as "dbr:Solar_Wind," "dbr:Linux," and

“dbr:Fibre_Channel” and valid entities such as“dbr:Farm,” “dbr:Local_Food,” and “dbr:Agriculture”.

5.1 Experiment Setup

Similarly to our previous tasks, this task consists of two parts: clustering and classification. Both of them use clustering and classification algorithms applied on the prepared dataset for each of the DBpedia entities. In both cases, we perform a binary clustering and a binary classification that represent two main categories for each entity of interest: the entity’s category and NOT the entity’s category (the positive class / cluster, and the negative class / cluster). For example, for the entity “dbr:Barack_Obama,” we want to classify entities that occur as objects in the RDF description of “dbr:Barack_Obama” as either valid or invalid.

The instances for the entity’s category (positive class / cluster) are the entities that are extracted from the DBpedia resources descriptions, for example, the positive instances for the entity “Barack_Obama” are extracted from the DBpedia resource description of “Barack_Obama.” such as “dbr:Occidental_College,” “dbr:Hawaii,” and “dbr:Illinois”. The invalid entities (negative class / cluster) are selected randomly from all of the remaining entities, except the entities that share the same class of the positive instances. For example, the “dbr:Illinois” has five classes “dbo: Place,” “dbo: Location,” “dbo: AdministrativeRegion,” “dbo: PopulatedPlace,” and “dbo: Region.” We do not select any entities (resources) that are associated with these classes, such as “dbr: New_Jersey,” “dbr: Los_Angeles,” and “dbr: Florida” etc. The number of negative instances depends on the corresponding number of positive instance in order to build a balanced dataset. The dataset is divided into training part and testing set and we perform a 10-fold cross validation. Finally, we report the evaluation results for both the positive and negative class.

5.2 Clustering Evaluation

In the Clustering evaluation part, there are two subsections, namely, invalid entity detection in resources description with classification on n-grams models and entity embedding models.

5.2.1 Clustering with N-gram Models

The table 5.1 to table 5.4 show the clustering results of the 2573 DBpedia entities with the uni-gram, bi-gram, tri-gram, and n-gram with n=1~3 for the results of the positive class.

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.726	0.760	0.681	64.0%
Mean Shift	0.520	0.010	0.020	48.1%
Birch	0.697	0.773	0.671	61.5%

Table 5. 1. Summary of the average clustering results with the uni-gram model for positive class

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.528	0.922	0.647	50.1%
Mean Shift	0.524	0.016	0.030	39.4%
Birch	0.513	0.956	0.656	50.1%

Table 5. 2. Summary of the average clustering results with the bi-gram model for positive class

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.533	0.900	0.646	51.1%
Mean Shift	0.272	0.006	0.011	42.1%
Birch	0.535	0.905	0.646	50.9%

Table 5. 3. Summary of the average clustering results with the tri-gram model for positive class

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.571	0.873	0.649	52.4%
Mean Shift	0.580	0.021	0.037	42.9%
Birch	0.502	0.993	0.665	50.0%

Table 5. 4. Summary of the average clustering results with the n-gram model for positive class

Based on the results of the uni-gram, bi-gram, tri-gram and n-gram models for the results of positive class, the bi-gram, tri-gram and n-gram models have similar performance; when we look at the detailed results of these three models, the n-gram model performs slightly better than the tri-gram model, followed by the bi-gram model. The uni-gram has the best performance among all

the n-gram models, but its performance is not significantly better than other three models; it performs only with around 2% better than the other three models in terms of F-score and Accuracy. Like in the case of the entity embedding models in the type identification task, K-means and Birch have similar performance, and they perform much better than Mean Shift regardless of which n-gram model is being used. For the results of all n-gram models, their performance is much better than that of Mean Shift, and the F-score and accuracy for Means Shift are extremely low; the algorithm cannot be used for detecting invalid entities in resources descriptions.

Table 5.5 to table 5.8 show the average results of the negative class for the 2573 DBpedia entities with four different n-gram models: uni-gram, bi-gram, tri-gram, and n-gram with n=1~3 of the negative class.

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.549	0.521	0.457	64.0%
Mean Shift	0.505	0.993	0.670	48.1%
Birch	0.507	0.457	0.415	61.5%

Table 5. 5. Summary of the average clustering results with the uni-gram model for negative class

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.393	0.081	0.073	50.1%
Mean Shift	0.500	0.989	0.664	39.4%
Birch	0.427	0.045	0.046	50.1%

Table 5. 6. Summary of the average clustering results with the bi-gram model for negative class

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.473	0.121	0.127	51.1%
Mean Shift	0.494	0.993	0.659	42.1%
Birch	0.496	0.113	0.121	50.9%

Table 5. 7. Summary of the average clustering results with the tri-gram model for negative class

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.409	0.175	0.152	52.4%
Mean Shift	0.509	0.991	0.672	42.9%
Birch	0.062	0.007	0.007	50.0%

Table 5. 8. Summary of the average clustering results with the n-gram model for negative class

Overall, we can notice that the clustering algorithms are not very good at detecting noisy/invalid entities. The n-gram model performs slightly better than the tri-gram model, followed by the bi-gram model. The uni-gram has the best performance among all the n-gram models, and its performance is significantly better than other three models. Unlike the results for positive class, Mean Shift performs better than K-means and Birch for the results of negative class.

5.2.2 Clustering with Entity Embedding Models

We tested the two entity embedding models Skip-gram and CBOW in these experiments as well. Table 5.9 and 5.10 show the average clustering results of the 2573 DBpedia entities with the Skip-gram and CBOW entity embedding models of the positive class.

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.894	0.930	0.906	90.2%
Mean Shift	0.064	0.003	0.005	49.5%
Birch	0.887	0.905	0.885	88.0%

Table 5. 9. Summary of the average clustering results with the Skip-gram entity embedding model for positive class

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.877	0.901	0.881	87.8%
Mean Shift	0.002	0.000	0.000	49.9%
Birch	0.861	0.869	0.851	84.8%

Table 5. 10. Summary of the average clustering results with the CBOW embedding model for positive class

Based on the evaluation results on the two entity embedding models for the results of positive class, the performance of Skip-grams is better than CBOW.

Furthermore, K-means has similar performance with Birch regardless of which entity embedding model is being used, and their performance are much better than that of Mean Shift with the Skip-gram and CBOW entity embedding models, K-means performs with around 2% better than Birch in terms of F-score and Accuracy.

Table 5.11 and 5.12 show the average clustering results with the Skip-gram and CBOW entity embedding models of the negative class

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.931	0.874	0.894	90.2%
Mean Shift	0.496	0.992	0.661	49.5%
Birch	0.906	0.854	0.864	88.0%

Table 5. 11. Summary of the average clustering results with the Skip-gram entity embedding model for negative class

Algorithm	Precision	Recall	F-Score	Accuracy
K-means	0.905	0.856	0.871	87.8%
Mean Shift	0.499	0.999	0.665	49.9%
Birch	0.878	0.828	0.836	84.8%

Table 5. 12. Summary of the average clustering results with the CBOW embedding model for negative class

Based on the evaluation results on the two entity embedding models for the results of negative class, the performance of Skip-grams is better than CBOW.

Furthermore, K-means and Birch have similar performance, and they perform much better than Mean Shift regardless of which entity embedding model is being used. With the Skip-gram and CBOW entity embedding models, K-means and Birch are with around 20% better than Mean Shift. Based on the overall clustering results for both positive and negative classes, the Skip-gram model has the best performance among all of the entity embedding and n-gram models, followed by CBOW model. K-means and Birch have similar performance and they perform much better than Mean Shift except for the results of negative class with the four n-gram models. With the Skip-gram and CBOW entity embedding models, K-means and Birch clustering algorithms are able to detect most of the external noisy entities.

5.3 Classification Evaluation

In the classification evaluation part, there are two subsections, namely, invalid entity detection in resources description with classification on n-grams models and entity embedding models. We report the results by 10-fold cross-validation for the classification experiment.

5.3.1 Classification Evaluation with N-gram Models

The table 5.13 to table 5.16 show the classification result with the uni-gram, bi-gram, tri-gram and n-gram models of the positive class.

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.503	0.499	0.492	49.4%	0.499
Decision Tree	0.786	0.774	0.770	77.5%	0.778
Extra Tree	0.874	0.757	0.796	81.3%	0.821
KNN	0.789	0.589	0.590	69.2%	0.700
Random Forest	0.825	0.684	0.730	75.9%	0.767
Naïve Bayes	0.912	0.920	0.912	91.2%	0.914
SVM	0.914	0.814	0.832	84.6%	0.860

Table 5. 13. Summary of the average classification results on the uni-gram model for positive class

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.500	0.500	0.491	49.6%	0.501
Decision Tree	0.693	0.635	0.645	66.2%	0.668
Extra Tree	0.810	0.595	0.643	69.5%	0.710
KNN	0.591	0.528	0.443	56.4%	0.572
Random Forest	0.764	0.554	0.600	66.2%	0.677
Naïve Bayes	0.848	0.871	0.854	85.4%	0.856
SVM	0.622	0.541	0.459	55.6%	0.602

Table 5. 14. Summary of the average classification results on the bi-gram model for positive class

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.494	0.494	0.485	49.1%	0.496
Decision Tree	0.733	0.547	0.577	63.7%	0.645
Extra Tree	0.775	0.495	0.513	61.9%	0.634
KNN	0.456	0.355	0.310	55.8%	0.563
Random Forest	0.743	0.472	0.483	60.3%	0.619
Naïve Bayes	0.783	0.811	0.789	78.8%	0.791
SVM	0.763	0.629	0.585	63.2%	0.668

Table 5. 15. Summary of the average classification results on the tri-gram model for positive class

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.501	0.499	0.491	49.5%	0.500
Decision Tree	0.784	0.767	0.765	77.0%	0.773
Extra Tree	0.870	0.744	0.784	80.3%	0.812
KNN	0.652	0.786	0.633	61.6%	0.621
Random Forest	0.819	0.680	0.724	75.3%	0.762
Naïve Bayes	0.908	0.888	0.892	89.5%	0.896
SVM	0.530	0.528	0.420	53.3%	0.579

Table 5. 16. Summary of the average classification results on the n-gram model for positive class

Based on the classification results on traditional n-gram models for positive class, the uni-gram model has the best performance among all of the n-gram models. The bi-gram, tri-gram and n-gram models have similar performance. When we look at the detailed results of these three models, the n-gram model performs slightly better than the tri-gram model, and the tri-gram model also performs slightly better than the bi-gram model. Uni-grams' performance is significantly better than the bi-gram and tri-gram models; it performs with around 20% better than the bi-gram and tri-gram models. However, the difference between the uni-gram and n-gram models is only around 5%. Naïve Bayes has the best results among all the classification algorithms regardless of which n-gram model is being used. One interesting point is that when switching from uni-gram to bi-gram, tri-gram, and n-gram models, the results of Support Vector Machine drop significantly, from 0.85 to 0.6 in terms of F-Score and Accuracy. The difference between them is around 25%.

The table 5.17 to table 5.20 show the classification result with the uni-gram, bi-gram, tri-gram and n-gram models of the negative class.

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.495	0.500	0.488	49.4%	0.500
Decision Tree	0.783	0.783	0.773	77.5%	0.778
Extra Tree	0.789	0.886	0.823	81.3%	0.821
KNN	0.725	0.812	0.698	69.2%	0.700
Random Forest	0.737	0.849	0.777	75.9%	0.767
Naïve Bayes	0.920	0.908	0.909	91.2%	0.914
SVM	0.843	0.906	0.849	84.6%	0.860

Table 5. 17. Summary of the average classification results on the uni-gram model for negative class

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.501	0.501	0.492	49.6%	0.501
Decision Tree	0.661	0.701	0.665	66.2%	0.668
Extra Tree	0.685	0.824	0.718	69.5%	0.710
KNN	0.530	0.616	0.483	56.4%	0.572
Random Forest	0.654	0.799	0.692	66.2%	0.677
Naïve Bayes	0.871	0.841	0.850	85.4%	0.856
SVM	0.502	0.664	0.495	55.6%	0.602

Table 5. 18. Summary of the average classification results on the bi-gram model for negative class

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.497	0.497	0.488	49.1%	0.496
Decision Tree	0.638	0.743	0.642	63.7%	0.645
Extra Tree	0.646	0.773	0.631	61.9%	0.634
KNN	0.542	0.772	0.577	55.8%	0.563
Random Forest	0.620	0.766	0.616	60.3%	0.619
Naïve Bayes	0.807	0.770	0.780	78.8%	0.791
SVM	0.691	0.707	0.596	63.2%	0.668

Table 5. 19. Summary of the average classification results on the tri-gram model for negative class

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.500	0.502	0.492	49.5%	0.500
Decision Tree	0.774	0.779	0.766	77.0%	0.773
Extra Tree	0.782	0.879	0.814	80.3%	0.812
KNN	0.635	0.457	0.431	61.6%	0.621
Random Forest	0.732	0.843	0.770	75.3%	0.762
Naïve Bayes	0.893	0.905	0.894	89.5%	0.896
SVM	0.411	0.630	0.455	53.3%	0.579

Table 5. 20. Summary of the average classification results on the n-gram model for negative class

Based on the classification results on the four n-gram models for the negative class, the uni-gram model has the best performance among all of the uni-gram, bi-gram, tri-gram and n-gram models. When we look at the detailed results of these three models, the n-gram model performs slightly better than the tri-gram model, and the tri-gram model also performs slightly better than the bi-gram model. The difference between the uni-gram and n-gram models is only around 10% in terms of F-score and Accuracy. Naïve Bayes has the best results among all the classification algorithms regardless of which n-gram model is being used. One interesting point is that when switching from uni-gram to bi-gram, tri-gram, and n-gram models, the results of Support Vector Machine (SVM) drop significantly, from 0.85 to 0.55 in terms of F-Score and from 86% to 58% in terms of Accuracy.

5.3.2 Classification Evaluation on Entity Embedding Models

Tables 5.21 and 5.22 show the classification results on the 2573 DBpedia entities on the Skip-gram and CBOW entity embedding models for positive class.

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.500	0.502	0.493	49.7%	0.501
Decision Tree	0.838	0.832	0.830	83.2%	0.833
Extra Tree	0.931	0.886	0.904	90.8%	0.910
KNN	0.935	0.949	0.939	93.9%	0.940
Random Forest	0.924	0.879	0.896	90.1%	0.903
Naïve Bayes	0.937	0.944	0.938	93.9%	0.939
SVM	0.944	0.958	0.948	94.8%	0.949

Table 5. 21. Summary of the average classification results with the Skip-gram entity embedding model for positive class

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.499	0.504	0.492	49.7%	0.501
Decision Tree	0.786	0.778	0.774	77.9%	0.782
Extra Tree	0.901	0.824	0.855	86.4%	0.866
KNN	0.894	0.923	0.900	89.4%	0.896
Random Forest	0.893	0.813	0.845	85.5%	0.858
Naïve Bayes	0.922	0.925	0.921	92.2%	0.923
SVM	0.932	0.939	0.933	93.3%	0.934

Table 5. 22. Summary of the average classification results with the CBOW entity embedding model for positive class

Based on the results on 2573 DBpedia entities on the Skip-gram and CBOW Entity embedding models for positive class, the Skip-gram model performs slightly better than the CBOW model, the Skip-gram model is with around 5% better than the CBOW model. The baseline algorithm get an Accuracy of around 50% for both models, which is same as random choice. For the results on the Skip-gram entity embedding models, K Nearest Neighbor, Naïve Bayes, and Support Vector Machine get very close performance, and their performance are better than other classifiers, their F-Score and Accuracy are more than or close to 0.95, which can detect most of the DBpedia entities correctly. When switching to CBOW model, performance of each algorithm drops around 10%, except for the baseline algorithm, which has an Accuracy that is still around 50%.

Tables 5.23 and 5.24 show the classification results on the 2573 DBpedia entities on the Skip-gram and CBOW entity embedding models of the negative class.

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.503	0.501	0.493	49.7%	0.501
Decision Tree	0.835	0.833	0.828	83.2%	0.833
Extra Tree	0.892	0.934	0.909	90.8%	0.910
KNN	0.949	0.931	0.937	93.9%	0.940
Random Forest	0.886	0.927	0.902	90.1%	0.903
Naïve Bayes	0.946	0.934	0.937	93.9%	0.939
SVM	0.957	0.941	0.947	94.8%	0.949

Table 5. 23. Summary of the average classification results with the Skip-gram entity embedding model for negative class

Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Baseline	0.504	0.499	0.493	49.7%	0.501
Decision Tree	0.784	0.785	0.778	77.9%	0.782
Extra Tree	0.839	0.909	0.868	86.4%	0.867
KNN	0.923	0.869	0.878	89.4%	0.896
Random Forest	0.830	0.903	0.868	85.5%	0.858
Naïve Bayes	0.928	0.921	0.921	92.2%	0.923
SVM	0.938	0.929	0.931	93.3%	0.934

Table 5. 24. Summary of the average classification results with the CBOW entity embedding model for negative class

Based on the results on 2573 DBpedia entities on the Skip-gram and CBOW Entity embedding models for the negative class, the Skip-gram model performs slightly better than the CBOW model, the difference is around 3% in terms of F-score and Accuracy. For the results on the Skip-gram entity embedding models, K-Nearest- Neighbor, Naïve Bayes, and Support Vector Machine (SVM) have high performance, their F-Score is around 0.95 (and the accuracy around 95%); these classification algorithms can detect most of external noisy entities correctly.

5.4 Synthesis and Discussion

For the invalid resources identification, overall, the Skip-gram model obtains the best results among all of the entity and n-gram models for both the positive and negative class, followed by the CBOW model, and then the uni-gram and n-gram model, regardless of which clustering or classification algorithm is being used. The n-gram model obtains a similar performance with the uni-gram model, followed by bi-gram and tri-gram models.

The classification algorithms perform better than clustering algorithms regardless of which entity embedding or n-gram model is used. The SVM classifier gets the best results among all of the classification algorithms on the Skip-gram entity embedding model for the both positive and negative classes. However, when switching to n-gram models, the Naïve Bayes classifier gets the best results. Furthermore, the performance of SVM varies with different models, with the Skip-gram, CBOW and uni-gram models, SVM has a similar performance with Naïve Bayes. However, when switching to the uni-gram, bi-gram and tri-gram models, the performance of SVM drops significantly. The Naïve Bayes, K-Nearest-Neighbor and Support Vector Machine have good performance with entity embedding models for the results of both positive and negative classes, thus there are three algorithms are suitable for the task of DBpedia invalid entity detection in resource description.

Chapter 6 Discussion and Conclusion

In this thesis, we addressed the tasks of building our own entity embedding and n-gram models for DBpedia quality enhancement by detecting invalid DBpedia types, completing missing DBpedia types, and detecting invalid DBpedia entities. We compared the results of different clustering and classification algorithms, and the results of different entity embedding and n-gram models.
The section 6.1 presents results of applying the Random Forest classification model on the whole DBpedia knowledge base. The section 6.2 presents our conclusions and recommendations for future work.

6.1 Applying the Random Forest classification model on the whole DBpedia Knowledge Base

To test the interest of the models learned in this thesis, we decided to apply one of our classification models to the whole DBpedia knowledge base. We performed several types of experiments: We experimented with the identification of new and similar types for entities in the DBpedia knowledge base to both ascertain the interest and accuracy of our classification procedure and the identification of wrong types for already available *rdf:type* statements.

Based on our experiments with the three clustering and six classification algorithms, we came to the conclusion that SVM had the best overall performance when couple with the skip-grams model. Random Forest has good overall performance, and most of its results are close to Support Vector Machine. However, Support Vector Machine is slow compared to Random Forest. By taking into account both performance and efficiency, we decided to use Random Forest classifier for experiments on the whole DBpedia Knowledge Base.

There are around 4.7 million of resources in the current DBpedia. We checked the availability of these entities in our embedding model. We found that 1.4 million of these entities were represented

by a vector in our model, with a coverage rate of around 30%. Compared to the pre-built model[20] that we first used, where only 0.22 million of them were available with a coverage around 5%, this is better and justifies the interest of our own word2vec model. Our explanation for absence of the remaining 3.3 million entities is the use of a threshold of 5 for taking into account an entity (an entity that occurs less than 5 times is ignored).

The first experiment on the whole DBpedia knowledge base tested how many entities have new types based on our methods. The experiment was processed as follows: for each DBpedia entity from the 1.4 million entities, we tested the entity with the 358 classifiers previously trained, then compared the classification results with the types already available in the DBpedia knowledge base for these entities. For example, "dbr:Donald_Trump" originally has two classes, "dbo:Person" and "dbo:Agent" in the DBpedia knowledge base, Our classification procedure discovered four classes for the entity "dbr:Donald_Trump:" "dbo:Person," "dbo:President," "dbo:Politician," and "dbo:President" based on our experiment results. Thus, the new types are "dbo:President" and "dbo:Politician." Based on the results on the whole DBpedia knowledge base, 80,797 out of 1,406,828 entities were associated to new types. There are around two types per entity on average. The percentage of entities with new types is 5.74%.

Table 6.1 shows the detailed results for the top five DBpedia classes that have the biggest number of new DBpedia entities.

DBpedia Class	Number of new entities	Percentage of new entities
Animal	2381	0.169%
Eukaryote	2869	0.204%
Person	2291	0.163%
Place	3399	0.242%
Species	3002	0.213%

Table 6. 1. DBpedia new entity based on the whole DBpedia knowledge base

[20] https://github.com/idio/wiki2vec

Figure 6.1 shows a set of new rdf:type triples discovered using our methods.

```
<dbr:Wright_R-540 rdf:type dbo:Aircraft>

<dbr:Asian_Infrastructure_Investment_Bank rdf:type dbo:Bank>

<dbr:Hernando rdf:type dbo:Place>

<dbr:Air_China_Flight_129 rdf:type dbo:Airline>

<dbr:New_Imperial_Hotel rdf:type dbo:Hotel>
```

Figure 6. 1. Example of new rdf:type triples

The second experiment counted how many entities had the same types as shown on DBpedia based on our methods. The experiment was processed as follows: for each DBpedia entity, we tested the entity with the 358 classifiers, then compared the classification results with the DBpedia knowledge base. For example, "dbr:Barack_Obama" originally has four classes: "dbo:Person," "dbo:Agent," "dbo:Politician," and "dbo:President." Based on our classification results, the four types were correctly mapped to the entity "dbr:Barack_Obama" using our classifiers. Based on the results on the whole DBpedia knowledge base, 1,144,719 out of 1,406,828 entities have the same types as on the DBpedia knowledge base. The percentage of the entities that have the same types is 81.37%. This shows the completeness of our classification methods and the interest of using our classifiers to automatically identify the types of new resources that will emerge in Wikipedia and DBpedia.

Table 6.2 shows the detailed results for the top five DBpedia classes that have the highest number of similar entities as shown on DBpedia based on our methods.

DBpedia Class	Number of same entities	Percentage of similar entities
CareerStation	10001	0.711%
Document	9983	0.710%
PersonFunction	9958	0.708%
Sound	9856	0.706%
SportsTeamMember	9833	0.699%

Table 6. 2. DBpedia same entity based on the whole DBpedia knowledge base

The third experiment counted how many classes have new entities based on our methods. The experiment was processed as follows: for each DBpedia class, we tested the class with the entities from the whole DBpedia knowledge base based on the 358 classifiers we trained before. Then we compared the classification results with the DBpedia knowledge base to find how many classes have new entities. For example, the class "dbo:Politician" originally has more than a hundred thousand entities, such as "dbr:Barack_Obama," "dbr:George_Bush," and "dbr:Bill_Clinton" etc. Based on the experiment results, "dbr:Donald_Trump" and "dbr: Rex_Tillerson" were added as new entities to the class "dbo:Politician." Based on the results on the whole DBpedia knowledge base, 358 out of 358 classes have new entities. The percentage of classes that have new entities is 100%.

The fourth experiment tested the number of invalid RDF triples through the predicate *rdf:type* based on our methods. The experiment was processed as follows: for each RDF triple using the predicate *rdf:type* : <Subject rdf:type Object> in the DBpedia knowledge base, we ran the 358 classifiers to classify the Subject and compared the output to the object . For example, based on our classification results, the Subject "dbr:Xanthine" is classified correctly with the Objects "dbo:ChemicalSubstance" and "ChemicalCompound." Thus the RDF triples <dbr:Xanthine rdf:type dbo:ChemicalSubstance> and <dbr:Xanthine rdf:type dbo: ChemicalCompound> are classified as valid RDF triples. However, the classification results do not find the type "dbo:Airport". Thus, the RDF triple <dbr:Xanthine rdf:type dbo:Airport > is classified as an invalid RDF triple. Among the 4,161,452 rdf:type triples that were tested, 968,944 were detected as invalid triples. The overall percentage of invalid RDF triples is 23.28%.

Figure 6.2 shows several invalid triples detected by our methods.

```
<dbr:Wright_R-540 rdf:type dbo:TelevisionShow>
<dbr:Asian_Infrastructure_Investment_Bank rdf:type dbo:University>
<dbr:African_Investment_Bank rdf:type dbo:University>
<dbr:Air_China_Flight_129 rdf:type dbo:ArtificialSatellite>
<dbr:Lufthansa_Flight_615 rdf:type dbo:Band>
```

Figure 6. 2. Example of invalid triples

Table 6.3 shows the detailed results for the top five DBpedia classes that have the biggest number of invalid triples.

DBpedia Class	Number of invalid RDF triples	Percentage of invalid RDF triples
ComicsCharacter	6670	1.603%
Language	10456	2.513%
Person	5538	1.331%
Place	7354	1.767%
Species	16027	3.852%

Table 6. 3. DBpedia invalid RDF triples

6.2 Conclusion and Future Work

In the DBpedia entity type detection part (Chapter 4), the experiments show that the system we built can detect most of the invalid DBpedia types correctly, and can help us to complete missing types for un-typed DBpedia resources. In the DBpedia invalid entity detection part (Chapter 5), our experimental results show that the system we built can detect most of the invalid DBpedia entities correctly. In the experiments in both parts (Chapter 4 and Chapter 5), we chose the best models and algorithms, and we presented further detailed results for a sample of DBpedia types and entities. Overall, the Skip-gram entity embedding model has the best results among all of the entity embedding and the n-gram models.

In the experiments in DBpedia Entity Type Detection (Chapter 4), Support Vector Machine has the best overall performance among all of the clustering and classification algorithms. In terms of the clustering part, K-means and Birch usually have similar performance. In terms of the classification part, Extra Tree, Random Forest, and SVM have robust performance among all of the entity embedding and the n-gram models on both training and test datasets. Naïve Bayes classifier has performance close to Support Vector Machine on the training dataset, but the performance drops significantly when switching to the test dataset.

In the experiments in DBpedia Invalid Entity Detection in Resources Description (Chapter 5), Naïve Bayes has the best overall performance among all of the clustering and classification algorithms. In terms of the clustering part, K-means and Birch usually perform much better than Mean Shift. In terms of the classification part, K Nearest Neighbor (KNN), Random Forest, and Naïve Bayes have robust performance among all of the entity embedding and n-gram models. Support Vector Machine classifier performs similarly to K Nearest Neighbor and Naïve Bayes on the Skip-gram model, CBOW and uni-gram models. The performance drops significantly when switching to the bi-gram, tri-gram and n-gram models.

Recalling the four research questions we proposed before:

RQ1: Can entity embeddings help detect the relevant types of an entity?
The answer to the first research question is yes. The entity embeddings can help detect relevant types of an entity. Based on the results with Support Vector Machine, more than 98% of the relevant types of an entity can be detected with the Skip-gram entity embedding model.

RQ2: How do entity embeddings compare with traditional n-gram models for type identification?
The answer to the second research question is not all of the entity embedding models perform better than traditional n-gram models. The Skip-gram entity embedding model has the best performance among all of the entity embedding and n-gram models. However, n-gram and uni-gram models usually perform better than the Continuous Bag-Of-Words (CBOW) entity embedding model.

RQ3: Can entity embeddings help detect the relevant entities in the RDF description of a DBpedia resource?
The answer to the third research question is yes. The entity embeddings can help detect the relevant entities in the RDF description of a DBpedia resource. Based on the results with Support Vector Machine, around 95% of the relevant entities in the RDF description of a DBpedia resource can be detected with the Skip-gram entity embedding model.

RQ4: How do entity embeddings compare with traditional n-gram models for invalid entity detection?

The answer to the fourth research question is all of the entity embedding models perform better than the traditional n-gram models. The Skip-gram entity embedding model has the best performance among all of the entity embedding and n-gram models. Furthermore, the Continuous Bag-Of-Words entity embedding model also performs better than n-gram, uni-gram, bi-gram, and tri-gram traditional models.

There are still some limitations to our approach. The first limitation is that vector representation of entities is the only feature for the description of DBpedia entities. Features based on the DBpedia knowledge base or extracted from the Wikipedia abstracts or complete pages could be used to enhance the performance of our classifiers. Another limitation is that our entity embedding models do not contain all of the DBpedia entities as there are many DBpedia entities missing from our entity embedding models even if there are more complete than the ones available in the state of the art. In terms of comparison with results obtained in the state of the art, our initial plan was to compare our methods with SDValide and SDType (Paulheim and Bizer, 2014). However, we were not able to run the code provided despite repeated efforts. Similarly, the authors did not share their specific datasets extracted from DBpedia for us to be able to use the same gold standards.

The approach for detecting the validity of an RDF triple in a resource description could be significantly enhanced. For example, we did not use the properties or predicates to identify the validity of an RDF triple. Most importantly, we relied on the available DBpedia descriptions as our gold standard Given that these descriptions might contain incorrect statements, it is not clear how this will impact the discovery of invalid triples in unknown resources.

Another limitation is that we only built classifiers based on 358 DBpedia classes. There are 199 DBpedia classes that do not have entities, and for which we cannot build classifiers based on our current method.

The first work for the future is to use more features rather than a single vector representation of entities. One important aspect would be to identify how properties can be exploited to assess the validity of an RDF triple rather than relying only on the semantic distance between the subject and

object based on the obtained vectors. Furthermore, how to build classifiers of these 199 ontological classes without any entities in DBpedia is important work for the future. One solution for solving the problem is to add some entities to these 199 classes manually and build classifiers for these classes or to compute a semantic distance between vectors of entities and vectors representing these classes to try to suggest entities. Another important direction of work for the future is to build larger entity embedding models to include more DBpedia entities, and use the same approaches to detect invalid DBpedia types, complete missing types, and detect invalid entities. In terms of the entity embedding model building, we plan to try other entity embedding tools, such as GloVe (Pennington et al., 2014) to see if they can lead to better models.

References

Abdi, H., & Williams, L. J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*.

Abele, A., McCrae, J P., Buitelaar P., Jentzsch A., & Cyganiak. R. (2017). "Linking Open Data cloud diagram 2017. Retrieved from http://lod-cloud.net/

Alani, H., Kim, S., Millard, D. E., Weal, M. J., Hall, W., Lewis, P. H., & Shadbolt, N. R. (2003). Automatic Ontology-Based Knowledge Extraction from Web Documents. *IEEE Intelligent Systems*, *18*(1), 14–21.

Apte, C., & Weiss, S. (1997). Data mining with decision trees and decision rules. Data Mining, 13(2–3), 197–210.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z. (2007). DBpedia: A nucleus for a Web of Open Data. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 4825 LNCS, pp. 722–735).

Bengio, Y., Ducharme, R., Vincent, P., Janvin, C. (2003). A Neural Probabilistic Language Model. Journal of Machine Learning Research, 3, 1137–1155.

Bengio, Y., Schwenk, H., Senécal, J. S., Morin, F., & Gauvain, J. L. (2006). Neural probabilistic language models. *Studies in Fuzziness and Soft Computing*, *194*, 137–186.

Bennett, K., & Bredensteiner, E. (2000). Duality and geometry in SVM classifiers. Icml, 57–64.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. Scientific American, 284, 34–43.

Bizer, C., Heath, T., & Berners-Lee, T. (2009a). Linked data-the story so far. International Journal on Semantic Web and Information Systems, 5(3), 1–22.

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., & Hellmann, S. (2009b). DBpedia - A crystallization point for the Web of Data. Journal of Web Semantics.

Bollacker, K., Cook, R., & Tufts, P. (2007). Freebase: A shared database of structured general human knowledge. *Proceedings of the National Conference on Artificial Intelligence*, *22*(2), 1962.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., & Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. *SIGMOD 08 Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 1247–1250.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Brown, S. D., & Myles, A. J. (2010). Decision Tree Modeling in Classification. In *Comprehensive Chemometrics* (Vol. 3, pp. 541–569).

Chen, T., Tang, L. A., Sun, Y., Chen, Z., Zhang, K. (2016). Entity embedding-based anomaly detection for heterogeneous categorical events. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) 2016 (Vol. 2016–January, pp. 1396–1403).

Chen, X., & Ishwaran, H. (2012). Random forests for genomic data analysis. Genomics.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P. (2011). Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research, 12, 2493–2537.

Debattista J., Lange C., Auer S. (2016). A Preliminary Investigation Towards Improving Linked Data Quality Using Distance-Based Outlier Detection. In: Li YF. et al. (eds) Semantic Technology. JIST 2016. Lecture Notes in Computer Science, vol 10055. Springer, Cham.

Derpanis, K. G. (2005). Mean Shift Clustering. Computer, 1(x), 1–3.

Erxleben, F., Günther, M., Krötzsch, M., Mendez, J., & Vrandečić, D. (2014). Introducing wikidata to the linked data web. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 8796, pp. 50–65).

Fabian, M., Gjergji, K., & Gerhard, W. (2007). YAGO: A core of semantic knowledge unifying wordnet and wikipedia. 16th International World Wide Web Conference, 697–706.

Fagerberg, J., Fosaas, M., & Sapprasert, K. (2012). Innovation: Exploring the knowledge base. Research Policy, 41(7).

Färber, M., Ell, B., Menne, C., & Rettinger, A. (2015). A Comparative Survey of DBpedia, Freebase,OpenCyc,Wikidata,And YAGO. *Semantic Web*, *1*, 1–5.

Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861–874.

Fellbaum, C. (2010). WordNet. In *Theory and Applications of Ontology: Computer Applications* (pp. 231–243).

Fleischhacker, D., Paulheim, H., Bryl, V., Völker, J., & Bizer, C. (2014). Detecting errors in numerical linked data using cross-checked outlier detection. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 8796, pp. 357–372).

Font, L., Zouaq, A., & Gagnon, M. (2015). Assessing the Quality of Domain Concepts Descriptions in DBpedia. In Proceedings - 11th International Conference on Signal-Image Technology and Internet-Based Systems, SITIS 2015 (pp. 254–261).

Font, L., Zouaq, A., & Gagnon, M. (2017), Assessing and Improving Domain Knowledge Representation in DBpedia. Open Journal of Semantic Web (OJSW), 4(1): p. 1-19.

Gangemi, A., Nuzzolese, A., Presutti, V., Draicchio, F., Musetti, A., & Ciancarini, P. (2012). Automatic Typing of DBpedia Entities. The Semantic Web – ISWC 2012 SE - 5, 7649, 65–81.

Ganguly, D., Roy, D., Mitra, M., Jones, G. J. F. (2015). Word Embedding Based Generalized Language Model for Information Retrieval. In Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, 795–798.

Garcia, A., Szomszor, M., Alani, H., & Corcho, O. (2009). Preliminary results in tag disambiguation using DBpedia. Information Retrieval, 39 Suppl 8(1–2), 41–44.

Genuer, R., Poggi, J.-M., & Tuleau-Malot, C. (2010). Variable selection using random forests. *Pattern Recognition Letters*, *31*(14), 2225–2236.

Genuer, R., Poggi, J.-M., & Tuleau-Malot, C. (2015). VSURF: An R Package for Variable Selection Using Random Forests. *The R Journal*, *7*(2), 19–33.

Geurts P., Ernst D., &and Wehenkel L. (2006), "Extremely randomized trees", Machine Learning, 63(1), 3-42, 2006.

Goldberg, Y., Levy, O: Word2vec Explained: Deriving Mikolov et al. (2014). Negative-Sampling Word-Embedding Method. arXiv Preprint arXiv:1402.3722, (2), 1–5.

Guo, G., Wang, H., Bell, D., Bi, Y., & Greer, K. (2003). kNN Model-Based Approach in Classification. On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, 2888, 986–996.

Guthrie, D., Allison, B., & Liu, W. (2006). A closer look at skip-gram modelling. Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC-2006), 1222–1225.

Hahm, Y., Park, J., Lim, K., Kim, Y., Hwang, D., & Choi, K. (2014). Named Entity Corpus Construction using Wikipedia and DBpedia Ontology. Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014), 2565–2569.

Haidar-Ahmad, L., Font, L., Zouaq, A., & Gagnon, M. (2016). Entity typing and linking using SPARQL patterns and DBpedia. In Communications in Computer and Information Science (Vol. 641, pp. 61–75).

Han L, Embrechts M, Szymanski B, Sternickel K, Ross A. (2006). Random Forests Feature Selection with Kernel Partial Least Squares: Detecting Ischemia from MagnetoCardiograms. In Proceedings of the European Symposium on Artificial Neural Networks. Burges, Belgium; 221–226.

Han, J., Kamber, M., Pei, (2012). Data Mining: Concepts and Techniques. 3rd edn. Morgan Kaufmann. San Francisco, CA.

Hartigan, J. A., & Wong, M. A. (1979). A K-Means Clustering Algorithm. Applied Statistics, 28(1), 100–108.

Hellmann, S., Filipowska, A., Barriere, C., Mendes, P. N., & Kontokostas, D. (2013). NLP & DBpedia: An upward knowledge acquisition spiral. In CEUR Workshop Proceedings (Vol. 1064).

Hellmann, S., Stadler, C., Lehmann, J., & Auer, S. (2009). DBpedia live extraction. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 5871 LNCS, pp. 1209–1223).

Hu, Z., Huang, P., Deng, Y., Gao, Y., Xing, E. (2015). Entity Hierarchy Embedding. In Proceedings of the Association for Computational Linguistics 2015 (ACL 2015), pp.1292–1300.

Jain, P., Hitzler, P., Sheth, A. A. P., Verma, K., & Yeh, P. P. Z. (2010). Ontology alignment for linked open data. Iswc, 402–417.

Jain, P., Hitzler, P., Verma, K., Yeh, P. Z., & Sheth, A. P. (2012). Moving Beyond SameAs with PLATO: Partonomy Detection for Linked Data. In Proceedings of the 23rd ACM Conference on Hypertext and Social Media (pp. 33–42).

Jurafsky, D., & Martin, J. H. (2007). Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. *Computational Linguistics*, *26*(4), 638–641.

Kabir, S., Ripon, S., Rahman, M., & Rahman, T. (2014). Knowledge-based Data Mining Using Semantic Web. *IERI Procedia*, *7*, 113–119.

Keong, B. V., Anthony, P. (2011). Meta search engine powered by DBpedia. In Proceedings of the 2011 International Conference on Semantic Technology and Information Retrieval, STAIR 2011 (pp. 89–93).

Kliegr, T. (2015). Linked hypernyms: Enriching DBpedia with Targeted Hypernym Discovery. Journal of Web Semantics, 31, 59–69.

Kliegr, T., & Zamazal, O. (2014). Towards Linked Hypernyms Dataset 2.0: complementing DBpedia with hypernym discovery. Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014), 3517–3523.

König, M., Dirnbek, J., & Stankovski, V. (2013). Architecture of an open knowledge base for sustainable buildings based on Linked Data technologies. *Automation in Construction*, *35*, 542–550.

Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., Zaveri, A. (2014). Test-driven evaluation of linked data quality. In Proceedings of the 23rd international conference on World Wide Web - WWW '14 (pp. 747–758).

Kornbrot, D. (2005). Pearson Product Moment Correlation. In Encyclopedia of Statistics in Behavioral Science.

Krishna, K., & Murty, M. N. (1999). Genetic K-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *29*(3), 433–439.

Krötzsch, M., Vrandečić, D., Völkel, M., Haller, H., Studer, R. (2007). Semantic Wikipedia. Web Semantics, 5(4), 251–261.

Lehmann, J., & Bühmann, L. (2010). ORE - A tool for repairing and enriching knowledge bases. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 6497 LNCS, pp. 177–193).

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Bizer, C. (2015). DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web, 6(2), 167–195.

Li, W., Raskin, R., & Goodchild, M. F. (2012). Semantic similarity measurement based on knowledge mining: An artificial neural net approach. *International Journal of Geographical Information Science*, *26*(8), 1415–1435.

Ling, C. X., Huang, J., & Zhang, H. (2003). AUC: A better measure than accuracy in comparing learning algorithms. Advances in Artificial Intelligence Lecture Notes in Computer Science (Vol. 2671, pp. 329–341).

Maedche, A., & Staab, S. (2001). Ontology Learning for the Semantic Web. IEEE Intelligent Systems, 16(2), 72–79.

Mayfield, J., & Finin, T. (2012). Evaluating the Quality of a Knowledge Base Populated from Text. Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-Scale Knowledge Extraction (AKBC-WEKEX 2012), 68–73.

Mendes, P. N., Jakob, M., & Bizer, C. (2012). DBpedia: A Multilingual Cross-Domain Knowledge Base. Language Resources and Evaluation LRES, 1813–1817.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013c). Distributed Representations of Words and Phrases and their Compositionality. *Nips*, 1–9.

Mikolov, T., Corrado, G., Chen, K., & Dean, J. (2013b). Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, 1–12.

Mikolov, T., Yih, W., & Zweig, G. (2013a). Linguistic regularities in continuous space word representations. *Proceedings of NAACL-HLT*, (June), 746–751.

Mitchell, T. M. (1997). Machine Learning. McGraw Hill.

Moran, S. (2012). Using Linked Data to Create a Typological Knowledge Base. In *Linked Data in Linguistics: Representing and Connecting Language Data and Language Metadata* (pp. 129–138).

Morsey, M., Lehmann, J., Auer, S., Stadler, C., Hellmann, S. (2012). DBpedia and the live extraction of structured data from Wikipedia. Program, 46(2), 157–181.

Pan, X. Y., & Shen, H. B. (2009). Robust prediction of B-factor profile from sequence using two-stage SVR based on random forest feature selection. *Protein Pept Lett*, *16*(12), 1447–1454.

Paulheim, H. (2017). Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, *8*(3), 489–508.

Paulheim, H., & Bizer, C. (2013). Type inference on noisy RDF data. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 8218 LNCS, pp. 510–525).

Paulheim, H., Bizer, C. (2014). Improving the Quality of Linked Data Using Statistical Distributions. International Journal on Semantic Web and Information Systems (IJSWIS), 10(June), 63–86.

Paulien Meijer, Verloop, N., & Driel, J. Van. (2001). Teacher knowledge and the knowledge base of teaching. International Journal of Educational Research, 35(5), 441–461.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Duborg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Journal of Machine Learning Research, 12, 2825–2830.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1532–1543).

Roark, B., Saraclar, M., & Collins, M. (2007). Discriminative n-gram language modeling. Computer Speech & Language, 21(2 (June 2006April 2007)), (pp. 373–392)1–30.

Roder, M., Usbeck, R., Speck, R., & Ngonga Ngomo, A. C. (2015). CETUS - a baseline approach to type extraction. In *Communications in Computer and Information Science* (Vol. 548, pp. 16–27).

Rodriguez, M. A. (2009). A Graph Analysis of the Linked Data Cloud. Statistics, (March), 1–7.

Rogers, J., & Gunn, S. (2006). Identifying feature relevance using a random forest. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 3940 LNCS, pp. 173–184).

Seok, M., Song, H.-J., Park, C.-Y., Kim, J.-D., Kim, Y.-S. (2016). Named Entity Recognition using Word Embedding as a Feature 1. International Journal of Software Engineering and Its Applications, 10(2), 93–104.

Shadbolt, N., Hall, W., & Berners-Lee, T. (2006). The semantic web revisited. IEEE Intelligent Systems.

Shen, W., Wang, J., Luo, P., & Wang, M. (2012). LINDEN: linking named entities with knowledge base via semantic knowledge. *Www '12*, 449.

Sheng, Z., Wang, X., Shi, H., & Feng, Z. (2012). Checking and handling inconsistency of DBpedia. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 7529 LNCS, pp. 480–488).

Simmie, J., & Lever, W. F. (2002). Introduction: The Knowledge-based City. *Urban Studies*, *39*(6), 855–857.

Staab, S., & Studer, R. (2009). Handbook on Ontologies - Second edition. International Handbooks on Information Systems, 811.

Suchanek, F. M., Kasneci, G., & Weikum, G. (2007). YAGO: a core of semantic knowledge. Proceedings of the 16th International Conference on World Wide Web, 697–706.

Suchanek, F. M., Kasneci, G., & Weikum, G. (2008). YAGO: A Large Ontology from Wikipedia and WordNet. *Web Semantics*, *6*(3), 203–217.

Syed, Z., & Finin, T. (2011). Creating and Exploiting a Hybrid Knowledge Base for Linked Data. In *Communications in Computer and Information Science* (Vol. 129, pp. 3–21).

Tanon, T. P., Vrandečić, D., Francisco, S., Schaffert, S., & Steiner, T. (2016). From Freebase to Wikidata : The Great Migration. Proceedings of the 25th International Conference on World Wide Web, 1419–1428.

Töpper, G., Knuth, M., & Sack, H. (2012). DBpedia ontology enrichment for inconsistency detection. *Proceedings of the 8th International Conference on Semantic Systems - I-SEMANTICS '12*, 33.

Van Erp M., Vossen P. (2017). Entity Typing Using Distributional Semantics and DBpedia. In: van Erp M. et al. (eds) Knowledge Graphs and Language Technology. ISWC 2016. Lecture Notes in Computer Science, vol 10579. Springer, Cham.

Van Hooland, S., Verborgh, R., de Walle, R., & Van de Walle, R. (2012). Joining the Linked Data Cloud in a Cost-Effective Manner. Information Standards Quarterly, 22(2), 24–28.

Verloop, N., Van Driel, J., & Meijer, P. (2001). Teacher knowledge and the knowledge base of teaching. *International Journal of Educational Research*, *35*(5), 441–461.

Vrandečić, D., & Krötzsch, M. (2014). Wikidata: A Free Collaborative Knowledgebase. *Communications of the ACM*, *57*(10), 78–85. https://doi.org/10.1145/2629489

Webb, A. (2002). Linear discriminant analysis. Statistical Pattern Recognition, 123–168.

Wienand, D., & Paulheim, H. (2014). Detecting incorrect numerical data in DBpedia. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 8465 LNCS, pp. 504–518).

Yao, X., & Van Durme, B. (2014). Information Extraction over Structured Data: Question Answering with Freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 956–966).

Yih, W.-T., Richardson, M., Meek, C., Chang, M.-W., & Suh, J. (2016). The Value of Semantic Parse Labeling for Knowledge Base Question Answering. *ACL (Short Paper)*, 201–206.

Zaveri, A., Kontokostas, D., Sherif, M. A., Bühmann, L., Morsey, M., Auer, S., Lehmann, J. (2013). User-driven quality evaluation of DBpedia. In Proceedings of the 9th International Conference on Semantic Systems - I-SEMANTICS '13 (p. 97).

Zhang, F., Yuan, N. J., Lian, D., Xie, X., & Ma, W.-Y. (2016). Collaborative Knowledge Base Embedding for Recommender Systems. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 353–362.

Zhang, H. (2004). The Optimality of Naive Bayes. Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference FLAIRS 2004, 1(2), 1–6.

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: An Efficient Data Clustering Method for Very Large Databases. Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, 1, 103–114.

Zhang, Z., Chen, S., & Feng, Z. (2013). Semantic Annotation for Web Services Based on DBpedia. In Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on (pp. 280–285).

Zheng, Z., Li, F., Huang, M., & Zhu, X. (2010). Learning to link entities with knowledge base. *Hlt 2010*, (June), 483–491.

Zhou, G., He, T., Zhao, J., Hu, P. (2015). Learning Continuous Word Embedding with Metadata for Question Retrieval in Community Question Answering. In Proceedings of ACL.

Zhou, H., Zouaq, A., Inkpen, D. (2017). DBpedia Entity Type Detection using Entity Embeddings and N-Gram Models. In Proceedings of the International Conference on Knowledge Engineering and Semantic Web (KESW 2017), Szczecin, Poland.

Appendix A

Appendix A shows more detailed results on specific types and entities.

The tables A1 and A2 show the detailed results of 10 typical DBpedia types with K-means clustering algorithm with skip-gram and n-gram models based on the test dataset.

Type	Precision	Recall	F-Score	Accuracy
Aircraft	1.000	1.000	1.000	100%
Book	0.857	1.000	0.923	91.7%
Cartoon	1.000	0.667	0.800	83.3%
Device	0.962	1.000	0.980	98.0%
Economist	1.000	1.000	1.000	100%
Film	1.000	1.000	1.000	100%
Game	0.800	0.762	0.781	78.6%
Holiday	0.556	1.000	0.714	60.0%
Island	0.875	1.000	0.933	92.9%
Language	1.000	0.600	0.750	80.0%

Table A. 1. Detailed K-means clustering results on ten typical DBpedia types with skip-gram model on test dataset

Type	Precision	Recall	F-Score	Accuracy
Aircraft	0.546	1.000	0.706	58.3%
Book	0.455	0.833	0.588	41.7%
Cartoon	0.517	1.000	0.682	53.3%
Device	0.510	1.000	0.676	52.0%
Economist	0.444	0.800	0.571	40.0%
Film	0.478	0.917	0.629	45.8%
Game	0.512	1.000	0.677	52.4%
Holiday	0.444	0.800	0.571	40.0%
Island	0.539	1.000	0.700	57.1%
Language	0.474	0.900	0.621	45.0%

Table A. 2. Detailed K-means clustering results on ten typical DBpedia types with n-gram model on test dataset

The tables A3 and A4 show the detailed results of 10 typical DBpedia types with SVM classification algorithm with skip-gram and n-gram models based on the test dataset.

Type	Precision	Recall	F-Score	Accuracy	AUC
Aircraft	0.857	1.000	0.923	91.7%	1.000
Book	1.000	1.000	1.000	100%	1.000
Cartoon	1.000	1.000	1.000	100%	1.000
Device	1.000	1.000	1.000	100%	1.000
Economist	1.000	1.000	1.000	100%	1.000
Film	1.000	1.000	1.000	100%	1.000
Game	0.889	0.762	0.821	83.3%	0.934
Holiday	1.000	1.000	1.000	100%	1.000
Island	1.000	1.000	1.000	100%	1.000
Language	1.000	0.900	0.947	95.0%	1.000

Table A. 3. Detailed SVM Classification results on ten typical DBpedia types with skip-gram model on test dataset

Type	Precision	Recall	F-Score	Accuracy	AUC
Aircraft	1.000	0.833	0.909	91.7%	1.000
Book	1.000	1.000	1.000	100%	1.000
Cartoon	1.000	0.933	0.966	96.7%	1.000
Device	1.000	0.920	0.958	96.0%	0.998
Economist	1.000	0.800	0.889	90.0%	1.000
Film	0.900	0.750	0.818	83.3%	0.979
Game	1.000	0.619	0.765	81.0%	0.918
Holiday	1.000	1.000	1.000	100%	1.000
Island	1.000	1.000	1.000	100%	1.000
Language	1.000	1.000	1.000	100%	1.000

Table A. 4. Detailed SVM Classification results on ten typical DBpedia types with n-gram model on test dataset

Appendix B

Appendix B shows the details of the 760 DBpedia Types (extracted from the DBpedia Ontology), whether they have any related entities and whether they have been used in the Training and test datasets

DBpedia Ontology (Type)	Any related entity	Used in dataset
Abbey	No	No
AcademicConference	Yes	No
AcademicJournal	Yes	Yes
AcademicSubject	No	No
Activity	Yes	Yes
Actor	Yes	Yes
AdministrativeRegion	Yes	Yes
AdultActor	Yes	Yes
Agent	Yes	Yes
Agglomeration	No	No
Aircraft	Yes	Yes
Airline	Yes	Yes
Airport	Yes	Yes
Album	Yes	Yes
Altitude	No	No
AmateurBoxer	Yes	Yes
Ambassador	Yes	Yes
AmericanFoorballPlayer	Yes	Yes
AmericanFoorballTeam	Yes	Yes
AmericanFootballCoach	No	No
AmericanFootballLeague	Yes	Yes
Amphibian	Yes	Yes
AmusementParkAttraction	Yes	Yes

AnatomicalStructure	Yes	Yes
Animal	Yes	Yes
AnimangaCharacter	Yes	Yes
Anime	Yes	Yes
Annotation	No	No
Arachnid	Yes	No
Archaea	Yes	No
Archeologist	No	No
ArcherPlayer	No	No
Archipelago	No	No
Architect	Yes	Yes
ArchitecturalStructure	Yes	Yes
Archive	No	No
Area	No	No
Arena	No	No
Aristocrat	No	No
Arrondissement	No	No
Artery	Yes	No
Article	No	No
ArtificialSatellite	Yes	Yes
Artist	Yes	Yes
ArtistDiscography	Yes	Yes
ArtisticGenre	No	No
Artwork	Yes	Yes
Asteroid	Yes	No
Astronaut	Yes	No
Athlete	Yes	Yes
Athletics	No	No
AthleticsPlayer	No	No
Atoll	No	No

Attack	No	No
AustralianFootballLeague	Yes	No
AustralianFootballTeam	Yes	Yes
AustralianRulesFootballPlayer	Yes	Yes
Automobile	Yes	Yes
AutomobileEngine	Yes	No
AutoRacingLeague	Yes	No
Award	Yes	Yes
BackScene	No	No
Bacteria	Yes	Yes
BadmintonPlayer	Yes	Yes
Band	Yes	Yes
Bank	Yes	Yes
Baronet	Yes	No
baseballLeague	Yes	Yes
baseballPlayer	Yes	Yes
BaseballSeason	Yes	No
BaseballTeam	Yes	Yes
BasketballLeague	Yes	Yes
BasketballPlayer	Yes	Yes
BasketballTeam	Yes	Yes
Bay	Yes	No
Beach	No	No
BeachVolleyballPlayer	Yes	No
BeautyQueen	Yes	Yes
Beer	No	No
Beverage	Yes	Yes
Biathlete	No	No
BiologicalDatabase	Yes	No
Biologist	No	No

Biomolecule	Yes	Yes
Bird	Yes	Yes
Birth	No	No
Blazon	No	No
BloodVessel	No	No
BoardGame	No	No
BobsleighAthlete	No	No
Bodybuilder	Yes	Yes
BodyOfWater	Yes	Yes
Bone	Yes	Yes
Book	Yes	Yes
BowlingLeague	No	No
Boxer	Yes	Yes
Boxing	No	No
BoxingCategory	No	No
BoxingLeague	No	No
BoxingStyle	No	No
Brain	Yes	Yes
Brewery	Yes	No
Bridge	Yes	Yes
BritishRoyalty	Yes	No
Broadcaster	Yes	Yes
BroadcastNetwork	Yes	Yes
BrownDwarf	No	No
Building	Yes	Yes
BullFighter	No	No
BusCompany	Yes	Yes
BusinessPerson	Yes	No
Camera	No	No
CanadianFootballLeague	Yes	No

CanadianFootballPlayer	No	No
CanadianFootballTeam	Yes	Yes
Canal	Yes	Yes
Canoeist	Yes	No
Canton	No	No
Capital	No	No
CapitalOfRegion	No	No
CardGame	No	No
Cardinal	Yes	Yes
CardinalDirection	No	No
CareerStation	Yes	No
Cartoon	Yes	Yes
Case	Yes	Yes
Casino	No	No
Castle	Yes	Yes
Cat	No	No
Caterer	No	No
Cave	Yes	Yes
Celebrity	No	No
CelestialBody	Yes	Yes
Cemetery	No	No
Chancellor	Yes	Yes
ChartsPlacements	No	No
Cheese	Yes	No
Chef	Yes	Yes
ChemicalCompound	Yes	Yes
ChemicalElement	No	No
ChemicalSubstance	Yes	Yes
ChessPlayer	Yes	Yes
ChristianBishop	Yes	Yes

ChristianDoctrine	No	No
ChristianPatriarch	No	No
Church	No	No
City	Yes	Yes
CityDistrict	No	No
ClassicalMusicArtist	Yes	No
ClassicalMusicComposition	Yes	No
Cleric	Yes	Yes
ClericalAdministrativeRegion	Yes	Yes
ClericalOrder	No	No
ClubMoss	Yes	No
Coach	Yes	Yes
CoalPit	No	No
CollectionOfValuables	No	No
College	Yes	Yes
CollegeCoach	Yes	Yes
Colour	Yes	Yes
Comedian	Yes	Yes
ComedyGroup	Yes	Yes
Comic	Yes	Yes
ComicsCharacter	Yes	Yes
ComicsCreator	Yes	Yes
ComicStrip	Yes	No
Community	No	No
Company	Yes	Yes
Competition	No	No
ConcentrationCamp	No	No
Congressman	Yes	Yes
Conifer	Yes	Yes
Constellation	Yes	Yes

Contest	No	No
Continent	Yes	No
ControlledDesignationOfOriginWine	No	No
Convention	Yes	Yes
ConveyorSystem	No	No
Country	Yes	Yes
CountrySeat	No	No
Creek	No	No
Cricketer	Yes	Yes
CricketGround	Yes	No
CricketLeague	Yes	No
CricketTeam	Yes	Yes
Criminal	Yes	Yes
CrossCountrySkier	No	No
Crustacean	Yes	Yes
CultivatedVariety	Yes	Yes
Curler	Yes	Yes
CurlingLeague	Yes	No
Currency	Yes	Yes
Cycad	Yes	No
CyclingCompetition	No	No
CyclingLeague	No	No
CyclingRace	Yes	Yes
CyclingTeam	Yes	Yes
Cyclist	Yes	Yes
Dam	Yes	Yes
Dancer	No	No
DartsPlayer	Yes	Yes
Database	Yes	No
Deanery	No	No

Death	No	No
Decoration	No	No
Deity	No	No
Demographics	No	No
Department	No	No
Depth	No	No
Deputy	No	No
Desert	No	No
Device	Yes	Yes
DigitalCamera	No	No
Dike	No	No
Diocese	Yes	Yes
Diploma	No	No
Disease	Yes	Yes
DisneyCharacter	No	No
District	No	No
DistrictWaterBoard	No	No
Divorce	No	No
Document	Yes	No
DocumentType	No	No
Dog	No	No
Drama	No	No
Drug	Yes	Yes
DTMRacer	No	No
Earthquake	Yes	Yes
Economist	Yes	Yes
EducationalInstitution	Yes	Yes
Egyptologist	No	No
Election	Yes	Yes
ElectionDiagram	No	No

ElectricalSubstation	No	No
Embryology	Yes	No
Employer	No	No
EmployersOrganisation	No	No
Engine	Yes	No
Engineer	Yes	Yes
Entomologist	Yes	Yes
Enzyme	Yes	Yes
Escalator	No	No
EthnicGroup	Yes	Yes
Eukaryote	Yes	Yes
EurovisionSongContestEntry	Yes	No
Event	Yes	Yes
Factory	No	No
Family	No	No
Farmer	No	No
Fashion	Yes	No
FashionDesigner	Yes	Yes
Fencer	No	No
Fern	Yes	Yes
FictionalCharacter	Yes	Yes
Fiefdom	No	No
FieldHockeyLeague	Yes	No
FigureSkater	Yes	Yes
File	No	No
FillingStation	No	No
Film	Yes	Yes
FilmFestival	Yes	Yes
Fish	Yes	Yes
Flag	No	No

FloweringPlant	Yes	Yes
Food	Yes	Yes
FootballLeagueSeason	Yes	Yes
FootballMatch	Yes	Yes
FormerMunicipality	No	No
FormulaOneRacer	Yes	Yes
FormulaOneRacing	No	No
FormulaOneTeam	Yes	Yes
Fungus	Yes	Yes
GaelicGamesPlayer	Yes	Yes
Galaxy	Yes	No
Game	Yes	Yes
Garden	Yes	No
Gate	No	No
GatedCommunity	No	No
Gene	Yes	No
GeneLocation	Yes	No
Genre	Yes	Yes
GeologicalPeriod	No	No
GeopoliticalOrganisation	No	No
Ginkgo	Yes	No
GivenName	Yes	Yes
Glacier	Yes	Yes
Globularswarm	No	No
Gnetophytes	Yes	No
GolfCourse	Yes	No
GolfLeague	Yes	No
GolfPlayer	Yes	Yes
GolfTournament	Yes	Yes
GovernmentAgency	Yes	Yes

GovernmentalAdministrativeRegion	No	No
GovernmentCabinet	No	No
GovernmentType	No	No
Governor	Yes	Yes
GrandPrix	Yes	Yes
Grape	Yes	Yes
GraveMonument	No	No
GreenAlga	Yes	No
GridironFootballPlayer	Yes	Yes
GrossDomesticProduct	No	No
GrossDomesticProductPerCapita	No	No
Group	Yes	Yes
Guitar	No	No
Guitarist	Yes	Yes
Gymnast	Yes	Yes
HandballLeague	Yes	No
HandballPlayer	Yes	No
HandballTeam	Yes	Yes
HighDiver	No	No
Historian	Yes	Yes
HistoricalAreaOfAuthority	No	No
HistoricalCountry	No	No
HistoricalDistrict	No	No
HistoricalPeriod	No	No
HistoricalProvince	No	No
HistoricalRegion	No	No
HistoricalSettlement	No	No
HistoricBuilding	Yes	Yes
HistoricPlace	Yes	Yes
HockeyClub	No	No

HockeyTeam	Yes	Yes
Holiday	Yes	Yes
HollywoodCartoon	Yes	Yes
Horse	Yes	Yes
HorseRace	Yes	Yes
HorseRider	Yes	No
HorseRiding	No	No
HorseTrainer	Yes	Yes
Hospital	Yes	Yes
Host	No	No
Hotel	Yes	Yes
HotSpring	No	No
HumanDevelopmentIndex	No	No
HumanGene	Yes	No
HumanGeneLocation	Yes	No
Humorist	No	No
IceHockeyLeague	Yes	Yes
IceHockeyPlayer	Yes	Yes
Ideology	No	No
Image	No	No
InformationAppliance	Yes	Yes
Infrastructure	Yes	Yes
InlineHockeyLeague	Yes	No
Insect	Yes	Yes
Instrument	No	No
Instrumentalist	Yes	Yes
Intercommunality	No	No
InternationalFootballLeagueEvent	No	No
InternationalOrganisation	No	No
Island	Yes	Yes

Jockey	Yes	Yes
Journalist	Yes	Yes
Judge	Yes	Yes
LacrosseLeague	Yes	No
LacrossePlayer	Yes	No
Lake	Yes	Yes
Language	Yes	Yes
LaunchPad	Yes	Yes
Law	No	No
LawFirm	Yes	Yes
Lawyer	No	No
LegalCase	Yes	Yes
Legislature	Yes	Yes
Letter	No	No
Library	Yes	Yes
Lieutenant	No	No
LifeCycleEvent	No	No
Ligament	Yes	No
Lighthouse	Yes	Yes
LightNovel	No	No
LineOfFashion	No	No
Linguist	No	No
Lipid	No	No
List	No	No
LiteraryGenre	No	No
Locality	No	No
Lock	No	No
Locomotive	Yes	Yes
LunarCrater	No	No
Lymph	Yes	No

Magazine	Yes	Yes
Mammal	Yes	Yes
Manga	Yes	Yes
Manhua	No	No
Manhwa	No	No
Marriage	No	No
MartialArtist	Yes	Yes
MathematicalConcept	No	No
Mayor	Yes	Yes
MeanOfTransportation	Yes	Yes
Media	No	No
Medician	Yes	Yes
Medicine	No	No
Meeting	No	No
MemberOfParliament	Yes	Yes
MemberResistanceMovement	No	No
Memorial	No	No
MetroStation	No	No
MicroRegion	No	No
MilitaryAircraft	No	No
MilitaryConflict	Yes	Yes
MilitaryPerson	Yes	Yes
MilitaryStructure	Yes	Yes
MilitaryUnit	Yes	Yes
MilitaryVehicle	No	No
Mill	No	No
Mine	No	No
Mineral	Yes	No
MixedMartialArtsEvent	Yes	Yes
MixedMartialArtsLeague	No	No

MobilePhone	No	No
Model	Yes	Yes
Mollusca	Yes	No
Monarch	Yes	Yes
Monastery	No	No
Monument	Yes	Yes
Mosque	No	No
Moss	Yes	No
MotocycleRacer	No	No
Motorcycle	Yes	Yes
MotorcycleRacingLeague	Yes	No
MotorcycleRider	Yes	Yes
MotorRace	No	No
MotorsportRacer	Yes	Yes
MotorsportSeason	Yes	No
Mountain	Yes	Yes
MountainPass	Yes	Yes
MountainRange	Yes	Yes
MouseGene	Yes	No
MouseGeneLocation	Yes	No
MovieDirector	No	No
MovieGenre	No	No
MovingImage	No	No
MovingWalkway	No	No
MultiVolumePublication	No	No
Municipality	No	No
Murderer	Yes	Yes
Muscle	Yes	Yes
Museum	Yes	Yes
Musical	Yes	Yes

MusicalArtist	Yes	Yes
MusicalWork	Yes	Yes
MusicComposer	No	No
MusicDirector	No	No
MusicFestival	Yes	Yes
MusicGenre	Yes	Yes
MythologicalFigure	Yes	No
Name	Yes	Yes
NarutoCharacter	No	No
NascarDriver	Yes	Yes
NationalAnthem	No	No
NationalCollegiateAthleticAssociationAthlete	Yes	No
NationalFootballLeagueEvent	Yes	No
NationalFootballLeagueSeason	Yes	Yes
NationalSoccerClub	No	No
NaturalEvent	Yes	Yes
NaturalPlace	Yes	Yes
NaturalRegion	No	No
NCAATeamSeason	Yes	Yes
Nerve	Yes	Yes
NetballPlayer	Yes	No
Newspaper	Yes	Yes
NobelPrize	No	No
Noble	Yes	Yes
NobleFamily	No	No
Non-ProfitOrganisation	Yes	Yes
NordicCombined	No	No
Novel	Yes	No
NuclearPowerStation	No	No
Ocean	No	No

OfficeHolder	Yes	Yes
OldTerritory	No	No
OlympicEvent	Yes	Yes
OlympicResult	Yes	Yes
Olympics	Yes	Yes
On-SiteTransportation	No	No
Openswarm	No	No
Opera	No	No
Organ	No	No
Organisation	Yes	Yes
OrganisationMember	Yes	No
Orphan	No	No
OverseasDepartment	No	No
PaintballLeague	No	No
Painter	Yes	Yes
Painting	No	No
Parish	No	No
Park	Yes	Yes
Parliament	No	No
PenaltyShootOut	No	No
PeriodicalLiterature	Yes	Yes
PeriodOfArtisticStyle	No	No
Person	Yes	Yes
PersonalEvent	No	No
PersonFunction	Yes	No
Philosopher	Yes	Yes
PhilosophicalConcept	No	No
Photographer	Yes	Yes
Place	Yes	Yes
Planet	Yes	Yes

Plant	Yes	Yes
Play	Yes	Yes
PlayboyPlaymate	Yes	Yes
PlayWright	No	No
Poem	Yes	No
Poet	Yes	Yes
PokerPlayer	Yes	Yes
PoliticalConcept	No	No
PoliticalFunction	No	No
PoliticalParty	Yes	Yes
Politician	Yes	Yes
PoliticianSpouse	No	No
PoloLeague	Yes	No
Polyhedron	No	No
Polysaccharide	No	No
Pope	Yes	Yes
PopulatedPlace	Yes	Yes
Population	No	No
Port	No	No
PowerStation	Yes	Yes
Prefecture	No	No
PrehistoricalPeriod	No	No
Presenter	Yes	Yes
President	Yes	Yes
Priest	No	No
PrimeMinister	Yes	Yes
Prison	Yes	Yes
Producer	No	No
Profession	No	No
Professor	No	No

ProgrammingLanguage	Yes	Yes
Project	Yes	No
ProtectedArea	Yes	Yes
Protein	Yes	Yes
ProtohistoricalPeriod	No	No
Province	No	No
Psychologist	No	No
PublicService	No	No
PublicTransitSystem	Yes	Yes
Publisher	Yes	Yes
Pyramid	No	No
Quote	No	No
Race	Yes	Yes
Racecourse	Yes	Yes
RaceHorse	Yes	Yes
RaceTrack	Yes	Yes
RacingDriver	Yes	Yes
RadioControlledRacingLeague	No	No
RadioHost	Yes	Yes
RadioProgram	Yes	Yes
RadioStation	Yes	Yes
RailwayLine	Yes	Yes
RailwayStation	Yes	No
RailwayTunnel	Yes	Yes
RallyDriver	No	No
Rebellion	No	No
RecordLabel	Yes	Yes
RecordOffice	No	No
Referee	No	No
Reference	No	No

Regency	No	No
Region	Yes	Yes
Religious	Yes	Yes
ReligiousBuilding	Yes	Yes
ReligiousOrganisation	No	No
Reptile	Yes	Yes
ResearchProject	Yes	No
RestArea	No	No
Restaurant	Yes	Yes
Resume	No	No
River	Yes	Yes
Road	Yes	Yes
RoadJunction	Yes	Yes
RoadTunnel	Yes	Yes
Rocket	Yes	Yes
RocketEngine	No	No
RollerCoaster	Yes	Yes
RomanEmperor	No	No
RouteOfTransportation	Yes	Yes
RouteStop	No	No
Rower	Yes	No
Royalty	Yes	Yes
RugbyClub	Yes	Yes
RugbyLeague	Yes	Yes
RugbyPlayer	Yes	Yes
Saint	Yes	Yes
Sales	Yes	No
SambaSchool	No	No
Satellite	Yes	Yes
School	Yes	Yes

ScientificConcept	No	No
Scientist	Yes	Yes
ScreenWriter	Yes	Yes
Sculptor	No	No
Sculpture	No	No
Sea	Yes	Yes
Senator	Yes	Yes
SerialKiller	No	No
Settlement	Yes	Yes
Ship	Yes	Yes
ShoppingMall	Yes	Yes
Shrine	No	No
Singer	No	No
Single	Yes	Yes
SiteOfSpecialScientificInterest	Yes	Yes
Skater	Yes	Yes
Ski_jumper	No	No
SkiArea	Yes	Yes
Skier	Yes	Yes
SkiResort	No	No
Skyscraper	Yes	No
SnookerChamp	Yes	Yes
SnookerPlayer	Yes	Yes
SnookerWorldRanking	Yes	No
SoapCharacter	Yes	Yes
Soccer	No	No
SoccerClub	Yes	Yes
SoccerClubSeason	Yes	No
SoccerLeague	Yes	Yes
SoccerLeagueSeason	No	No

SoccerManager	Yes	Yes
SoccerPlayer	Yes	Yes
SoccerTournament	Yes	Yes
SocietalEvent	Yes	Yes
SoftballLeague	Yes	No
Software	Yes	Yes
SolarEclipse	Yes	No
Song	Yes	Yes
SongWriter	No	No
Sound	Yes	No
Spacecraft	Yes	Yes
SpaceMission	No	No
SpaceShuttle	Yes	No
SpaceStation	Yes	No
Species	Yes	Yes
SpeedSkater	No	No
SpeedwayLeague	Yes	No
SpeedwayRider	Yes	Yes
SpeedwayTeam	Yes	Yes
Sport	Yes	Yes
SportCompetitionResult	Yes	Yes
SportFacility	Yes	Yes
SportsClub	No	No
SportsEvent	Yes	Yes
SportsLeague	Yes	Yes
SportsManager	Yes	Yes
SportsSeason	Yes	Yes
SportsTeam	Yes	Yes
SportsTeamMember	Yes	No
SportsTeamSeason	Yes	Yes

Square	No	No
SquashPlayer	Yes	Yes
Stadium	Yes	Yes
Standard	No	No
Star	Yes	Yes
State	No	No
StatedResolution	No	No
Station	Yes	Yes
Statistic	No	No
StillImage	No	No
StormSurge	No	No
Stream	Yes	Yes
Street	No	No
SubMunicipality	No	No
SumoWrestler	Yes	No
SupremeCourtOfTheUnitedStatesCase	Yes	Yes
Surfer	No	No
Surname	Yes	No
Swarm	No	No
Swimmer	Yes	Yes
Synagogue	No	No
SystemOfLaw	No	No
TableTennisPlayer	Yes	Yes
Tax	No	No
Taxon	No	No
TeamMember	No	No
TeamSport	No	No
TelevisionDirector	No	No
TelevisionEpisode	Yes	Yes
TelevisionHost	Yes	No

TelevisionPersonality	No	No
TelevisionSeason	Yes	Yes
TelevisionShow	Yes	Yes
TelevisionStation	Yes	Yes
Temple	No	No
TennisLeague	Yes	No
TennisPlayer	Yes	Yes
TennisTournament	Yes	Yes
TermOfOffice	No	No
Territory	No	No
Theatre	Yes	Yes
TheatreDirector	No	No
TheologicalConcept	No	No
TimePeriod	Yes	No
TopicalConcept	Yes	Yes
Tournament	Yes	Yes
Tower	Yes	Yes
Town	Yes	Yes
TrackList	No	No
TradeUnion	Yes	Yes
Train	Yes	Yes
TrainCarriage	No	No
Tram	No	No
TramStation	No	No
Treadmill	No	No
Treaty	No	No
Tunnel	Yes	No
Type	No	No
UndergroundJournal	No	No
UnitOfWork	Yes	Yes

University	Yes	Yes
Unknown	No	No
Valley	Yes	Yes
Vein	Yes	No
Venue	Yes	Yes
Vicar	No	No
VicePresident	No	No
VicePrimeMinister	No	No
VideoGame	Yes	Yes
VideogamesLeague	Yes	No
Village	Yes	Yes
Vodka	No	No
VoiceActor	Yes	Yes
Volcano	Yes	Yes
VolleyballCoach	Yes	No
VolleyballLeague	Yes	Yes
VolleyballPlayer	Yes	Yes
Watermill	No	No
WaterPoloPlayer	No	No
WaterRide	Yes	No
WaterTower	No	No
WaterwayTunnel	Yes	No
Weapon	Yes	Yes
Website	Yes	Yes
Windmill	No	No
WindMotor	No	No
Wine	No	No
WineRegion	Yes	Yes
Winery	Yes	Yes
WinterSportPlayer	Yes	Yes

WomensTennisAssociationTournament	Yes	Yes
Work	Yes	Yes
WorldHeritageSite	Yes	Yes
Wrestler	Yes	Yes
WrestlingEvent	Yes	Yes
Writer	Yes	Yes
WrittenWork	Yes	Yes
Year	Yes	No
YearInSpaceflight	Yes	Yes
Zoo	No	No

Table B. 1. Details of DBpedia Ontology

Appendix C

This section shows some sample DBpedia SPARQL queries

The following sample DBpedia SPARQL query use `select` to extract the capital of Canada by using the triple: `<Subject Predicate Object>`. In this case, the `Subject` is `http://dbpedia.org/resource/Canada`, and `Predicate` is `dbo:capital`. The results stored in DBpedia is Ottawa.

```
select distinct ?Captical where
{
<http://dbpedia.org/resource/Canada> dbo:capital ?Capital
}
```

We can extract other properties of <`http://dbpedia.org/resource/Canada`> by changing the `Predicate`. For example, if we to extract the largest of Canada, we can change the `Predicate` to `dbo:largestCity` as shown below.

```
select distinct ?Largestcity where
{
<http://dbpedia.org/resource/Canada> dbo:largestCity ?LargestCiry
}
```

Other examples such as total area and total population of Canada are shown below.

Total Area of Canada:

```
select distinct ?Area where
{
<http://dbpedia.org/resource/Canada> dbo: areaTotal ?LargestCiry
}
```

Total Population of Canada:

```
select distinct ?Population where
{
<http://dbpedia.org/resource/Canada> dbo: populationTotal ?Population
}
```

If we know a country's capital is Ottawa, we can extract the country's name as well, in the tuple `<Subject Predicate Object>`, if we know the `Predicate` and `Object`, we can extract `Subject` as the following example shows.

```
select distinct ?Country where
{
?Country dbo:capital <http://dbpedia.org/resource/Ottawa>
}
```

The following DBpedia SPARQL uses ask to get result whether the Canada's Total population is greater than 36,000,000 or not. The result will be in the format true/false result.

```
prefix xsdt: <http://www.w3.org/2001/XMLSchema#>
ask where
{
<http://dbpedia.org/resource/Canada> dbo:populationTotal ?TotalPopulation.
filter(?TotalPopulation >"36000000"^^xsdt:integer )
}
```

The following DBpedia SPARQL uses filter and in extract the area and population of Canada at the same time.

```
SELECT ?Area ?Population
WHERE
{
 <http://dbpedia.org/resource/Canada> ?Area ?Population
```

```
filter(?Area in(dbo:areaTotal,dbo:populationTotal))
}
```

The following DBpedia query uses filter to filter more condition(s). In the query, it first as the name that stored in DBpedia, then it filers the condition the `birthplace`, if the birth place is Canada, the result will be kept. Otherwise, the results will be filtered. The limit is used to limit the number of results.

```
SELECT ?name
WHERE {
?a <http://dbpedia.org/property/name> ?name.
?a dbo:birthPlace ?country.
filter regex(str(?country), "Canada")
} limit 100
```

The following DBpedia SPARQL query is based on the previous query has more complex condition by using `filter` and `not exist`. In the following query, it extracts the first 100 people's name that the people's birth place is Canada, while the `not exist` has another condition that the people's birth place should not be Toronto. Since Toronto is a city in Canada, they are linked in DBpedia. The second `filter` will filter some people's birth place is Toronto.

```
SELECT ?name
WHERE {
?a <http://dbpedia.org/property/name> ?name.
?a dbo:birthPlace ?country
filter regex(str(?country), "Canada")
filter not exists{
{
?a <http://dbpedia.org/property/name> ?name.
?a dbo:birthPlace ?city
filter regex(str(?city), "Toronto")
}
}
} limit 100
```