information

MDPI

Article

A Comparison of Word Embeddings and N-gram Models for DBpedia Type and Invalid Entity Detection †

Hanqing Zhou *, Amal Zouaq and Diana Inkpen

School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa ON K1N 6N5, Canada; azouaq@uottawa.ca (A.Z.); diana.inkpen@uottawa.ca (D.I.)

* Correspondence: hzhou020@uottawa.ca; Tel.: +1-613-562-5800

† This paper is an extended version of our conference paper: Hanqing Zhou, Amal Zouaq, and Diana Inkpen. 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, 8–10 November 2017, pp. 309–322.

Received: 6 November 2018; Accepted: 20 December 2018; Published: 25 December 2018

Abstract: This article presents and evaluates a method for the detection of DBpedia types and entities that can be used for knowledge base completion and maintenance. This method compares entity embeddings with traditional N-gram models coupled with clustering and classification. We tackle two challenges: (a) the detection of entity types, which can be used to detect invalid DBpedia types and assign DBpedia types for type-less entities; and (b) the detection of invalid entities in the resource description of a DBpedia entity. Our results show that entity embeddings outperform n-gram models for type and entity detection and can contribute to the improvement of DBpedia's quality, maintenance, and evolution.

Keywords: semantic web; DBpedia; entity embedding; n-grams; type identification; entity identification; data mining; machine learning

1. Introduction

The Semantic Web is defined by Berners-Lee et al. [1] 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. In this context, 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) [2,3].

Knowledge bases represent the backbone of the Semantic Web, and they also represent Web resources [4]. Knowledge bases are being created by the integration of resources like Wikipedia and Linked Open Data [5,6] and have turned into a crystallization point for the emerging Web of Data [7,8]. 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 [9]. 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/) [10,11]. 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 [12], knowledge extraction [13], information retrieval [14], querying [15,16] and question answering [17].

Given the automatic extraction framework of DBpedia, along with its dynamic nature, several inconsistencies can exist in DBpedia [18–21]. One important quality problem is related to DBpedia types (classes). In fact, there exist invalid DBpedia types for some entities. For example, the DBpedia entity "dbr:Xanthine (http://dbpedia.org/resource/Xanthine)" belongs to three DBpedia types: "dbo:ChemicalSubstance"(http://dbpedia.org/ontology/ChemicalSubstance), "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 [22]. Consequently, some entities are not correctly linked to other entities. For instance, the description of the DBpedia entity "dbr:Earthquake" contains the triple <dbr:Vanilla_Ice dbo:genre dbr:Earthquake>, which is an erroneous fact. Thus, "dbr:Vanilla_Ice" is considered an invalid entity in the resource description of "dbr:Earthquake".

Identifying these invalid types and entities manually is unfeasible. In fact, the automatic enrichment and update of DBpedia with new type statements (through rdf:type) is becoming an important challenge [23,24]. In this paper, we rely on vector-based representations such as word embeddings and entity embeddings [25,26] to reach these objectives. Word embeddings are vector representations of word(s) [27–29] that have several applications in natural language processing tasks, such as entity recognition [30], information retrieval [31], and question answering [32]. Entity embeddings, as defined in this article, are similar to word embeddings [27–29] but they differ in that they are based on vectors that describe the entity URIs instead of words [25,26].

In this article, we compare two vector-based representations that emerge from text, that is, n-gram models and entity embeddings models. Both types of representations are learned from Wikipedia text and represent n-grams and entities (URIs) using a vector space model. In fact, n-gram models have long been a strong baseline for text classification. The emergence of entity URIs from one side and word embeddings models from the other side represents a good opportunity for entity representation using a vector model and for the comparison of the performance of sparse vector space models with dense vectors in natural language processing tasks.

Overall, we aim at addressing the following research questions:

RQ1: How do entity embeddings compare with traditional n-gram models for type identification?

RQ2: How do entity embeddings compare with traditional n-gram models for invalid entity detection?

The article is structured as follows. In Section 1, we discuss the motivation of this work, our goals and contributions. Section 2 presents background information about automatic type identification and erroneous information detection. In Section 3, we discuss our work for collecting datasets, building entity embedding and n-gram models. Then, in Section 4, we propose different clustering and classification methods to detect invalid DBpedia types, and complete missing types for 358 types from the DBpedia ontology. In Section 5, we present our approach to detect invalid DBpedia resources using our own entity embedding and n-gram models. In Section 6, we run our best trained models on the whole DBpedia, in order to see how many wrong type and invalid entities we find. In Section 7, we conclude our work and contributions, and we discuss possible directions of work for the future. This article is an extended version of our published conference paper [33]. In particular, Section 4 of this article is based on the conference paper [33].

2. Background and Related Work

2.1. DBpedia and Linked Data Quality Assessment and Enhancement

Several approaches in the literature aim to enhance the quality of DBpedia and Linked Data. Paulheim [34] 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 the DBpedia knowledge base, namely, the completeness problem and the correctness problems, and they also pointed out several approaches for the DBpedia knowledge base refinement. The survey by Färber et al. [35] analyzed the five most 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. [36] proposed 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 that is used during crowdsourcing quality assessment. The quality problem taxonomy has four dimensions: accuracy, relevancy, representational-consistency, and interlinking. Based on this evaluation, Zaveri et al. [36] identified around 222,298 incorrect triples and concluded that 11.93% of the tested DBpedia triples were having quality issues.

Other approaches focused on particular aspects of the DBpedia knowledge base. For example, Font et al. [19,37] performed an in-depth evaluation of domain knowledge representation in DBpedia. The article 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. [20] using rule-based reasoning with MapReduce. Five different types of inconsistencies were 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. [21] 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 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 found 50 thousand inconsistent instances. Likewise, Lehmann and Bühmann [38] 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: incorrect property ranges and incompatibility problems with external ontologies.

2.2. Automatic Type Detection

There are several cutting-edge related works for automatic type detection [39–44].

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 [19] 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 Röder et al. [43], which implements a pattern extraction from DBpedia abstracts and creates local type hierarchies based on the extracted types. The final step maps the structured types to the DOLCE + DnS ontology (http://stlab.istc.cnr.it/stlab/WikipediaOntology/) classes. Another automatic DBpedia

entity typing approach is proposed by Gangemi et al. [39], which presents 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.

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 is called SDType (Statistical Distribution Typing) by Paulheim and Bizer [41] and Paulheim and Bizer [42]. 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 link 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 38,000 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 [44] which combines word embeddings with external information for entity typing. The vector models represent the word embeddings learned using Word2vec.

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 [44] but they combine word embeddings with external information while we rely on the information in Wikipedia to learn entity embeddings for entity typing.

2.3. Outlier Detection in Linked Data

Several approaches for detecting invalid or faulty knowledge in DBpedia have been proposed in the state of the art [9,11,38], 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. [18]. 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.

SDValidate, proposed by Paulheim and Bizer [42], 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 a confidence score for the selected statements based on statistical distributions related to 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 statement. 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 [42], SDValidate detected 13,000 erroneous statements in DBpedia.

Another outlier detection approach to enhance quality of linked data is by Debattista et al. [45], 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, 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.

3. Methodology

This section 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 at the date of our test in March 2017. In this version of the DBpedia ontology and knowledge base, only 461 classes had instances (through rdf:type). Among these 461 types, we ignored the classes with less than 20 instances (199), which led to 358 classes for our experiments. Examples of some of the 199 classes include "ArtisticGenre", "TeamSport", and "SkiResort".

3.1. General Architecture

This section describes the general architecture of the various modules involved in our experiments (See Figure 1).

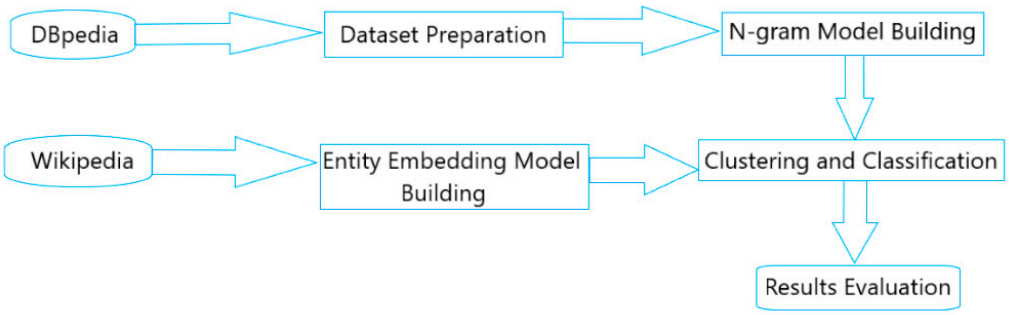

Figure 1. Flowchart of the DBpedia invalid type and entity detection pipeline.

The first step is the development of the entity embedding models that are built with Wiki2vec (https://github.com/idio/wiki2vec) with input from the Wikipedia dumps (https://dumps.wikimedia.org/enwiki/). The second step is the development of n-gram models that are extracted with the Weka n-gram tokenizer (http://weka.sourceforge.net/doc.dev/weka/core/tokenizers/NGramTokenizer.html) 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

Initially, we relied on pre-built models from wiki2vec for the English Wikipedia (February 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 this problem, we used both the Skip-gram [28,29] and CBOW [28,29] architectures for learning our models. To increase the coverage, we trained our model on Wikipedia 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 [27–29] 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 makes it possible 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) 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 (http://www.nltk.org/) 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 with the types that are already represented in DBpedia, when applicable. For example, the vector related to "dbr:Bill_Clinton" is associated with the type "dbo:President". To give a better idea about this notion of distance, a visualization plot based on t-SNE (https://lvdmaaten.github.io/tsne/) is provided in Figure 2.

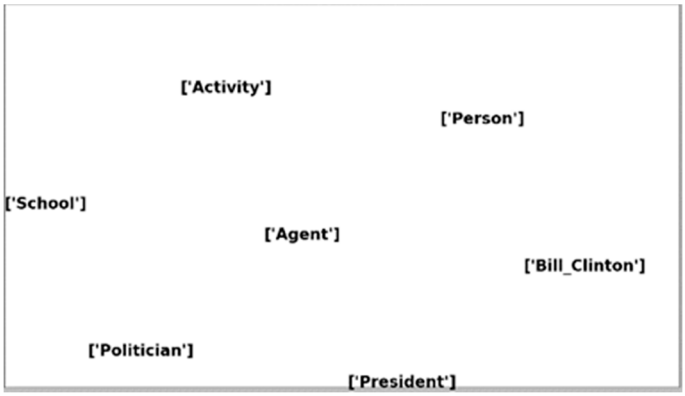

Figure 2. Visualization of DBpedia types for the entity Bill_Clinton.

The 2-D plot is based on a 100-dimension vector representation and uses PCA (Principle Component Analysis) [46] 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 (http://dbpedia.org/page/Barack_Obama)". For single words, the dataset usually contains both DBpedia entities and words. For example, both "DBPEDIA_ID/Poetry" and "Poetry" are represented in our model. We only use DBpedia entity "DBPEDIA_ID/Poetry" in our tasks, since we are only interested in DBpedia entities.

3.3. Dataset 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 SPARQL endpoint (http://dbpedia.org/sparql).

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 the 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 2111 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 (http://www.site.uottawa.ca/~{}diana/resources/kesw17/).

3.3.2. Dataset for Invalid Entity Detection

The second dataset for the task of detecting invalid DBpedia resources is collected separately from the previous datasets. We randomly select 120,000 and 16,000 entities as training and test datasets that are available in our entity embedding models through DBpedia SPARQL endpoint. For each of the selected DBpedia entities, we need all of their corresponding DBpedia resources; that is, we select all the objects related to that particular entity through a property. These related resources are tagged as positive examples of valid entities in the target DBpedia entity description. 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, "dbr:Hawaii" has five classes: "dbo:Place", "dbo:Location", "dbo:City", "dbo:PopulatedPlace", and "dbo:Settlement". We do not select any entity that is associated with these classes. The number of negative entities depends on the corresponding number of positive entities in order to build a balanced dataset.

Table 1 summarizes the size of the three datasets.

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

Dataset	Training Dataset for DBpedia Invalid Type Detection	Test Dataset for DBpedia Invalid Type Detection	Training Dataset for DBpedia Invalid Entity Detection	Test Dataset for DBpedia Invalid Entity Detection
Number of Entities	360,843	2111	160,000	12,000

3.4. N-gram Model Building

The n-gram models are extracted using the Weka n-gram tokenizer. We first collect the DBpedia entities' abstracts through DBpedia public SPARQL endpoint for preparing the training and test datasets. Then we learn the n-gram models using Weka by varying the length of the word sequences (n = 1...3) and using TF-IDF to weight the n-grams. We performed Random Forest feature selection [47–50] in order to reduce the number of features to 1000. This number was determined after a set of empirical experiments.

We obtained 36,842 uni-grams, 62,637 bi-grams, 81,830 tri-grams, and 181,309 n-grams. With the Random Forest feature selection technique, the dimension of n-gram models decreased to 1000.

3.5. Evaluation Metrics

The evaluation measures include Precision, Recall, F-Score, Accuracy, and Area Under the Curve (AUC) (for the classification experiments). Finally, the Student's *t*-test measure is used to assess the statistical significance of our results.

4. DBpedia Entity Type Detection

4.1. Experiment Setup

This phase consists of two parts: clustering and classification. Both parts use Scikit-Learn (http://scikit-learn.org/stable/) [51] 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 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 using the Euclidean distance. 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*. These entities are considered as examples of the positive class, the same number of negative entities is then selected from DBpedia excluding the entities from the positive class. Because the produced clusters are not labelled, the cluster with the highest 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).

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

4.2.1. Clustering with N-gram Models

This section provides clustering with N-gram models.

Based on the results of the uni-gram, bi-gram, tri-gram and n-gram models (Table 2); the uni-gram, bi-gram and tri-gram models have similar performance. According to the results of Tables 2–5, the n-gram model differs significantly from the other three models; it performs around 10% better than the other three models in terms of F-score. We report the improvements as differences (percentage

points). According to the clustering results with the n-gram model (Table 2), Birch outperform by around 40% Mean Shift in terms of F-score.

Table 2. Results of clustering algorithms with n-gram models on the training dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy
Uni-gram	K-means	0.819	0.657	0.669	66.3%
	Mean Shift	0.357	0.431	0.367	27.4%
	Birch	0.811	0.745	0.726	70.2%
Bi-gram	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%
Tri-gram	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%
N-gram	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 3. Summary for clustering algorithms with n-gram model on test dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy
Uni-gram	K-means	0.505	0.940	0.657	50.9%
	Birch	0.507	0.933	0.654	50.8%
Bi-gram	K-means	0.503	0.937	0.654	50.5%
	Birch	0.505	0.933	0.653	50.5%
Tri-gram	K-means	0.513	0.930	0.655	51.1%
	Birch	0.514	0.929	0.655	51.2%
N-gram	K-means	0.507	0.935	0.656	50.9%
	Birch	0.510	0.945	0.662	51.6%

Table 4. Summary for clustering with entity embedding models on the training dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy
Skip-Gram	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%
CBOW	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 5. Summary for clustering algorithms with entity embedding model on test dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy
Skip-Gram	Kmeans	0.820	0.935	0.856	83.0%
	Birch	0.837	0.932	0.861	83.7%
CBOW	K-means	0.543	0.924	0.667	53.8%
	Birch	0.535	0.916	0.660	52.8%

Given our results on the training datasets, where K-means and Birch obtained the best performance with both entity and n-gram models, we report only their results on the test set. Table 3 shows the clustering results of K-means and Birch on the test dataset with different n-gram models.

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, and there is no significant difference between them.

4.2.2. Clustering with Entity Embedding Models

Here we present the clustering results both on the training dataset and the held-out test dataset using entity embedding models.

We explored two word embedding models in our experiments. Table 4 shows the average results for clustering on the 358 DBpedia types with Skip-gram and CBOW entity embedding models. We report the metrics for the positive class only.

Based on the results of the two entity embedding models, the performance of Skip-gram is usually better than CBOW. Skip-gram outperforms the CBOW model with about 26% in terms of F-Score. For Accuracy, the difference is even bigger; Skip-gram surpasses by 32% CBOW. Furthermore, K-means has similar performance as Birch, and their performances are much better than that of Mean Shift. The Skip-gram entity embedding model has the best performance among all of the entity embedding and n-gram models based on the training dataset, followed by the CBOW entity embedding model. We also performed clustering on the test set (Table 5).

The superiority, in terms of accuracy and F-Score, of the skip-gram model is again apparent on the test set. There was no significant difference between clustering with CBOW and the n-gram models.

4.3. Classification Evaluation

We experimented 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 the N-gram models. Table 6 shows the average classification results (10-fold cross validation) with the n-gram models on the training dataset.

Table 6. Evaluation of classification algorithms with n-gram models on the training dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Uni-gram	Baseline	0.498	0.496	0.485	49.7%	0.500
	DT	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
	RF	0.932	0.858	0.886	90.1%	0.951
	NB	0.835	0.927	0.871	86.9%	0.876
	SVM	0.961	0.837	0.884	90.6%	0.960
Bi-gram	Baseline	0.497	0.497	0.485	49.5%	0.801
	DT	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
	RF	0.934	0.859	0.888	90.2%	0.956
	NB	0.836	0.928	0.872	86.9%	0.874
	SVM	0.956	0.833	0.879	90.2%	0.955
Tri-gram	Baseline	0.501	0.501	0.489	49.9%	0.501
	DT	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
	RF	0.938	0.860	0.891	90.5%	0.957
	NB	0.840	0.930	0.875	87.2%	0.880
	SVM	0.953	0.832	0.878	90.1%	0.950

Table 6. *Cont.*

Model	Algorithm	Precision	Recall	F-Score	Accuracy	AUC
n-gram	Baseline	0.504	0.501	0.490	50.2%	0.499
	DT	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
	RF	0.955	0.911	0.928	93.7%	0.977
	NB	0.855	0.939	0.887	88.4%	0.894
	SVM	0.964	0.895	0.921	93.6%	0.976

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.

For our experiments on the test datasets, we selected the best classification algorithms (Extra tree, RF and SVM) based on the 358 tested DBpedia types on n-gram models (Table 7).

Table 7. Summary for classification algorithms with n-gram model on test dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Uni-gram	Extra Tree	0.962	0.916	0.932	93.8%	0.974
	RF	0.959	0.896	0.918	92.8%	0.967
	SVM	0.982	0.874	0.912	92.7%	0.970
Bi-gram	Extra Tree	0.948	0.851	0.886	90.1%	0.952
	RF	0.949	0.833	0.874	89.3%	0.948
	SVM	0.968	0.778	0.842	87.9%	0.953
Tri-gram	Extra Tree	0.920	0.815	0.847	86.2%	0.915
	RF	0.919	0.800	0.836	85.0%	0.916
	SVM	0.941	0.697	0.766	81.7%	0.913
n-gram	Extra Tree	0.970	0.906	0.930	93.7%	0.974
	RF	0.964	0.898	0.921	93.0%	0.968
	SVM	0.987	0.962	0.907	92.4%	0.965

Similar to the results on the training dataset, 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, the difference between the n-gram model and the uni-gram model is not significant in terms of F-score and Accuracy.

4.3.2. Classification with Entity Embedding Models

Table 8 compares the results of the Skip-gram and CBOW models using the same classification algorithms.

Based on the classification results on entity embedding models on training dataset (Table 8), 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. For the results with Skip-gram entity embeddings, Extra Tree, Random Forest, Naïve Bayes, and SVM get close performance with an F-Score and Accuracy greater than or close to 0.95. When switching to the CBOW model, the performance of these classification algorithms drops by at least 10%, except for the baseline algorithm.

Based on the overall results of entity embedding and n-gram models, tree-based algorithms and SVM over-perform the other results. 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.

Table 8. Evaluation of classification algorithms with entity embedding models on the training dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Skip-Gram	Baseline	0.499	0.497	0.487	49.8%	0.502
	DT	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
	RF	0.958	0.935	0.944	94.9%	0.986
	NB	0.967	0.955	0.959	96.4%	0.991
	SVM	0.958	0.986	0.970	97.3%	0.995
CBOW	Baseline	0.496	0.500	0.486	49.7%	0.503
	DT	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
	RF	0.936	0.857	0.887	90.1%	0.955
	NB	0.837	0.925	0.871	87.1%	0.877
	SVM	0.959	0.836	0.883	90.3%	0.955

Furthermore, like the results in the clustering part, the Skip-gram entity embedding model still has the best performance among all models. The Skip-gram model obtains a performance slightly better than the n-gram model in our classification experiments, but 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 respectively.

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 9), the Skip-gram model still performs better than the CBOW model with a drop of around 10% with CBOW. 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.

Table 9. Summary for classification algorithms with the entity embedding model on the test dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Skip-Gram	Extra Tree	0.972	0.931	0.946	95.2%	0.986
	RF	0.961	0.922	0.938	94.5%	0.983
	SVM	0.966	0.967	0.962	96.5%	0.991
CBOW	Extra Tree	0.920	0.850	0.875	88.6%	0.947
	RF	0.917	0.948	0.872	88.3%	0.941
	SVM	0.926	0.890	0.897	90.4%	0.962

4.4. t-Tests

Several Student's *t*-tests were applied on the precision, recall, f-score, and accuracy results of the Skip-gram entity embedding and the n-gram models, using SVM, the best classification algorithm on both the training and test dataset with one-tailed hypothesis. The details of Student's *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 10.

Table 10. Student's *t*-tests based on the training and test datasets.

Dataset	Results	Significance Level	*t* Value	*p* Value	Student's *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

The results on the training dataset are not statistically significant for precision. However, recall, f-score and accuracy results are statistically significant at $p < 0.01$. For the Student's *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.

4.5. 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 used.

5. DBpedia Invalid Entity Detection in Resources Description

This section describes the task of DBpedia invalid entity detection in resource description, including the methods and the experimental setups 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. 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. 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" and "dbo:Education" for instance.

At the time of our experiments, there were several outliers and invalid facts in some resource 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 later. 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", "dbr:Hydrazine", "dbr:Johns_Hopkins_University", etc., have indeed been detected as invalid entities. The original entities from the "Cake" resource description, such as "dbr:Cupcake",

"dbr:Butter", "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, "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. 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 a training part and a testing set, and we perform a 10-fold cross validation. Finally, we report the evaluation results of the negative class only as we want to test the ability to detect invalid entities.

5.2. Clustering Evaluation

5.2.1. Clustering with N-gram Models

Table 11 shows the clustering results of K-means, Mean Shift and Birch on the training dataset with different n-gram models.

Table 11. Summary for clustering algorithms with n-gram model on the training dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy
Uni-gram	K-means	0.691	0.766	0.657	59.5%
	Mean Shift	0.501	0.997	0.667	47.0%
	Birch	0.558	0.450	0.406	60.3%
Bi-gram	K-means	0.482	0.104	0.108	51.3%
	Mean Shift	0.509	0.990	0.671	32.0%
	Birch	0.591	0.089	0.125	51.7%
Tri-gram	K-means	0.465	0.087	0.095	50.6%
	Mean Shift	0.491	0.995	0.657	39.4%
	Birch	0.573	0.099	0.128	51.4%
N-gram	K-means	0.490	0.134	0.136	52.0%
	Mean Shift	0.537	0.991	0.695	36.1%
	Birch	0.343	0.020	0.025	49.5%

We observe that the uni-gram, bi-gram and tri-gram models have similar performance. We also observe that outlier detection based on n-gram models is a difficult task with a top accuracy of ~60%. Table 12 shows the clustering results of K-means and Birch on the test dataset with different n-gram models, which show a similar pattern than the results on the training dataset.

Table 12. Summary for clustering algorithms with n-gram model on test dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy
Uni-gram	K-means	0.556	0.456	0.448	60.8%
	Birch	0.422	0.089	0.121	49.4%
Bi-gram	K-means	0.595	0.104	0.104	51.9%
	Birch	0.543	0.029	0.051	50.4%
Tri-gram	K-means	0.510	0.066	0.068	50.4%
	Birch	0.473	0.021	0.037	50.0%
N-gram	K-means	0.605	0.150	0.144	50.3%
	Birch	0.603	0.033	0.060	50.8%

5.2.2. Clustering with Entity Embedding Models

Table 13 describes summary results for clustering algorithms with entity embedding model on training dataset. We report the metrics for the negative class only.

Table 13. Summary for clustering algorithms with entity embedding models on training dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy
Skip-Gram	K-means	0.931	0.874	0.894	91.2%
	Mean Shift	0.496	0.992	0.661	49.5%
	Birch	0.906	0.854	0.864	88.0%
CBOW	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%

We also notice, from Tables 13 and 14, that, compared to n-gram models, the accuracy increases significantly to over 90% with entity embedding models. Similarly to our previous observations, the Skip-gram model outperforms the CBOW model.

Table 14. Summary for clustering algorithms with entity embedding models on the test dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy
Skip-Gram	K-means	0.705	0.944	0.807	77.3%
	Birch	0.918	0.245	0.393	61.1%
CBOW	K-means	0.570	0.265	0.308	54.4%
	Birch	0.518	0.084	0.128	50.9%

The results of the Skip-gram model exceed by 35% the CBOW model in terms of F-Score, and even more in terms of Accuracy.

5.3. Classification Evaluation

5.3.1. Classification with N-gram Models

Tables 15 and 16 show the classification results with various traditional N-gram models on the training and test datasets. Based on these 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.

On the test set, 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.

Table 15. Summary for classification algorithms with n-gram model on training dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Uni-gram	Baseline	0.500	0.497	0.493	49.7%	0.500
	DT	0.767	0.810	0.781	77.7%	0.779
	Extra Tree	0.776	0.902	0.826	81.1%	0.817
	KNN	0.479	0.700	0.498	51.9%	0.550
	RF	0.775	0.887	0.816	80.2%	0.808
	NB	0.832	0.941	0.873	88.2%	0.872
	SVM	0.906	0.854	0.876	86.6%	0.960
Bi-gram	Baseline	0.500	0.503	0.496	49.7%	0.500
	DT	0.729	0.751	0.722	72.4%	0.726
	Extra Tree	0.704	0.867	0.748	72.1%	0.734
	KNN	0.476	0.804	0.548	49.7%	0.521
	RF	0.710	0.839	0.734	71.3%	0.727
	NB	0.777	0.792	0.709	84.9%	0.750
	SVM	0.890	0.798	0.837	73.0%	0.847
Tri-gram	Baseline	0.500	0.501	0.495	49.8%	0.501
	DT	0.709	0.694	0.654	67.5%	0.682
	Extra Tree	0.708	0.739	0.650	65.7%	0.673
	KNN	0.447	0.726	0.492	49.4%	0.519
	RF	0.708	0.701	0.625	64.6%	0.664
	NB	0.751	0.785	0.694	78.8%	0.728
	SVM	0.830	0.737	0.773	70.6%	0.790
N-gram	Baseline	0.499	0.501	0.495	49.7%	0.500
	DT	0.768	0.819	0.785	78.2%	0.785
	Extra Tree	0.772	0.907	0.821	80.5%	0.812
	KNN	0.531	0.751	0.536	52.1%	0.544
	RF	0.767	0.887	0.805	79.1%	0.800
	NB	0.751	0.801	0.699	87.2%	0.741
	SVM	0.914	0.820	0.861	71.9%	0.869

Table 16. Summary of classification algorithms with n-gram model on test dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Uni-gram	Extra Tree	0.674	0.693	0.674	67.2%	0.675
	RF	0.678	0.707	0.682	67.8%	0.681
	SVM	0.833	0.806	0.820	82.5%	0.825
Bi-gram	Extra Tree	0.737	0.851	0.784	76.8%	0.770
	RF	0.754	0.849	0.793	78.0%	0.783
	SVM	0.704	0.894	0.826	88.8%	0.778
Tri-gram	Extra Tree	0.681	0.775	0.718	70.2%	0.705
	RF	0.693	0.765	0.720	70.8%	0.712
	SVM	0.955	0.824	0.884	88.8%	0.890
Mix-gram	Extra Tree	0.769	0.877	0.815	80.3%	0.805
	RF	0.792	0.879	0.829	81.9%	0.821
	SVM	0.833	0.909	0.870	71.2%	0.830

5.3.2. Classification with Entity Embedding Models

As shown in Tables 17 and 18, Extra Tree, Random Forest, and SVM have high-quality results, and can be used to detect most DBpedia entities correctly. When switching from the Skip-gram model to the CBOW model, the overall results drop by around 5%.

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. The Extra Tree, Random Forest, Support Vector Machine have good performance with entity embedding models for the detection of the negative class.

Table 17. Summary of classification algorithms with entity embedding model on training dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Skip-Gram	Baseline	0.503	0.501	0.493	49.7%	0.501
	DT	0.835	0.834	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
	RF	0.886	0.927	0.902	90.3%	0.903
	NB	0.946	0.934	0.937	93.9%	0.939
	SVM	0.957	0.941	0.947	94.8%	0.949
CBOW	Baseline	0.504	0.499	0.493	96.7%	0.501
	DT	0.784	0.785	0.778	77.9%	0.782
	Extra Tree	0.839	0.909	0.868	86.4%	0.866
	KNN	0.923	0.869	0.878	89.4%	0.896
	RF	0.830	0.903	0.860	85.5%	0.858
	NB	0.928	0.921	0.921	92.2%	0.923
	SVM	0.958	0.942	0.947	94.9%	0.950

Table 18. Summary of classification algorithms with entity embedding model on test dataset.

Model	Algorithm	Precision	Recall	F-Score	Accuracy	AUC
Skip-Gram	Extra Tree	0.64.3	0.664	0.647	64.2%	0.644
	RF	0.784	0.879	0.829	81.5%	0.814
	SVM	0.833	0.983	0.902	89.2%	0.882
CBOW	Extra Tree	0.643	0.664	0.647	64.2%	0.644
	RF	0.651	0.684	0.661	65.5%	0.657
	SVM	0.833	0.909	0.870	84.2%	0.830

6. Evaluation on the Whole DBpedia Knowledge Base

In this section, we present the results of applying the Random Forest classification model on 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 coupled with the skip-grams model. Random Forest displayed good overall performance, and most of its results were 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 the Random Forest classifier for our experiments on the whole DBpedia knowledge base.

There were around 4.7 million resources in DBpedia at the time of our experiments. 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 that we first used, where only 0.22 million 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).

New types discovery. 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 had 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:Agent", "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 with new types. There are around two new types per entity, on average. The percentage of entities with new types is 5.74%.

Table 19 shows the detailed results for the top five DBpedia classes that have the highest number of new DBpedia entities.

Table 19. DBpedia new entities based on the whole DBpedia knowledge base.

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%

Below is a set of new rdf:type triples found by our classification experiments:

```
<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>
```

Type Confirmation. 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 had 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 had the same types as those available in 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 20 shows the detailed results for the top five DBpedia classes that have the highest number of similar entities as DBpedia based on our methods.

Table 20. DBpedia similar entities based on the whole DBpedia knowledge base.

DBpedia Class	Number of Similar Entities	Percentage of Similar Entities
CareerStation	10,001	0.711%
Document	9983	0.710%
PersonFunction	9958	0.708%
Sound	9856	0.706%
SportsTeamMember	9833	0.699%

Entity Extraction. The third experiment counted how many classes were related to 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 using our 358 classifiers. Then we compared the classification results with the DBpedia knowledge base to find the number of classes with new entities. For example, the class "dbo:Politician" originally had more than a hundred thousand entities, such as "dbr:Barack_Obama", "dbr:George_Bush", "dbr:Bill_Clinton", etc. Following our experiment, "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 obtained new entities (100%).

Invalid RDF Triple Detection. 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 correctly classified 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%.

The following examples show several invalid triples detected using 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>
```

Table 21 shows the detailed results for the top five DBpedia classes that have the biggest number of invalid triples.

Table 21. DBpedia invalid RDF triples.

DBpedia Class	Number of Invalid RDF Triples	Percentage of Invalid RDF Triples
ComicsCharacter	6670	1.603%
Language	10,456	2.513%
Person	5538	1.331%
Place	7354	1.767%
Species	16,027	3.852%

7. Conclusions and Future Work

In this article, 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 in resources description. We compared the results of different clustering and classification algorithms, and the results of different entity embedding and n-gram models.

In the DBpedia entity type detection part (Section 4), the experiments show that we 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 (Section 5), our experimental results show that we can detect most of the invalid DBpedia entities correctly. In the experiments in both parts (Sections 4 and 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. Given an accuracy greater than or equal to 96%, we are confident that our approach is able to complete and correct the DBpedia knowledge base.

Recalling the two research questions that we proposed at the beginning of this article:

RQ1: How do entity embeddings compare with traditional n-gram models for type identification?

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. However, not all entity embedding models outperform traditional n-gram models. N-gram and uni-gram models performed better than the Continuous Bag-Of-Words (CBOW) entity embedding model.

RQ2: How do entity embeddings compare with traditional n-gram models for invalid entity detection?

All the entity embedding models (Skip-Gram, CBOW) performed better than the traditional n-gram models. Entity embeddings, and more precisely the Skip-Gram model, helped detect the relevant entities in the RDF description of a DBpedia resource. Based on our results with SVM, we showed that around 95% of the relevant entities in the RDF description of a DBpedia resource can be detected with the Skip-gram entity embedding model.

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, even if they 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 SDValidate and SDType [41,42]. 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.

Another limitation is that 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 impacts 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 for the 199 ontological classes without any entities in DBpedia is an important avenue for the future. Another important direction is to build larger entity embedding models to include more DBpedia entities, and also to

explore how graph embeddings models such as node2vec [52] can be used in similar experiments. In terms of the entity embedding model building, we also plan to try other entity embedding tools, such as GloVe [53] to see if they can lead to better models.

Author Contributions: Conceptualization, A.Z. and D.I.; Methodology, A.Z., and D.I.; Software, H.Z.; Validation, H.Z., A.Z., and D.I.; Data set preparation, H.Z.; Writing—Original Draft Preparation, H.Z.; Writing—Review and Editing, A.Z. and D.I.; Supervision and Funding, A.Z. and D.I.

Funding: This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Acknowledgments: We thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for the financial support.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Berners-Lee, T.; Hendler, J.; Lassila, O. The semantic web. *Sci. Am.* **2001**, *284*, 34–43. [CrossRef]
2. Bizer, C.; Heath, T.; Berners-Lee, T. Linked Data—The Story So Far. *Int. J. Semant. Web Inf. Syst.* **2009**, *5*, 1–22. [CrossRef]
3. Bizer, C.; Lehmann, J.; Kobilarov, G.; Auer, S.; Becker, C.; Cyganiak, R.; Hellmann, S. DBpedia—A crystallization point for the Web of Data. *J. Web Semant.* **2009**, *7*, 154–165. [CrossRef]
4. Kabir, S.; Ripon, S.; Rahman, M.; Rahman, T. Knowledge-based Data Mining Using Semantic Web. *IERI Procedia* **2014**, *7*, 113–119. [CrossRef]
5. König, M.; Dirnbek, J.; Stankovski, V. Architecture of an open knowledge base for sustainable buildings based on Linked Data technologies. *Autom. Construct.* **2013**, *35*, 542–550. [CrossRef]
6. Syed, Z.; Finin, T. Creating and Exploiting a Hybrid Knowledge Base for Linked Data. In Proceedings of the Second International Conference, ICAART 2010, Valencia, Spain, 22–24 January 2010. [CrossRef]
7. Hellmann, S.; Stadler, C.; Lehmann, J.; Auer, S. DBpedia live extraction. In Proceedings of the OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", Vilamoura, Portugal, 1–6 November 2009; Volume 5871, pp. 1209–1223.
8. Moran, S. Using Linked Data to Create a Typological Knowledge Base. In *Linked Data in Linguistics: Representing and Connecting Language Data and Language Metadata*; Springer: Berlin/Heidelberg, Germany, 2012. [CrossRef]
9. Morsey, M.; Lehmann, J.; Auer, S.; Stadler, C.; Hellmann, S. DBpedia and the live extraction of structured data from Wikipedia. *Program* **2012**. [CrossRef]
10. Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; Ives, Z. DBpedia: A nucleus for a Web of open data. In Proceedings of the 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, 11–15 November 2007. [CrossRef]
11. Lehmann, J.; Isele, R.; Jakob, M.; Jentzsch, A.; Kontokostas, D.; Mendes, P.N.; Hellmann, S.; Morsey, M.; van Kleef, P.; Auer, S.; et al. DBpedia—A large-scale, multilingual knowledge base extracted from Wikipedia. *Semant. Web* **2015**, *6*, 167–195. [CrossRef]
12. Zhang, Z.; Chen, S.; Feng, Z. Semantic annotation for web services based on DBpedia. In Proceedings of the 2013 IEEE 7th International Symposium on Service-Oriented System Engineering (SOSE 2013), San Francisco Bay, CA, USA, 25–28 March 2013. [CrossRef]
13. Alani, H.; Kim, S.; Millard, D.E.; Weal, M.J.; Hall, W.; Lewis, P.H.; Shadbolt, N.R. Automatic Ontology-Based Knowledge Extraction from Web Documents. *IEEE Intell. Syst.* **2003**, *18*, 14–21. [CrossRef]
14. Keong, B.V.; Anthony, P. Meta search engine powered by DBpedia. In Proceedings of the 2011 International Conference on Semantic Technology and Information Retrieval (STAIR 2011), Kuala Lumpur, Malaysia, 27–29 June 2011. [CrossRef]
15. Meimaris, M.; Papastefanatos, G.; Mamoulis, N.; Anagnostopoulos, I. Extended Characteristic Sets: Graph Indexing for SPARQL Query Optimization. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19–22 April 2017; pp. 497–508. [CrossRef]

16. Wang, M.; Wang, R.; Liu, J.; Chen, Y.; Zhang, L.; Qi, G. Towards Empty Answers in SPARQL: Approximating Querying with RDF Embedding. In Proceedings of the 17th International Semantic Web Conference, Monterey, CA, USA, 8–12 October 2018; Vrandečić, D., Bontcheva, K., Suárez-Figueroa, M.C., Celino, V.P., Sabou, M., Kaffee, Lu., Simperl, E., Eds.; Springer: Cham, Switzerland, 2018; Volume 11136.
17. Azmy, M.; Shi, P.; Lin, J.; Ilyas, I.F. Farewell Freebase: Migrating the SimpleQuestions Dataset to DBpedia. In Proceedings of the 27th International Conference on Computational Linguistics, Santa Fe, NM, USA, 20–26 August 2018.
18. Fleischhacker, D.; Paulheim, H.; Bryl, V.; Völker, J.; Bizer, C. Detecting errors in numerical linked data using cross-checked outlier detection. In Proceedings of the 13th International Semantic Web Conference, Riva del Garda, Italy, 19–23 October 2014. [CrossRef]
19. Font, L.; Zouaq, A.; Gagnon, M. Assessing the Quality of Domain Concepts Descriptions in DBpedia. In Proceedings of the 11th International Conference on Signal-Image Technology and Internet-Based Systems (SITIS 2015), Bangkok, Thailand, 23–27 November 2015. [CrossRef]
20. Sheng, Z.; Wang, X.; Shi, H.; Feng, Z. Checking and handling inconsistency of DBpedia. In Proceedings of the International Conference on Web Information Systems and Mining (WISM 2012), Chengdu, China, 26–28 October 2012. [CrossRef]
21. Töpper, G.; Knuth, M.; Sack, H. DBpedia ontology enrichment for inconsistency detection. In Proceedings of the 8th International Conference on Semantic Systems—I-SEMANTICS'12, Graz, Austria, 5–7 September 2012. [CrossRef]
22. Wienand, D.; Paulheim, H. Detecting incorrect numerical data in DBpedia. In Proceedings of the 11th International Conference, The Semantic Web: Trends and Challenges (ESWC 2014), Anissaras, Greece, 25–29 May 2014. [CrossRef]
23. Kliegr, T. Linked hypernyms: Enriching DBpedia with Targeted Hypernym Discovery. *J. Web Semant.* **2015**, *31*, 59–69. [CrossRef]
24. Kliegr, T.; Zamazal, O. Towards Linked Hypernyms Dataset 2.0: Complementing DBpedia with hypernym discovery. In Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014), Reykjavik, Iceland, 26–31 May 2014. [CrossRef]
25. Chen, T.; Tang, L.A.; Sun, Y.; Chen, Z.; Zhang, K. Entity embedding-based anomaly detection for heterogeneous categorical events. In Proceedings of the IJCAI International Joint Conference on Artificial Intelligence, New York, NY, USA, 9–15 July 2016.
26. Hu, Z.; Huang, P.; Deng, Y.; Gao, Y.; Xing, E. Entity Hierarchy Embedding. In Proceedings of the Association for Computational Linguistics 2015 (ACL 2015), Beijing, China, 26–31 July 2015; pp. 1292–1300.
27. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Distributed Representations of Words and Phrases and Their Compositionality. *Proc. Adv. Neural Inf. Process. Syst.* **2013**, *2*, 3111–3119. [CrossRef]
28. Mikolov, T.; Corrado, G.; Chen, K.; Dean, J. Efficient Estimation of Word Representations in Vector Space. In Proceedings of the International Conference on Learning Representations (ICLR 2013), Scottsdale, Arizona, 2–4 May 2013; pp. 1–12.
29. Mikolov, T.; Yih, W.-T.; Zweig, G. Linguistic Regularities in Continuous Space Word Representations. In Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013), Atlanta, GA, USA, 9–14 June 2013. [CrossRef]
30. Seok, M.; Song, H.J.; Park, C.Y.; Kim, J.D.; Kim, Y. Named entity recognition using word embedding as a feature. *In. J. Softw. Eng. Its Appl.* **2016**, *10*, 93–104. [CrossRef]
31. Ganguly, D.; Roy, D.; Mitra, M.; Jones, G.J.F. 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—SIGIR'15, Santiago, Chile, 9–13 August 2015. [CrossRef]
32. Zhou, G.; He, T.; Zhao, J.; Hu, P. Learning Continuous Word Embedding with Metadata for Question Retrieval in Community Question Answering. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Beijing, China, 26–31 July 2015; Volume 1. [CrossRef]
33. Zhou, H.; Zouaq, A.; Inkpen, D. DBpedia entity type detection using entity embeddings and N-gram models. In Proceedings of the 8th International Conference on Knowledge Engineering and the Semantic Web (KESW 2017), Szczecin, Poland, 8–10 November 2017. [CrossRef]

34. Paulheim, H. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web* **2017**. [CrossRef]
35. Färber, M.; Ell, B.; Menne, C.; Rettinger, A. A Comparative Survey of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO. *Semantic Web*, 31 July 2015.
36. Zaveri, A.; Kontokostas, D.; Sherif, M.A.; Bühmann, L.; Morsey, M.; Auer, S.; Lehmann, J. User-driven quality evaluation of DBpedia. In Proceedings of the 9th International Conference on Semantic Systems–I-SEMANTICS'13, Graz, Austria, 4–6 September 2013. [CrossRef]
37. Font, L.; Zouaq, A.; Gagnon, M. Assessing and Improving Domain Knowledge Representation in DBpedia. *Open J. Semant. Web* **2017**, *4*, 1–19.
38. Lehmann, J.; Bühmann, L. ORE—A tool for repairing and enriching knowledge bases. In Proceedings of the 9th International Semantic Web Conference (ISWC 2010), Shanghai, China, 7–11 November 2010. [CrossRef]
39. Gangemi, A.; Nuzzolese, A.; Presutti, V.; Draicchio, F.; Musetti, A.; Ciancarini, P. Automatic Typing of DBpedia Entities. In Proceedings of the Semantic Web—ISWC 11th International Semantic Web Conference, Boston, MA, USA, 11–15 November 2012; Volume 7649, pp. 65–81. [CrossRef]
40. Haidar-Ahmad, L.; Font, L.; Zouaq, A.; Gagnon, M. Entity typing and linking using SPARQL patterns and DBpedia. In Proceedings of the Third SemWebEval Challenge at ESWC 2016, Heraklion, Greece, 29 May–2 June 2016. [CrossRef]
41. Paulheim, H.; Bizer, C. Type inference on noisy RDF data. In Proceedings of the 12th International Semantic Web Conference, Sydney, Australia, 21–25 October 2013. [CrossRef]
42. Paulheim, H.; Bizer, C. Improving the Quality of Linked Data Using Statistical Distributions. *Int. J. Semant. Web Inf. Syst.* **2014**. [CrossRef]
43. Röder, M.; Usbeck, R.; Speck, R.; Ngonga Ngomo, A.C. CETUS—A baseline approach to type extraction. In Proceedings of the Second SemWebEval Challenge at ESWC 2015, Portorož, Slovenia, 31 May–4 June 2015. [CrossRef]
44. Van Erp, M.; Vossen, P. Entity typing using distributional semantics and DBpedia. In Proceedings of the SWC 2016 International Workshops: KEKI and NLP&DBpedia, Kobe, Japan, 17–21 October 2017. [CrossRef]
45. Debattista, J.; Lange, C.; Auer, S. A preliminary investigation towards improving linked data quality using distance-based outlier detection. In Proceedings of the 6th Joint International Semantic Technology Conference (JIST 2016), Singapore, 2–4 November 2016. [CrossRef]
46. Abdi, H.; Williams, L.J. Principal component analysis. *Wiley Interdiscip. Rev. Comput. Stat.* **2010**, *2*, 433–459. [CrossRef]
47. Genuer, R.; Poggi, J.M.; Tuleau-Malot, C. Variable selection using random forests. *Pattern Recognit. Lett.* **2010**, *31*, 2225–2236. [CrossRef]
48. Genuer, R.; Poggi, J.-M.; Tuleau-Malot, C. VSURF: An R Package for Variable Selection Using Random Forests. *R J.* **2015**, *7*, 19–33.
49. Pan, X.Y.; Shen, H.B. Robust prediction of B-factor profile from sequence using two-stage SVR based on random forest feature selection. *Protein Pept. Lett.* **2009**, *16*, 1447–1454. [CrossRef] [PubMed]
50. Rogers, J.; Gunn, S. Identifying feature relevance using a random forest. In Proceedings of the Statistical and Optimization Perspectives Workshop "Subspace, Latent Structure and Feature Selection" (SLSFS 2005), Bohinj, Slovenia, 23–25 February 2006. [CrossRef]
51. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-Learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
52. Grover, A.; Leskovec, J. node2vec. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining—KDD'16, San Francisco, CA, USA, 13–17 August 2016. [CrossRef]
53. Pennington, J.; Socher, R.; Manning, C. Glove: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014. [CrossRef]