Detecting and correcting typing errors in open-domain knowledge graphs using semantic representation of entities

by

Daniel Durães Caminhas

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Large and accurate Knowledge Graphs (KGs) are often used as a source of structured knowledge in many natural language processing (NLP) tasks, including question-answering systems, conversational agents, information integration, named entity recognition, document ranking, among others.

Various approaches for creating and updating KGs exist, each with its own advantages and disadvantages. Manually curated KGs can be very accurate, but require too much effort and tend to be small and domain-specific. Larger cross-domain KGs can be created automatically from unstructured or semi-structured data, but even the best methods today are error-prone. Given this trade-off between completeness and correctness, researchers have been attempting to refine KGs after they have been constructed, by adding missing knowledge or finding and correcting erroneous information.

This thesis proposes a fully automatic method for detecting and correcting type assignments in an open-domain KG, provided that the entities in the KG are mentioned in a text corpus. Our approach consists of creating semantic representations (embeddings) of the entities in the KG that take into account how they are mentioned in the corpus and their properties in the KG itself, and using these embeddings as features for machine learning classifiers which are trained to distinguish entities of each type.

To test our solution, we use DBpedia as the KG and Wikipedia as the text corpus, and we perform an extensive retrospective evaluation in which almost 15,000 entity-type pairs were verified by humans. Our results reveal several problems in the DBpedia ontology and led us to the conclusion that our method significantly outperforms alternative solutions for finding erroneous type assignment in knowledge graphs.

Preface

This thesis is an original work by Daniel Duraes Caminhas. Professor Denilson Barbosa has provided guidance for the work presented in this thesis and assisted with the manuscript composition by providing editorial feedback. The data collection described on section 4.1 and the retrospective evaluations described on sections 4.3 and 5.2 were done in collaboration with Daniel Cones and Natalie Hervieux. A preliminary version of the work presented here was published as: D. Caminhas, D. Cones, N. Hervieux, and D. Barbosa. *Detecting and correcting typing errors in dbpedia.* In Proceedings of the International Workshop on Challenges and Experiences From Data Integration to Knowledge Graphs, 4pp. ACM, 2019.

This thesis is dedicated to everyone[1] *who hasn't had a thesis dedicated to them and who cares enough to read it.*

[1]Except those who support racism, homophobia, sexism, classism, religious intolerance, or any other form of discrimination.

Science doesn't always go forwards. It's a bit like doing a Rubik's cube. You sometimes have to make more of a mess with a Rubik's cube before you can get it to go right.

– Jocelyn Bell Burnell, 1943.

Acknowledgements

First and foremost, I would like to thank my supervisor, Denilson Barbosa, for his advice and support throughout my Masters. He constantly encouraged and guided me in the right direction, but always gave me the freedom to explore my own ideas. Always optimistic, on many occasions, he believed in me more than myself. I am sorry for my stubbornness and pessimism, and thanks a lot for your mentorship and patience.

I also would like to thank my family, especially my mother, father, and sister, who taught me the importance of a good education and always supported me in my academic life. Even distant, they are definitely the main reason why I am who I am today.

Last but not least, I want to thank everyone who somehow supported me and contributed to making my life on the past two years much easier and enjoyable, including but not limited to Zinat, Erin, Barbara & Lucas, and Fernando (in no particular order).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Knowledge graphs in the semantic web

The semantic web promotes publishing and querying knowledge on the Web in a semantically structured way. The idea is that the traditional Web, which is made of human-readable of documents, should be extended to a Web of data where not only documents and links between documents, but also entities (e.g., persons or organizations) and relations between entities can be represented in a machine readable format [8]. With the advent of Linked Data, it was proposed to interlink different datasets in the Semantic Web so that the collection of datasets could be viewed as a single large dataset.

In the backbone of the Linked Open Data (LOD) are the Knowledge Graphs (KGs). A KG is a graph database used to store knowledge in a machine-readable format. It allows knowledge to be represented as a semantic network in which entities are interconnected by relations. The entities are represented as the nodes of the graph, while edges represent relations among entities (e.g., *Ottawa is the capital of Canada*). Entities can also have types (e.g., *Canada is a country, Ottawa is a city*). The set of possible types and relations are described in an ontology, which defines their interrelations and restrictions of their usage [32]. The types are organized hierarchically in a taxonomy. Figure 1.1 shows a few types of an ontology organized into a taxonomy. Figure 1.2 shows an example of a few entities (also known as *resources* in the context of KGs)[1] and relations in a KG (DBpedia).

[1]Throughout this document we use the terms *entity(ies)* and *resource(s)* interchangeably.

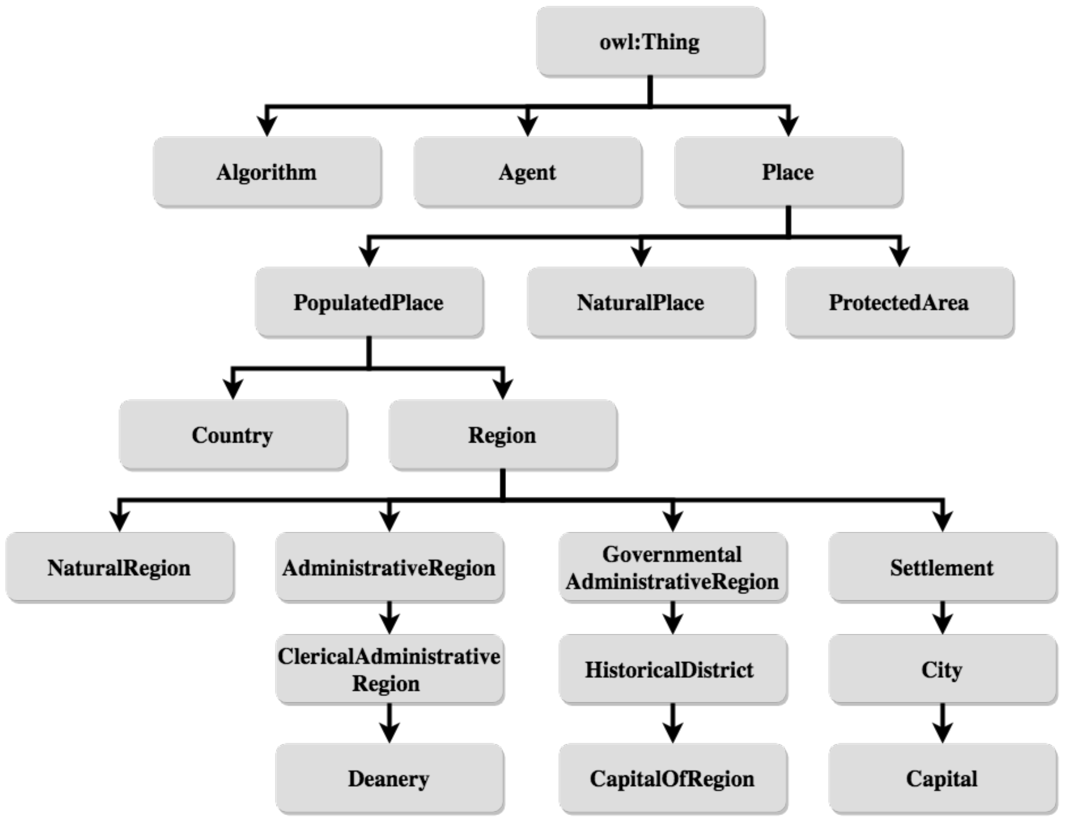


Figure 1.1: Partial representation of the taxonomy of an open KG (DBpedia).

1.2 Motivation

Because KGs can be used to feed intelligent systems with structured knowledge, they became essential for many natural language processing (NLP) applications, such as question-answering systems and conversational agents. Moreover, they have been used to support a wide range of NLP tasks, including but not limited to data integration, named entity recognition, topic detection, document ranking, and distant supervision [32] [19] [26].

Various approaches have been applied to the construction of KGs, which can be manually curated, like *Cyc*[2], edited collaboratively like Freebase [3] and Wikidata [47], or built automatically, such as DBpedia [19] and YAGO [41]. Each approach has its own advantages and disadvantages. Manually curated KGs tend to suffer from limited coverage since it is very difficult to manually collect information about all entities of interest. Meanwhile, KGs constructed

[2]https://www.cyc.com/

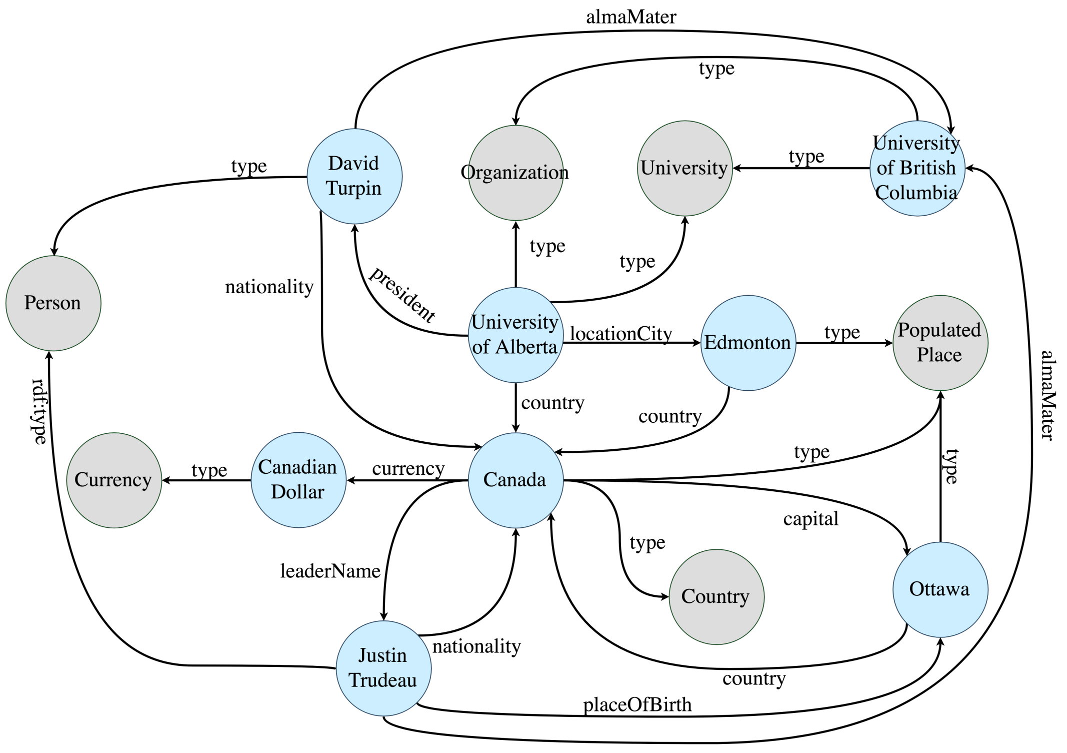


Figure 1.2: Entities and relations from a open knowledge graph (DBpedia). The green nodes represent types of the Ontology, while the blue ones are entities of the knowledge graph. The arrows represent relations between pairs of entities and between an entity and a type in the ontology.

automatically are more likely to contain errors, such as incorrect type information, incorrect relations between entities, incorrect interlinks between different KGs, or incorrect literal values, such as strings, numbers and dates. Thus, in the construction of KGs, there is a trade-off between completeness and correctness [32]. Because of this, researchers have been attempting to refine KGs after they have been constructed, by adding missing knowledge or finding and correcting erroneous information.

In this work, we address the problem of finding and correcting erroneous type assignment in a KG. Although erroneous relation assertions are more frequent than incorrect type assertions, we choose to focus on detecting and correcting typing errors. This is because the type of a resource contains important semantic information which is essential for many NLP tasks and it is one the atomic building blocks of KG [33]. For instance, knowing that *Canada* is a *Country* may help a question answering system answer the question "which

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?resource
WHERE {
  ?resource rdf:type dbo:Person .
  ?resource rdf:type dbo:Company
}
```

Figure 1.3: Disjointness axiom in SPARQL.

countries have English as an official language?". Another motivation for our work is that little research effort has been made to develop methods for finding erroneous type assertions [32], therefore we have more to contribute to this area.

As an example of the type of problem that we are trying to solve, at the time of writing, DBpedia says that the entity `dbr:Egypt`[3] is a `dbo:MusicalArtist`. Similarly, `dbr:United_Nations` and `dbr:European_Union` are, among other things, also classified as a `dbo:Country`, together with another 7,106 entities, which seems unreasonably high[4], even accounting for entities that were historically identified as such. Table 1.1 shows other examples of type inconsistencies that can be identified using disjointness axioms. Disjointness axioms, state which concepts are disjoint with other concepts (e.g., an entity cannot be person and a company, but it can be a person and an artist) [22]. For example, Figure 1.3 shows a SPARQL query that checks the disjointness between types `dbo:Person` and `dbo:Company`.

Besides incorrect type assignments, DBpedia also suffers from the problem of *missing* types for some entities. For example, 27% of the 30935 entities classified as a `dbo:University` are not classified as an `dbo:Organization`.

To test our method we use DBpedia, a large-scale, multilingual, cross-domain KG which is automatically built by extracting structured and semi-structured data from Wikipedia. Because it is automatically constructed, qual-

[3]Throughout the document, we use the customary `dbr:`, `dbo:`, and `dbp:` prefixes to indicate resources (which are entities), ontological predicates (e.g., types), and properties, respectively.

[4]Wikipedia states that the United Nations have 193 members, while there are 8 other entities that are not members but are recognized as countries by at least one UN member.

Table 1.1: Number of inconsistencies in DBpedia identified using disjointness axioms.

Disjointness Axiom	Number of Entities
Person and Company	3892
Company and Record Label	3745
Person and Place	1761
Company and Work	1529
Person and University	1014
Company and Software	1004
Software and Television Show	908
Person and Book	625
Person and Software	601
Person and Aircraft	595
Animal and Plant	200
Person and Country	194
Software and Engine	147
Software and Place	136
Person and City	119
Software and Aircraft	112

ity issues are inherent in the KG. In fact, about 12% of its triples have some type of problem, which may be due to problems with Wikipedia (e.g., inconsistent usage of infoboxes) or bugs and limitations on the DBpedia extraction framework [55]. That does not mean that DBpedia is unsuitable for real-world applications. In fact, it has become one of the major hubs of the LOD ecosystem. For example, it is currently used to enrich web search with facts or suggestion about common-sense information, such as entertainment topics. However, its data quality is probably not enough for developing a medical application [55]. Therefore, if we were able to refine DBpedia, we would expand the range of possible application of the KG while improving the reliability of existing systems that already use it. Although we test our solution in DBpedia, it can also be used to improve arbitrary KGs, regardless of how they are built.

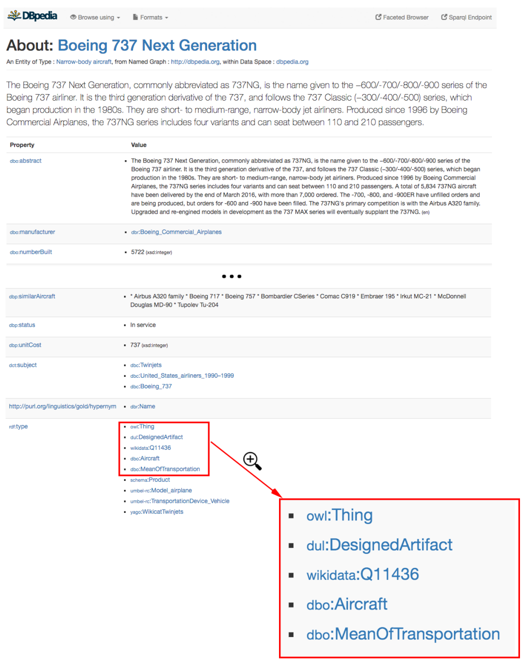
Manually identifying and correcting errors in a large-scale KG is not feasible and does not scale. It has been estimated that more than 3,000 years would be necessary to validate entities in DBpedia with a crowdsourcing approach [34]. Because of that, we decided to develop a solution to automatically refine DBpedia. In contrast to KG construction approaches, which aim to build the graph from scratch, refinement solutions are post-processing operations which assume that a KG has already been built, but it could be improved by adding missing knowledge or identifying and removing errors [32].

Another possible approach for solving this problem would be exploiting other KG interlinks, which is possible because DBpedia is connected with other Linked Datasets by around 50 million RDF links[5]. For instance, we could use Wikidata or the Google's KG to verify the types of the entities in DBpedia. Nevertheless, this approach has a few drawbacks. First of all, the target KG can also be incorrect or incomplete, which is likely to happen with any of the large scale KGs. For example, we found that generally only high-level types are assigned to entities in the Google KG: for instance, most entities of type `dbo:Aircraft` in DBpedia are labeled simply as *Thing* in the Google KG (see Figure 1.4). Moreover, interlinks to other KGs may also be wrong. In fact, it has been shown that about 50% of the interlinks connecting resources in DBpedia to equivalent resources in other KGs are incorrect [10]. Thus, finding equivalent entities in different KGs is a challenging task itself.

Links between KGs at the schema level (i.e., links between ontologies) are even more problematic. The heterogeneity of the ontologies represents an obstacle for automatic tools to determine which types represent overlapping sets of individuals that should be compared [28][8]. This is because concepts that have a strong semantic similarity may not be equivalent [8]. For example, in DBpedia `dbo:Competition` refers to *contest*. It is a subtype of `dbo:event`, with properties such as `dbp:numberOfPeopleAttending`, `dbp:startDate`, and `dbp:followingEvent`. In Wikidata, `competition (Q476300)` makes reference to the *rivalry between organisms, animals, individuals, groups, etc.*

We found that, out of the 685 types in DBpedia ontology, 311 are linked to exactly one type in the Wikidata ontology (one-to-one). Three types in DBpedia have more than one equivalent type in Wikidata (one-to-many), while eight Wikidata types are linked to only 4 types in DBpedia (many-to-one). To

[5]See https://wiki.dbpedia.org/about.



DBpedia Browse using Formats Faceted Browser Sparql Endpoint

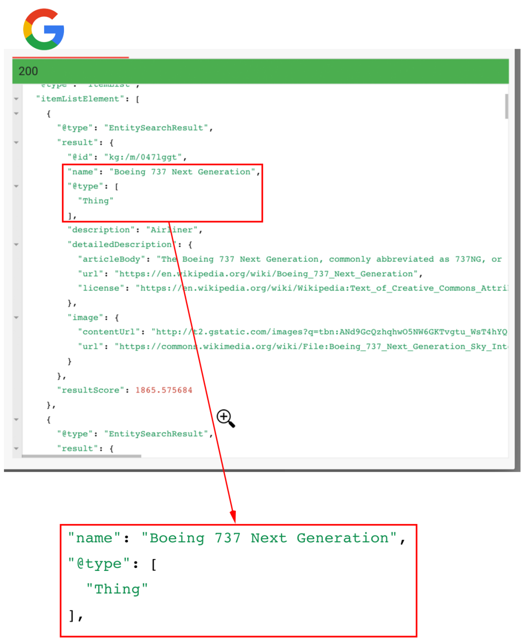
About: Boeing 737 Next Generation

An Entity of Type : Narrow-body aircraft, from Named Graph : http://dbpedia.org, within Data Space : dbpedia.org

The Boeing 737 Next Generation, commonly abbreviated as 737NG, is the name given to the –600/-700/-800/-900 series of the Boeing 737 airliner. It is the third generation derivative of the 737, and follows the 737 Classic (–300/-400/-500) series, which began production in the 1980s. They are short- to medium-range, narrow-body jet airliners. Produced since 1996 by Boeing Commercial Airplanes, the 737NG series includes four variants and can seat between 110 and 210 passengers.

Property	Value
dbo:abstract	The Boeing 737 Next Generation, commonly abbreviated as 737NG, is the name given to the –600/-700/-800/-900 series of the Boeing 737 airliner. It is the third generation derivative of the 737, and follows the 737 Classic (–300/-400/-500) series, which began production in the 1980s. They are short- to medium-range, narrow-body jet airliners. Produced since 1996 by Boeing Commercial Airplanes, the 737NG series includes four variants and can seat between 110 and 210 passengers. A total of 5,834 737NG aircraft have been delivered by the end of March 2016, with more than 7,000 ordered. The -700, -800, and -900ER have unfilled orders and are being produced, but orders for -600 and -900 have been filled. The 737NG's primary competition is with the Airbus A320 family. Upgraded and re-engined models in development as the 737 MAX series will eventually supplant the 737NG. (en)
dbo:manufacturer	dbr:Boeing_Commercial_Airplanes
dbo:numberBuilt	5722 (xsd:integer)
• • •	
dbp:similarAircraft	* Airbus A320 family * Boeing 717 * Boeing 757 * Bombardier CSeries * Comac C919 * Embraer 195 * Irkut MC-21 * McDonnell Douglas MD-90 * Tupolev Tu-204
dbp:status	In service
dbp:unitCost	737 (xsd:integer)
dct:subject	dbc:Twinjets dbc:United_States_airliners_1990–1999 dbc:Boeing_737
http://purl.org/linguistics/gold/hypernym	dbr:Name
rdf:type	owl:Thing dul:DesignedArtifact wikidata:Q11436 dbo:Aircraft dbo:MeanOfTransportation schema:Product umbel-rc:Model_airplane umbel-rc:TransportationDevice_Vehicle yago:WikicatTwinjets

- owl:Thing
- dul:DesignedArtifact
- wikidata:Q11436
- dbo:Aircraft
- dbo:MeanOfTransportation



```
200
  "itemListElement": [
    {
      "@type": "EntitySearchResult",
      "result": {
        "@id": "kg:/m/047lggt",
        "name": "Boeing 737 Next Generation",
        "@type": [
          "Thing"
        ],
        "description": "Airliner",
        "detailedDescription": {
          "articleBody": "The Boeing 737 Next Generation, commonly abbreviated as 737NG, or
          "url": "https://en.wikipedia.org/wiki/Boeing_737_Next_Generation",
          "license": "https://en.wikipedia.org/wiki/Wikipedia:Text_of_Creative_Commons_Attri
        },
        "image": {
          "contentUrl": "http://t2.gstatic.com/images?q=tbn:ANd9GcQzhqhwO5NW6GKTvgtu_WsT4hYQ
          "url": "https://commons.wikimedia.org/wiki/File:Boeing_737_Next_Generation_Sky_Int
        }
      },
      "resultScore": 1865.575684
    },
    {
      "@type": "EntitySearchResult",
      "result": {
```

```
"name": "Boeing 737 Next Generation",
"@type": [
  "Thing"
],
```

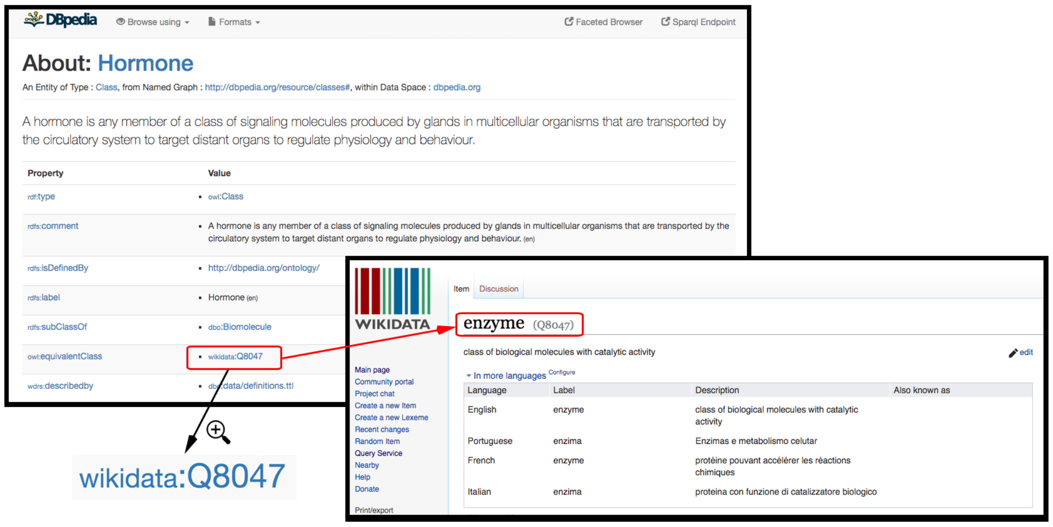
Figure 1.4: The types assigned in the Google's KG are mostly correct but, usually, more high-level types are assigned.

date, links between those two KGs do not exist for 363 types[6].

Even when links exist, sometimes they are wrong or are not precise. For example, the DBpedia type `dbo:Hormone` is linked to the type `enzyme` in Wikidata (Figure 1.5). `dbo:Satellite` is linked to a more specific type of satellite, `artificial satellite`. Similarly, `dbo:Producer` is linked to a lower level type, `film producer`. This means that if we try for example to verify the DBpedia type for a music producer (correctly) labeled as `dbo:Producer` in DBpedia, we will end up flagging the type assigned in DBpedia as incorrect. Therefore, even when it is possible to find the equivalent resource for a entity in a secondary KG, many times a simple comparison of their types is not possible.

In face of these challenges, an alternative approach for KG refining is necessary.

[6]Numbers obtained by querying DBpedia's SPARQL endpoint.



About: Hormone

An Entity of Type : Class, from Named Graph : http://dbpedia.org/resource/classes#, within Data Space : dbpedia.org

A hormone is any member of a class of signaling molecules produced by glands in multicellular organisms that are transported by the circulatory system to target distant organs to regulate physiology and behaviour.

Property	Value
rdf:type	• owl:Class
rdfs:comment	• A hormone is any member of a class of signaling molecules produced by glands in multicellular organisms that are transported by the circulatory system to target distant organs to regulate physiology and behaviour. (en)
rdfs:isDefinedBy	• http://dbpedia.org/ontology/
rdfs:label	• Hormone (en)
rdfs:subClassOf	• dbo:Biomolecule
owl:equivalentClass	• wikidata:Q8047
wdrs:describedby	• dbo:data/definitions.ttl

wikidata:Q8047

enzyme (Q8047)

class of biological molecules with catalytic activity

Language	Label	Description	Also known as
English	enzyme	class of biological molecules with catalytic activity	
Portuguese	enzima	Enzimas e metabolismo celutar	
French	enzyme	protéine pouvant accélérer les réactions chimiques	
Italian	enzima	proteina con funzione di catalizzatore biologico	

Figure 1.5: Example of incorrect mapping between DBpedia and Wikidata.

1.3 Problem definition

This work aims to address erroneous type assignments in KGs using the semantic representations (embeddings) of resources. These embeddings are used as a feature for a set of binary machine learning classifiers (one classifier for each type in the ontology) which are trained to predict the type for the entities. More specifically, we are interested in answering the following research question: *Can semantic representations of entities be used to detect and correct erroneous type assignment in knowledge graphs?*

1.4 Outline

The remaining chapters of this document are organized as follows: Chapter 2 gives a brief overview of a few existing KGs and discuss related works on KG refinement.

In Chapter 3, we describe how we created a semantic representation of the DBpedia resources by deriving from the Wikipedia corpus word2vec-like embeddings and combining them with ontological properties of the entities using PCA [44]. As we will show, this procedure allowed us to combine knowledge from DBpedia and Wikipedia to create vectors that could be used as a feature for machine learning classifiers, which are trained to identify and correct wrong type assignment, as well as assigning types to entities if this information is missing.

In Chapter 4, we explain how the classifiers were trained and tested using a manually curated partial gold standard with 3876 entities of 35 different types of the DBpedia ontology. The performed experiments show that our approach is able to automatically find errors and assign types for DBpedia entities with over 97% accuracy. We also compare the performance of different machine learning algorithms and present the results of a medium-scale evaluation of our proposed approach involving over 350,000 entity-type pairs.

In Chapter 5 we propose an alternative approach for automatically collecting training data for the binary classifiers and then extend our experiments to encompass all types present in the KG. We demonstrate the applicability of our fully automatic solution by automatically checking almost 3,000,000 entity-type pairs in DBpedia. Using a retrospective evaluation, we estimate that the proposed approach is capable of correctly identifying the type for more than 83% of the resources in the KG.

In Chapter 6, Error Analysis, we discuss the situations in which our proposed solution did not perform well and exam the possible causes for the errors. We argue that inconsistencies in the usage of the ontology of the KG creates a challenging environment for automatic (and even manual) error detection.

Finally, the limitations of our method, conclusion, and future work are presented in Chapter 7.

Chapter 2

Related Work

As discussed in Chapter 1, Knowledge graphs (KGs) have become essential for many Natural Language Processing (NLP) applications. However, constructing KGs is a challenging task which involves a trade-off between correctness and completeness [32]. One way to address these shortcomings would be trying to increase the KG coverage or completeness after the construction of the KG, by applying one or more refinement steps. Refinement techniques consider that the KG has already been created and can be improved in a post-processing stage by adding missing knowledge or identifying and removing the erroneous information. Thus, the refinement approaches can be used to improve arbitrary KG, regardless of the technique used for building them. In this chapter, we discussed some of the previous works in KG refining and the similarities and difference between them and our proposed solution.

2.1 Knowledge graphs

In this section, we provide a brief overview of existing open and proprietary KGs. We give special emphasis to DBpedia since it is the KG that we used for testing our solution.

2.1.1 Open Knowledge Graphs

DBpedia

DBpedia [19] is the most popular and prominent KG in the Linked Open Data (LOD) cloud. It contains over 4.5 million of entities, most of them (4.22

million) are assigned to at least one of the 685 types of the DBpedia ontology and are described by 2,795 different properties. This comprises over 3 billion pieces of information (RDF tuples), out of which 580 million were extracted from the English edition of Wikipedia, and the remaining were extracted from other language editions. Table 2.1 shows the number of instances for a few of the types within the ontology[1].

Table 2.1: Number of resources per type in DBpedia

Type	**Instances**
Resource (overall)	4,233,000
Person	1,450,000
Place	735,000
Populated Places	478,000
Creative Work	411,000
Species	251,000
Organizations	241,000
Music Album	123,000
Films	87,000
Companies	58,000
Educational Institutions	49,000
Video Games	19,000
Diseases	6,000

Since it is an open KG (i.e., it is available for everyone on the Web), DBpedia has been extensively used by the research community, and several applications, algorithms, and tools have been built around it. It has also been used in commercial settings. The BBC and the New York Times, for example, have used DBpedia to organize their content [8].

DBpedia is created automatically using an extraction framework which retrieves structured data from Wikipedia, such as the data contained in infoboxes[2], categorization information, geo-coordinates, and links to external web pages. Figure 2.1 shows an overview of the DBpedia extraction frame-

[1]Statistics retrieved from https://wiki.dbpedia.org/services-resources/ontology and https://wiki.dbpedia.org/about on June 2019

[2]Semi-sctructured part of many Wikipedia articles that appears as a table on the side of the page.

work. Wikipedia pages are feed into the system and parsed by into an Abstract Syntax Tree, which is forwarded to the extractors. In total, 24 extractors are used for many different purposes, for instance, to extract label, abstracts, and geographical coordinates. Each extractor consumes an Abstract Syntax Tree and yields a set of RDF statements.

Despite its importance as a general purpose KG, as well as its crucial role for the LOD movement, about 12% of the RDF tuples in DBpedia have some quality issues [55]. There are many possible sources of errors in DBpedia. For example: Infoboxes are the most valuable content for the DBpedia extractors. They are based on a template that specifies a list of attributes that can form the infobox. A wide range of infobox templates are used in Wikipedia (e.g., templates for people, organizations, automobiles, etc.). However, adherence to templates and other editorial practices are hard to enforce, especially over time since the templates themselves also have been changing. Consequently, extracted resources may not be associated to a type in the ontology and heuristics must be used to perform the assignment, making the process non-deterministic.

To address this problem, with the DBpedia 3.2 release, a new infobox extraction method was introduced. Hand-generated mappings from Wikipedia infoboxes to the DBpedia ontology are now used. These mappings, which are also maintained by the DBpedia user community, define fine-grained rules on how to parse infoboxes. Although they addressed the problem of having different infoboxes for the same type, the mappings themselves can also contain errors, which is possibly the reason why erroneous type assignment still exists in the KG.

Freebase

Freebase [3] is a large collaborative KG created by Metaweb Tecjnologies, Inc. in 2007. It consists of data comprised mainly by its community members, although it also contained data harvested from sources such as Wikipedia, NNDB, Fashion Model Directory and MusicBrainz [8].

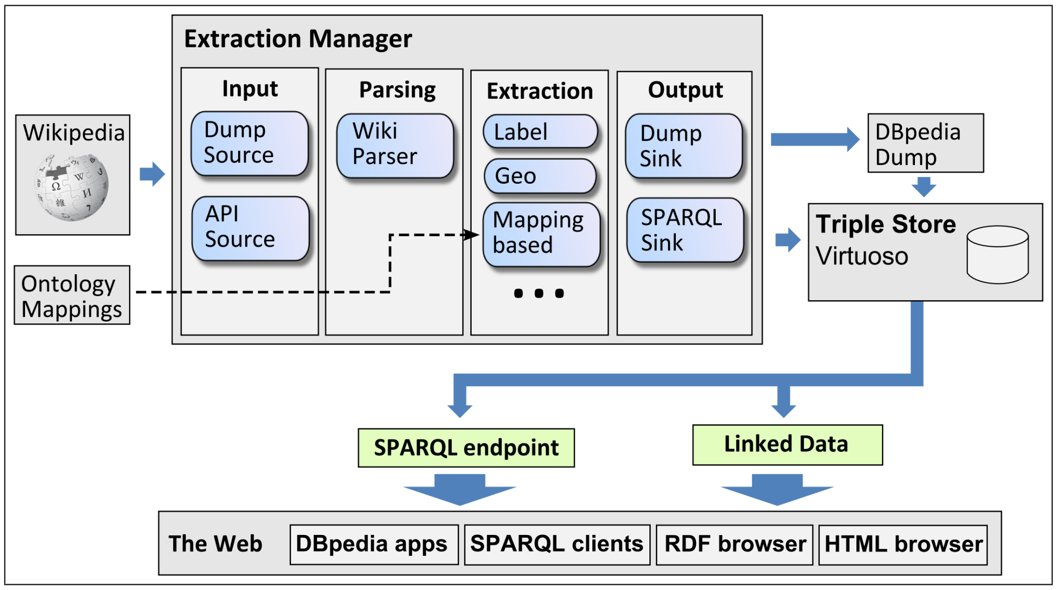


Figure 2.1: Overview of DBpedia extraction framework. Reprinted from Lehmann *et al.* (2012) [19]

In 2010 Freebase was acquired by Google and was frozen in 2016. Google's KG is partially powered by Freebase [37]. The last version of Freebase contains about 50 million entities and 3 billion facts (RDF tuples). Its schema had around 27,000 entity types and 38,000 kinds of relations among entities [32]. Despite no longer being updated, Freebase is still used for research purposes.

Wikidata

Wikidata [47] is a Wikimedia project launched in October 2012 which aims at creating a collaboratively-edited KG. Its goal is to provide data which can be used by any Wikimedia project, including Wikipedia. Like Wikipedia, Wikidata allows users to extend and edit the data and its schema. One particular feature of Wikidata is that it not only store the facts but also their corresponding source so that the validity of facts can be checked.

After Freebase was frozen its data was moved to Wikidata. As of July 2019, Wikidata contained 58 million of entities and over 732 million RDF statements[3].

[3]See https://tools.wmflabs.org/wikidata-todo/stats.php

OpenCyc

The *Cyc* is one of the oldest KGs. The project started in 1984 at Microelectronics and Computer Technology Corporation. Since January 1995, it has been under active development by the Cycorp company. Its goal is to store common-sense rules about how the world works. It was largely created by handwritten axioms and it is estimated that is has taken well over 1,000 person-years of effort to construct it. Although *Cyc* is a proprietary KG, a reduced and open source version is available as *OpenCyc*, which contains about 120,000 entities and 2.5 million facts. Its schema contains roughly 45,000 types and 19,000 relations [32].

YAGO

YAGO (Yet Another Great Ontology) [41] is another open-source KG extracted from Wikipedia. As of July of 2019, YAGO had knowledge of more than 10 million entities and contains more than 120 million facts about these entities. Those entities are organized into an ontology containing 488,469 types which are created by combining the taxonomy of WordNet with the Wikipedia category system. The accuracy of YAGO has been manually evaluated for each of its 77 relations, proving a confirmed accuracy of 95% [23]

YAGO also extracts and combines entities and facts from Wikipedia in different languages. However, unlike DBpedia which creates different interlinks KG for each language edition of Wikipedia, YAGO aims to build a single coherent KG from various Wikipedia language editions, which includes a taxonomy [23].

2.1.2 Proprietary Knowledge Graphs

Google's Knowledge Graph

The Google's KG was introduced in 2012 as a way to enhance search results. It helps understanding the user queries semantically and answering many of the user information needs directly, instead of just retrieving documents from the web [46]. Google is very secretive about their KG and how it is constructed.

They initially reported having more than 500 million entities, as well as more than 3.5 billion facts and relationships between these entities [37]. Less than seven months later, in December of 2012, the KG was already covering 570 million entities and 18 billion facts [6][27].

Apparently, the Google's KG is built using a combination of data automatically extracted from public resources such as Wikipedia and CIA Worlds Factbook and data provided by human workers. It is also partially powered by Freebase [7] [36]. The company have reported that they use query logs of the searches performed by the users to determine the type of entities that they are more interested in including in the KG [46].

Microsoft's Satori

Microsoft Satori is the KG behind Bing[4] and Cortana[5] [9]. Like Google, Microsoft does not disclosure details on how it is built. Statistics about the size of the KG are also not publicly available, although it has been said that as 2012, there were were 300 million entities and 800 million relations [32].

2.2 Quality evaluation of knowledge graphs

Although the usage of KGs to represent ontological knowledge have been significantly increasing, the quality of the data may largely impact their usability. In reality, data quality is commonly conceived as "fitness for use" for a certain application or use case [56]. For example, while a noisy KG can probably be used for enriching Web search with facts or suggestions about common-sense information, such as entertainment topics, it may not be suitable for developing a medical application. While there a large number of carefully curated and high-quality datasets (in particular in the life-sciences domain), there are also datasets extracted from unstructured and semi-structured sources or that are build on crowdsourcing manner, leading to quality issues [55]. Because of that, some research effort has been made to develop methods and metrics to determine the quality of datasets.

[4]https://www.bing.com/

[5]https://www.microsoft.com/en-ca/windows/cortana

In this direction, Zaveri *et al.* (2013) [55] proposed a data quality assessment methodology, which comprises of a semi-automatic process followed by a manual verification to evaluated the quality of DBpedia. They concluded that while a substantial number of problems exists, the overall quality of DBpedia is relatively high, with roughly 11.93% of the tuples presenting some quality issues. In particular, the following problems, as defined by Zaveri *et al.* (2013) [56], were identified as affecting a significant number of resources:

- Accuracy: the degree to which the data correctly represents real-world facts.

- Relevancy: the provision of information which is in accordance with the task at hand and important to the users' queries. Example of extraction of irrelevant information includes: extraction of attributes containing layout information, and image-related information (extraction of an image caption or name of the image is irrelevant in DBpedia as the image is not displayed for any resource).

- Representational-consistency: the degree to which the format and structure of information conform to previously returned information and other datasets.

- Interlinking: the degree to which entities representing the same concept are linked to each other. Links can be to external websites or other datasets. DBpedia, for example, is connected with other Linked Datasets by around 50 million RDF links[6].

This thesis target the first item of this list, accuracy. By removing incorrect type information from the KG, it becomes more suitable for a wider range of applications, while improving the reliability of existing systems that already use it.

[6]https://wiki.dbpedia.org/about

2.3 Approaches for Knowledge graph completion

Approaches for KG completion usually target the task of adding missing entities, assigning types for entities, or adding relations between entities in the KG. The ultimate goal is to improve the overall quality of the graph by adding missing knowledge and, therefore, increasing its coverage. That can be done using information contained in the KG itself (internal methods) as well as data from external sources (external methods).

Internal methods usually use the properties of the entities as features for classification. For example, an entity that contains a property *director* it is likely to be a *movie*. Paulheim *et al.* (2013) [33] explores how the type information can be generated heuristically by exploiting other axioms in a KG (e.g., the probability of an entity being of type *Actor* is high if it is linked to several other entities by a property like *cast*). They use a weighted voting approach to avoid the propagation of errors from single wrong axioms. The authors also proposed to apply this approach for detecting erroneous type assignment on KGs [34]. This method is one of those used by DBpedia to assigned additional type statements to untyped entities [32].

In this work, we also use the properties of the entities, which are encoded as a 300-dimension vector, as features for machine learning classifiers. As we show on Section 3.1.2 these features significantly increase the semantic relatedness of entities of the same type. The idea of using machine learning for entity type recognition is not new. Sleeman *et al.* (2013) [38] proposed to use Support Vector Machines (SVMs) in order to reduce the computational cost of performing coreference resolution, in which entities and their types are identified using contextual information and linguistic-based analysis. The authors exploits interlinks between DBpedia and Freebase and classifies entities in one KG based on properties in the other KG. The problem with this approach is that interlinks between KGs are frequently wrong [10]. Moreover, they constrain the solution to identify only instances of type of person, location and organization.

Using of association rule mining [12] for predicting missing types in DBpedia has also been proposed [31]. Because DBpedia has different type systems (i.e., entities can be linked to types on the DBpedia ontology, YAGO, schema.org, Wikidata, etc.), there is an overlap of types that allows the leaning of axioms, which in turn can be used to assign types to the entities.

Topic modelling has also been used for type prediction. Entities in the KG can be represented as documents. Latent Dirichlet Allocation (LDA) [2] can be applied to these documents for finding topics. By associating topics to the type of the entities in the KG, it is possible to infer the type for an entity without type by detecting the topics for that entity [39].

Nuzzolese *et al.* (2012) proposes an external approach for predicting the types of entities in the KG. They exploit wikilinks (i.e., interlinks connecting Wikipedia articles) to create feature vectors (e.g., based on the categories of the related pages) which are used as features for a k-nearest neighbours (kNN) classifier to predict types for entities in the KG (given that the KG contains links to Wikipedia). Apriosio *et al.* (2013) [1] also uses a kNN classifier from type assignment, but the features are the types of the entities in different languages edtions of DBpedia. Another approach for type assignment consists of applying hypernym extraction techniques on the abstract of the resource in the KG ([30][15]), for example, using the Hearst patterns [11].

A lot of research effort has also been deployed on to predict relations between entities. Most of the approaches rely on external knowledge, focusing on extracting facts from unstructured and structured noisy Web sources [4]. The new facts can be then added to the KG if they are missing. For example, Xu *et al.* (2019) [52] unifiers KG embeddings [49] and relation extraction models for KG completion. In the backbone of their solution is a bi-directional long short term memory (LSTM) network with multiple levels of attention to learn representations of text expressing relations. Knowledge representation machinery nudges the language model to agree with facts in the KG. This allows learning language and knowledge representations jointly. Several other works have also proposed neural methods for extracting relations from text [48][20][57][16].

Among the internal methods for predicting relations in KG is the work of

Socher *et al.* (2013) [40]. They use a tensor neural network to predict the likely truth of additional facts based on existing facts in the KG. For example, if a person is born in a city in Germany, the model can predict that the nationality of that person is German. Zhao *et al.* (2015) [58] also relies exclusively on the information from the KGs. They propose a Pairwise-interaction Differentiated Embeddings model to embed entities and relations in the KG into a lower dimensional space and then predict the possible truth of additional facts to extend the KG.

Association rule mining can also be used as an internal method for predicting relations. Kim *et al.* (2015) [14], for instance, predicts relations between entities in DBpedia using association rules mined from Wikipedia categories, which mainly utilize lexical patterns in category expression and a hierarchy of categories.

2.4 Approaches for error detection on Knowledge graphs

Approaches for error detection in KG can also target erroneous type information, incorrect relationships between entities, incorrect interlinks between different KGs, or incorrect literal values. Like what happens with completion methods, this can be done using only the information contained in the KG itself (internal methods) or can also use data from external data sources (external methods).

Although several methods have been proposed for assigning types to the entities in the KG, methods for finding erroneous type assertions are rare. To the best of our knowledge, Ma *et al.* (2014) [22] was the first attempt to detect type inconsistencies in DBpedia. They proposed an improved approach to learn disjointness axioms using association rule mining. The axioms are learned from DBpedia and tested in DBpedia and Zhishi.me [29]. Although this approach is, in fact, able to identify several inconsistencies, it has a few limitations. First of all, the association rules are learned from DBpedia, which is itself a noisy dataset, therefore, there will always be some wrong axioms.

Secondly, some entities in DBpedia are assigned to a single incorrect type. For example, the only assigned type for `dbr:Nail_polish` is `dbo:Person`, which is wrong. However, since there is no other type associated with this entity, there is no axiom capable of identifying this error, because each rule involves two types.

The usage of entity embedding for erroneous type detection has also been attempted before. Zhou *et al.* (2017) [59] used binary clustering to separate two groups of entities for each type of interest: the entities belonging to the type category and the ones that do not belong to the type. For example, they create one cluster with entities of the type `Country` and another cluster with entities that are `not a Country`. However, since the `not a Country` cluster can contain entities of many different types, they are not able to correct wrong type assignment. Another important difference between our work the one presented Zhou *et al.* resides on the creation of the embedding. While they use only Wikipedia for training the embeddings, our embedding also comprises information about the entities properties in DBpedia (as we explain in Section 3.1). Moreover, Zhou *et al.* use the types assignments in DBpedia itself to create the training and testing datasets. They query a public DBpedia SPARQL endpoint to select, for each DBpedia type, entities as positive examples of that type. Negative examples are chosen from a random selection of instances from all the remaining types. We argue that this approach will create a noisy training and testing dataset, since as we discussed, many entity types in DBpedia are wrong, and that is exactly the problem that we are attempting to solve. In this work, we use a manually curated partial gold standard for training and testing our classifiers. We then proposed an automatic approach for selecting only a small number of reference entities for each type. Because of that, the direct comparison between our solution and theirs is not meaningful.

Methods for finding other erroneous relations in KGs usually combine multiple techniques. If the ontology of the KG is rich enough, so that the possible types of nodes and edges in the KG can be explored, reasoning can be used for error detection. Reasoning is a field of study in the artificial intelligence community which deals with automatically deriving proofs for theorems, and for uncovering contradictions in a set of axioms [32]. However, because of inconsistencies in the ontology of large KG, such as DBpedia, approaches exploiting reasoning are typically combined with other methods, such as statistical methods ([13][45][34]) and association rule mining ([17]).

One of the few external approaches for finding erroneous relations is DeFacto [18]. They present an algorithm for validating RDF tuples by finding confirming sources for it on the web, using a search engine. Statements with no or only very few web pages supporting the corresponding sentences are then assigned a low confidence score.

Outlier detection methods are frequently employed for detecting incorrect literal values. The problem with this approach is that outlier detection does not necessarily identify errors, but sometimes natural outliers (such as the population of very large cities). However, it has been shown that most of the outliers are in fact errors [50]. An alternative approach that explores the interlinks in the KG have also been proposed [21]. By comparing the same entity in different KG, the facts in one KG are assumed to be wrong if multiple other sources have a consensus for a conflicting fact.

Chapter 3

Method

As mentioned in Chapter 1, our goal is to verify if the semantic representation (embeddings) of the entities in a Knowledge Graph can be used to detect and correct erroneous type information. We hypothesize that these distributed representations of the resources can be used as features in machine learning models trained to detect incorrect type assignment.

In this chapter, we first explain how these embeddings, which we call resource2vec, can be created by concatenating two other embeddings: Wikipedia-2vec [53] and DBpedia2vec embeddings. Then, we present the machine learning algorithms that we chose to test our hypothesis.

3.1 Representing DBpedia resources

3.1.1 wikipedia2vec Embeddings

The wikipedia2vec embeddings are embeddings that represent Wikipedia entities (Wikipedia articles). They are created using Wikipedia2Vec [53], a tool that allows learning embeddings of words and entities simultaneously from a text corpus, and places similar words and entities close to one another in a continuous vector space. To learn word embeddings, Wikipedia2vec implements the conventional skip-gram model, which learns vector representation for words using a neural network to predict neighbouring words given each word in a text contained on a Wikipedia page [25]. The extension proposed by Yamada et al. (2016) [54] is used to learn the embeddings of entities. Learning these embeddings involves optimizing three submodels, which are illustrated

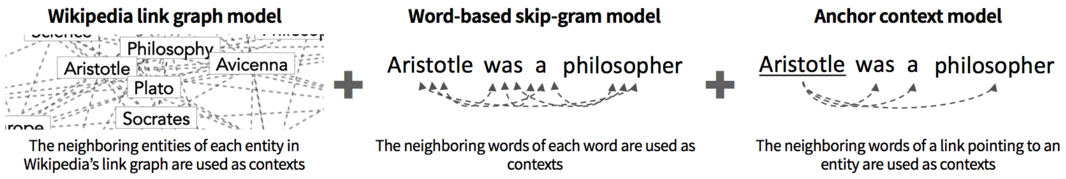

Wikipedia link graph model
Philosophy
Aristotle
Avicenna
Plato
Socrates
The neighboring entities of each entity in Wikipedia's link graph are used as contexts
Word-based skip-gram model
Aristotle was a philosopher
The neighboring words of each word are used as contexts
Anchor context model
Aristotle was a philosopher
The neighboring words of a link pointing to an entity are used as contexts

Figure 3.1: Submodels optimized by Wikipedia2Vec to learn embeddings. Reprinted from Yamada *et al.* (2018) [53].

by Figure 3.1:

- Wikipedia link graph model, which learns entity embeddings by predicting neighbouring entities in link graph of Wikipedia, an undirected graph which nodes are entities and edges represent links between entities. A link between entities (interwiki link) exists if one Wikipedia page is linked to the other.
- Word-based skip-gram model, which learns word embeddings by predicting neighbouring words given each word in a text contained on a Wikipedia page.
- Anchor context model, which aims to place similar words and entities near one another in the vector space, and to create interactions between embeddings of words and those of entities. Here, we obtain referent entities and their neighbouring words from links contained in a Wikipedia page, and the model learns embeddings by predicting neighbouring words given each entity.

The problem with this approach is that Wikipedia editorial guidelines[1] instructs its contributors to avoid overlinking, by creating an interwiki link only when the name first occurs in the page. On top of that, only the most relevant entities on the page should be linked, frequently excluding the names of subjects which most readers will be at least somewhat familiar with. Therefore, many references to entity names do not appear as links in Wikipedia, limiting

[1]https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Linking

the ability of Wikipedia2vec to learn meaningful embeddings. To address this problem, the tool provides a feature that automatically generates links [53].

In an attempt to obtain high-quality embeddings, we trained Wikipedia2vec embeddings with 500 dimensions (i.e., on Figure 3.2 $n1 = 500$) using the Wikipedia dump extracted on Feb. 2019, using 10 iterations over the articles, a windows size of 10, and a minimum number of 10 occurrences for words and 5 occurrences for entities.

3.1.2 DBpedia2vec Embeddings

In order to better represent entities in DBpedia, we introduce a different embedding scheme that takes into account the *properties* of these entities in the KG. The intuition behind these embeddings, which we call DBpedia2vec, is that most resources of the same type share the same properties. For example, countries usually have properties such as `dbo:areaTotal`, `dbo:capital`, and `dbo:largestCity`, while companies are more likely to have properties like `dbo:headquarter`, `dbo:numberOfEmployees`, and `dbo:revenue`. By including the properties of the resources on their representation, we expect to increase the semantic relatedness between resources of the same type.

To create DBpedia2vec embeddings, we first create a list of all distinct properties existing in DBpedia. The name of each property p is stored in the t-th dimension of a k-dimension vector, where $k = 3480$, the number of distinct properties in DBpedia. For each DBpedia resource, i, an k-dimension one-hot encoding vector is created in which the t-th dimension will be equal to 1 if the property p exists for the resource i, otherwise, it will be 0. This allows us to represent the presence or absence of each property in each resource. Table 3.1 shows a few examples (in reality the table would have 3481 columns, most of them were omitted for a better visualization). To increase the type differentiation power of the DBpedia2vec embeddings, we ignore properties that are common across all resources in DBpedia, such as `dbo:wikiPageID`, `dbo:wikiPageWikiLink`, `dbo:abstract`, and `dbo:sameAs`.

The problem with the one-hot encoding representation is that it leads to very sparse and high dimension vectors, since most of the resources will have a

Table 3.1: Examples of one-hot encodings of DBpedia entities based on their properties.

	`dbo:areaTotal`	`dbo:capital`	`dbo:currency`	`dbo:populationTotal`	`dbo:birthDate`	`dbo:birthPlace`	`dbo:spouse`	`dbo:country`	`dbo:president`	`dbp:provost`	...
`dbr:Canada`	1	1	1	1	0	0	0	0	0	0	
`dbr:France`	1	1	1	1	0	0	0	0	1	0	
`dbr:Barack_Obama`	0	0	0	0	1	1	1	0	0	0	
`dbr:Paris`	1	0	0	1	0	0	0	1	0	0	
`dbr:University_of_Alberta`	0	0	0	0	0	0	0	1	1	1	
⋮											

very small subset of the existing DBpedia properties. To solve this problem, we apply Probabilistic Principal component analysis (PCA)[44] to linearly reduce the dimensionality of the embeddings using Singular Value Decomposition of the data. In this way, we are able to project the 3480-dimension embeddings to a lower dimensional and continuous space with 300 dimensions (i.e., in Figure 3.2 $n_2 = 300$). The number of dimension for these embeddings was determined empirically.

Table 3.2 shows examples of cosine similarities between pairs of entities represented using DBpedia2vec. It is possible to notice that pairs of resources of the same type are being placed close to each other (higher cosine similarity), while resources of different types have lower similarity between them.

3.1.3 resource2vec Embeddings

To create a distributed representation for DBpedia entities we concatenate Wikipedia2vec with DBpedia2vec embeddings, as illustrated by Figure 3.2.

Table 3.3 shows examples of cosine similarities for pairs of entities of the same type. Note how the final resource2vec embeddings in fact place entities belonging to the same type closer to each other, in comparison to the wikipedia2vec embeddings alone. One may argue that because the similarity

Table 3.2: Similarity between pairs of entities represented using DBpedia2vec embeddings.

	United States	**Stanford University**	**George W. Bush**	**Vancouver**	**Microsoft**
Canada	**0.9792**	0.0669	0.0321	0.3454	0.0778
University of Alberta	0.1019	**0.8515**	0.1535	0.2896	0.2189
Barack Obama	0.0241	0.0397	**0.7511**	0.0115	0.0463
Ottawa	0.3489	0.1996	0.0948	**0.9721**	0.1326
Apple Inc.	0.0805	0.1713	0.0533	0.1133	**0.9438**

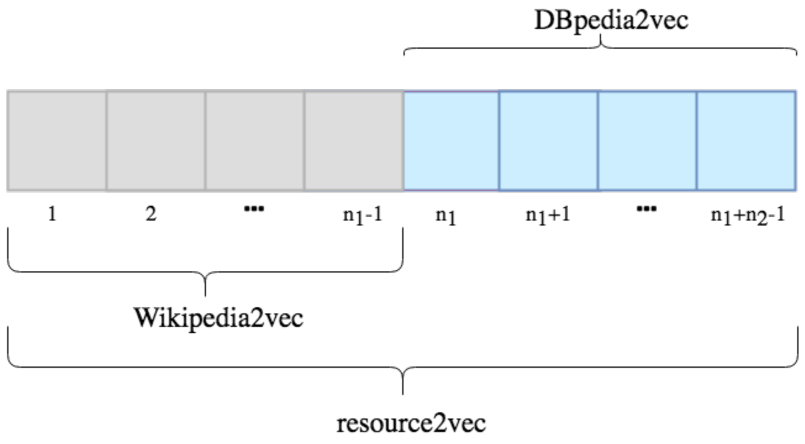


Figure 3.2: Wikipedia2vec embeddings are concatenated with DBpedia2vec embeddings to create resource2vec embddings.

between pairs of entities of the same type is higher using DBpedia2vec embeddings than using the resource2vec embeddings, we should train the classifiers using the DBpedia2vec embeddings only. However, while DBpedia2vec increase the semantic relatedness between entities of the same type, they would not be enough to represent the resources alone, since entities of different types may share some properties (for example if they have a common ancestor in DBpedia ontology). Moreover, multiple resources of the same type may have identical DBpedia2vec embeddings, which would make the classifiers overfit. We empirically verified that despise the higher similarity between entities of the same type using DBpedia2vec embeddings, classifiers trained using only the DBpedia2vec embeddings have lower performance than classifiers trained using the concatenated resource2vec embeddings.

Table 3.3: Cosine similarity between pairs of entities of the same type using different embeddings.

	Wikipedia2vec	**DBpedia2vec**	**resource2vec**
Canada and United States	0.2920	0.9792	0.5900
University of Alberta and Stanford University	0.3352	0.8515	0.4811
Barack Obama and George W. Bush	0.5225	0.7511	0.6011
Ottawa and Vancouver	0.4102	0.9721	0.6545
Apple Inc. and Microsoft	0.4422	0.9438	0.6002

3.2 Identifying and correcting erroneous types

To verify if the type assigned to a given resource is correct, we rely on a set of binary classifiers. Each classifier is trained to distinguish resources from a particular type from the resources of all other types. Thus, one classifier must be trained for each type of DBpedia ontology.

The resource2vec embedding of the entity is the only feature required for the classification. A classifier for a particular type c_i is created using resource2vec embeddings of entities with type c_i as positive examples and resource2vec embeddings of randomly selected resources from all other types as negative examples.

This approach allows us not only to identify erroneous type assignments but also to assign the correct type to any DBpedia resource for which the resource2vec embedding is created, even if no type has been assigned yet in DBpedia. An example is shown in Figure 3.3. The resource2vec embedding for `dbr:Canada` is fed to all classifiers. The classifiers which were trained to identify entities of the types that countries typically belongs to (e.g., `dbo:Place`, `dbo:Location`, `dbo:Country`, and `dbo:PopulatedPlace`) are expected to output *True*, while all other classifiers should output *False*.

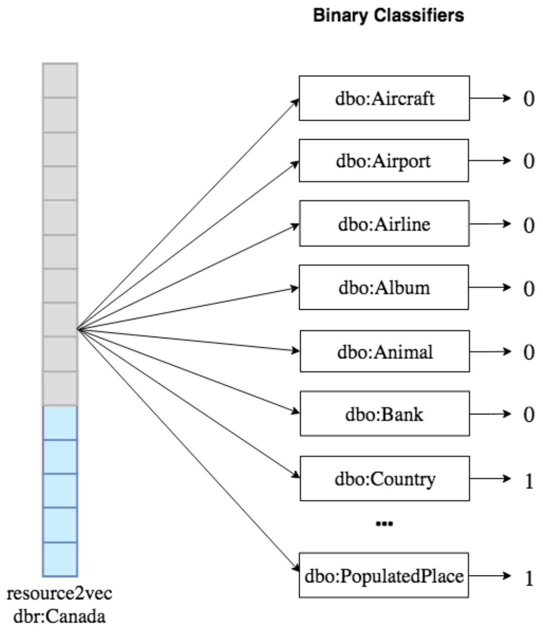


Figure 3.3: Example of type check for the resource Canada

We decided to start our experiments with popular algorithms for binary classification, namely Naive Bayes, K-nearest neighbours (K-NN), and Nearest Centroids. These algorithms were chosen because of their simplicity and because they required a small training dataset, unlike more sophisticated techniques such as neural networks. As we discuss in Section 4.1, collecting training data in our case is a challenging and costly task, therefore if these simpler algorithms perform well the overhead of the solution can be significantly reduced. The next subsections provide a brief overview of these three algorithms. Later on Section 4.2 we compare the performance among them.

3.2.1 Naive Bayes

Naive Bayes is a probabilistic supervised classifier technique based on Bayes' Theorem with an assumption of independence among features. Despite this apparently oversimplified assumption, it is highly scalable and performs well in many real-world problems. Furthermore, when this assumption holds, the algorithm performs better than other models and only requires a small number of training data to estimate its parameters.

Naive Bayes is a conditional probability model. Given a class variable y and a dependent feature vector $\mathbf{x} = (x_1, \ldots, x_n)$, Bayes' theorem states that:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)} \tag{3.1}$$

Because the features are assumed to be independent,

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y).$$

Thus, the equation 3.1 can be rewrite as:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)} \tag{3.2}$$

In practice, there is interest only in the numerator of that fraction, because $P(x_1, \ldots, x_n)$ is constant for the input vector $\mathbf{x}$ being classified. Therefore the denominator can be removed and a proportionality can be introduced. The numerator is equivalent to the joint probability model $p(y, x_1, \ldots, x_n)$ which can be rewritten as products of sequences, using the chain rule for repeated applications of the definition of conditional probability:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \tag{3.3}$$

We can create a classifier combining this model with a decision rule. One common rule is to pick the hypothesis that is most probable (*Maximum A Posteriori*). The corresponding classifier, a Bayes classifier, is the function that assigns a class label $\hat{y}$ for some possible outcomes as follows:

$$\hat{y} = \arg\max_{y} \hat{P}(y) \prod_{i=1}^{n} \hat{P}(x_i \mid y), \tag{3.4}$$

We write $\hat{P}$ for P because the true value of the parameters $P(y)$ and $P(x_i \mid y)$ is unknown, but they can be estimated from the training data.

Because the resource2vec embeddings used as features take up continuous values, we assume that these values are sampled from a Gaussian, and the conditional probability formula changes to:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\omega_y^2}} \exp\left(-\frac{(x_i - \mu_i)^2}{2\omega_y^2}\right) \tag{3.5}$$

In reality, the conditional independence assumption does not hold for the majority or the problems and definitely does not hold for the resource2vec embeddings (e.g.,, a resource with a `dbp:Currency` property is likely to also have a `dbp:Capital` property). However, Naive Bayes models perform well despite the violation of this assumption. That is because even though the probabilities estimated by Naive Bayes can be of low quality, the winning class usually has a probability much higher than the other ones. Since the classification decision is based sonly on which class gets the highest score, regardless of how accurate the estimates are, the correct prediction can be done even with incorrect probabilities [24].

The main strength of Naive Bayes is its efficiency. Training and classification can be accomplished with one pass over the data. For a n-dimension feature vector, a training with s samples will cost $O(ns)$, because all it needs to do is compute the frequency of every feature value $x_1, x_2, \cdots, x_n$ for each class. During the classification, we need to retrieve n feature values for each class, thus, if c is the number of classes, the algorithm's time complexity for testing is $O(nc)$.

3.2.2 Nearest Centroids

The nearest centroid classifier represents objects as points in a high-dimension space and represents each class by its centroid. The classification consists in assigning the class of the nearest centroid to test samples [42]. The training examples are the resource embeddings vectors, each with a class label. The training phase of the algorithm consists in storing the feature vectors and class labels of the training samples. The centroid of a class c is computed as the vector average or center of mass of its members [24]:

$$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d), \tag{3.6}$$

where D_c is the set of entities in the dataset whose class is c. Three example centroids are shown as solid circles in Figure 3.4

The boundaries between types are the set of points with equal distance from the centroids. This set of points is always a line or a hyperplane in a higher-dimensional space. The classification of a new point is based on the region it falls into. This is equivalent to assign to the new point the class of the nearest centroid $\vec{\mu}(c)$. For example, the start on figure 3.4 is located on the *Aircraft* space, therefore it will be assigned to the type *Aicraft*. Distances are typically computed as Euclidean distances, but other measures are also possible.

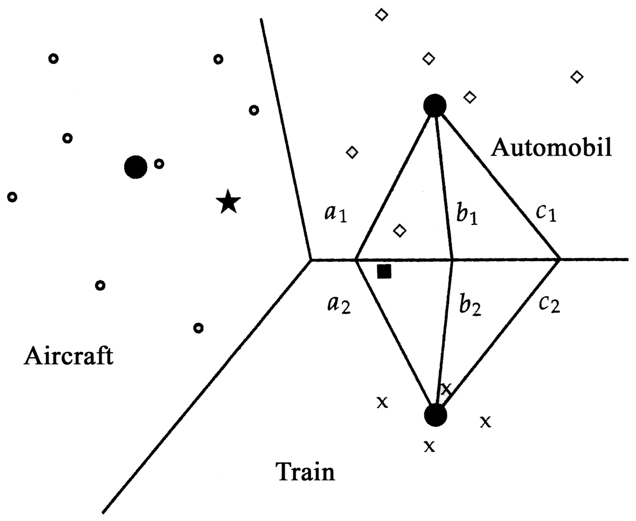


Figure 3.4: Example of nearest neighbor classification. Adapted from Manning *et al.*(2010) [24]

3.2.3 k-nearest neighbors

Unlike the nearest centroids classifiers, the k-nearest neighbour (kNN) classifier determines the decision boundary locally (i.e., using the samples themselves, not the centroid of the class). For $k = 1$, each sample is assigned to the class of its closest neighbour. For other values of k, samples are classified based on the types of the majority of its k nearest neighbours [24].

The only way to determine the best value for K is through experimentation. Small values can be noisy and subject to the effects of outliers, while larger

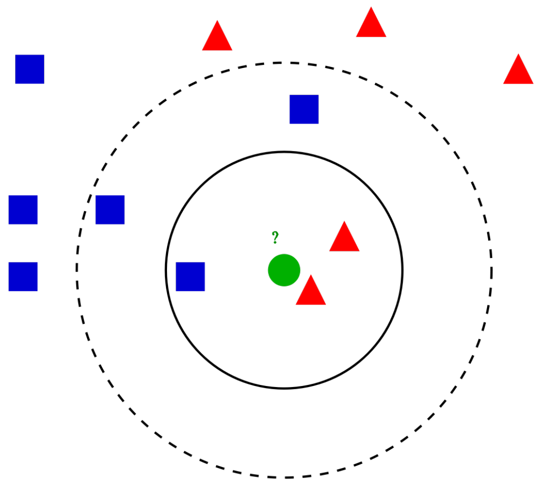


Figure 3.5: Example of k-NN classification [51].

values will have smoother decision boundaries which mean lower variance but increased bias. In binary classification problems like ours, it is helpful to choose k to be an odd number as this avoids ties. In this work, we used 5-fold cross-validation to find the best value of k, which in our case is $k = 9$.

During the classification, an unlabeled embedding is classified by assigning the label which is most frequent among the k training samples nearest to that query point. A commonly used distance metric for continuous variables is the Euclidean distance, the same used in this work.

An example of k-NN classification is shown in Figure 3.5. If $k = 3$ (solid line circle), the test sample (green dot) will be assigned to the class of red triangles because there are 2 triangles and only 1 square between the top 3 nearest neighbours. If $k = 5$ (dashed line circle), the sample will is assigned to the blue squares [51].

The space complexity for s training samples with n-dimension feature vectors is $O(ns)$. During the classification, we need to compare the test point to every data point in the training set. Therefore, its time complexity for testing is $O(ns)$.

Like Nearest Centroids, the training phase of the algorithm consists in storing the feature vectors and class labels of the training samples. Thus, we can say that its time complexity for training is $O(1)$ or even inexistent. Because of this, we use the terms ***reference entities*** or ***representative entities*** to make reference to the data samples used for performing classification using these algorithms. The term ***dataset*** is still used to refer to the collections of all reference entities.

Chapter 4

Supervised approach for error detection

Supervised classification is often considered to be an upper-bound on what can be achieved in large-scale classification problems. Because of this, our first attempt to detect and correct erroneous type assignments is using a supervised solution. In this Chapter, we describe how we do that.

4.1 Dataset

For training and testing the classifiers to detect erroneous type assignments, we consider different methods for obtaining a dataset (i.e., to select the reference entities). Each of them has its own advantages and disadvantages. The first option would be using DBpedia itself as a silver standard. However, since there is a significant amount of noise in the knowledge graph (KG), otherwise error detection would not be necessary, this approach is only suitable for evaluating KG completion, not for error detection since it assumes that the given KG is already of reasonable quality [32]. Zhou *et al.* (2017) [59], for instance, use DBpedia a silver standard for assigning types for entities in the KG.

As an alternative, we could exploit an external KG based on the interlinks. Since DBpedia has interlinks to other KGs, like Freebase (which is incorporated by the Google's KG) and Wikidata, we could retrieve the type for DBpedia resources from another KG to create a dataset for training and testing the classifiers. Although this approach allows us to quickly collect a large number of data samples, as discussed in Section 1.2, the comparison among KGs has important disadvantages:

- The target KG may be incorrect or incomplete.
- The interlink between equivalent resources may be incorrect.
- The ontologies are not aligned.

In face of the challenges of using an external KG for creating a dataset, a suitable approach would be the development of a partial gold standard, manually created by humans, who select and label a subset of the entities in the KG. One of the main advantages of the usage of a partial gold standard is that it can be used to compare different methods (e.g., to compare different machine learning algorithms for training the classifiers). Moreover, gold standards can provide high-quality data. The main disadvantage of this approach is the cost of collecting the data samples, which is relatively high. Because of that, this type of dataset is usually small.

Considering the characteristics of each solution, we decided to create a gold standard to better train and evaluate our classifiers. Our gold standard encompasses the following 35 types from the DBpedia ontology: *Aircraft, Airline, Airport, Album, AmericanFootballPlayer, Animal, Automobile, Bacteria, Bank, Book, Building, City, Country, Currency, Food, Galaxy, HorseTrainer, Language, MilitaryConflict, Murderer, MusicalArtist, MythologicalFigure, Planet, Plant, President, Software, Song , Sport, Swimmer, Theatre, TimePeriod, Train, University, Volcano,* and *Weapon.* We chose these types with the goal of maximizing the diversity of entities while minimizing inter-type overlap (which could potentially confuse our analysis and preliminary conclusions). If the proposed approach works well in this simplified setting, it may be worth scaling the solution to consider all types in DBpedia.

To build the gold standard, three annotators[1] were asked to use any resources at their disposal (e.g., Wikipedia entity lists) to find examples of entities in each of the 35 types. The annotators were instructed to select entities

[1]All annotators were computer science students and were trained for this task.

that would maximize the diversity of the dataset (for example, select cities of different countries and continents, of different sizes). In total, 3876 entities were selected. The number of entities per type varied from 94 (for the types *Sport* and *Software*) to 112 (for the type *Aircraft*). Our gold standard can be downloaded from *https://bit.ly/2FcqQQW*.

4.2 Selecting the best classifier

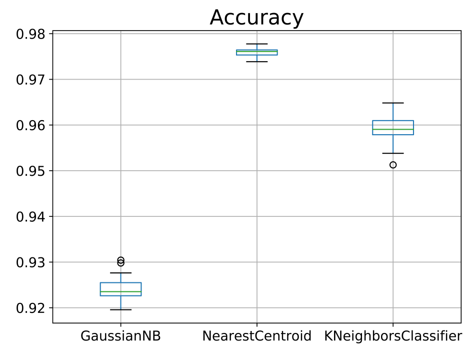
Our first experiment consisted of comparing the three algorithms for binary classification described in Section 3.2: Naive Bayes, K-nearest neighbours (K-NN), and Nearest Centroids. As in all other experiments described in this document we used the 800-dimensions resource2vec embeddings created as described in Section 3.1.

In this experiment, we used the manually curated gold standard to create the dataset to train and evaluate the classifiers. One classifier is trained per type, using as positive samples the entities listed in the gold standard for that type. Negative examples are randomly selected entities of other types. Hyper-parameter tuning for k-NN lead to $k = 9$ and it was performed using 5-fold cross-validation. Table 4.1 shows the preliminary results. The reported values are the average among the 35 classifiers, each of which trained and tested using a 10-fold cross-validation approach. The Nearest Centroid algorithm appears to leads to better classifiers, achieving more than 97% of performance in all metrics, while the Naive Bayes classifiers seem to have the lowest performance.

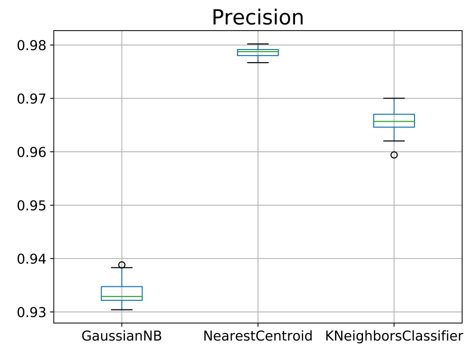
Table 4.1: Comparison between the algorithms used for creating binary classifiers for error detection on type assignment.

Classifier	Accuracy	Precision	Recall	F1-Score
Nearest Centroid	**0.9758**	**0.9785**	**0.9758**	**0.9756**
k-NN	0.9589	0.9654	0.9589	0.9583
Naive Bayes	0.9238	0.9333	0.9238	0.9225

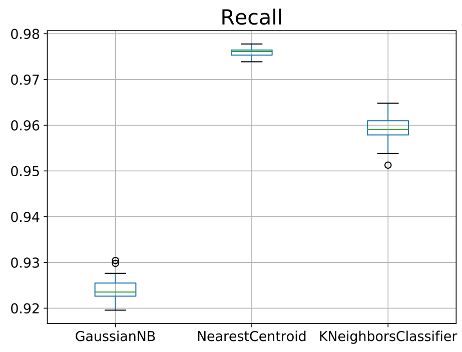
The boxplots of Figure 4.1 shows the data distribution by treatment (i.e., classifier) for each metric (accuracy, precision, recall, F1-score). They suggest the existence of differences among the classifiers. To verify if this difference is statistically significant we performed an analysis of variance test (ANOVA) (see Appendix A.2 for details). We define the minimum difference of practical meaning as 0.8 Cohen's d [5]. Desired error levels are defined as $\alpha = 0.5$ and $\beta = 0.2$. For this setting, using statistical power calculations F-test for one factor balanced ANOVA we found that we need a minimum of 18 samples for each group. However, if the null hypothesis is rejected pairwise comparisons of the classifiers will need to be performed to determine which classifiers are different from the other. These comparisons would requirer 36 samples in each group, because of this, we perform 36 cross-validation experiments with each classifier to collect the samples for the statistical analysis.



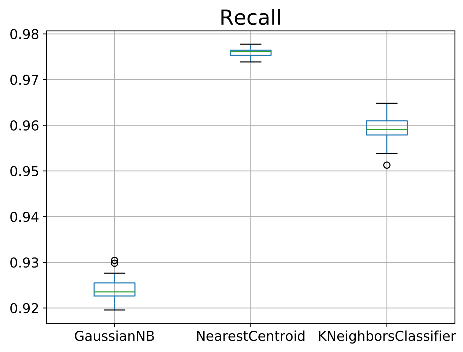
(a) Accuracy



(b) Precision



(c) Recall



(d) F1-Score

Figure 4.1: Data distribution for accuracy, precision and recall, respectively.

Results for the ANOVA test on all metrics are shown in Tables 4.2 to 4.5.

In all cases, the F-statistics and their corresponding P-value ($PR(> F)$) obtained from ANOVA suggests the rejection of the null hypothesis in favour

Table 4.2: Comparison among classifiers' accuracies using ANOVA

	sum_sq	df	F	PR(>F)
C(treatments)	0.050591	2.0	5121.715721	2.145485e-105
Residual	0.000519	105.0	-	-

Table 4.3: Comparison among classifiers' precision using ANOVA

	sum_sq	df	F	PR(>F)
C(treatments)	0.038934	2.0	6027.525317	4.501223e-109
Residual	0.000339	105.0	-	-

Table 4.4: Comparison among classifiers' recall using ANOVA

	sum_sq	df	F	PR(>F)
C(treatments)	0.050591	2.0	5121.715721	4.295516e-35
Residual	0.000519	105.0	-	-

Table 4.5: Comparison among classifiers' F1-Score using ANOVA

	sum_sq	df	F	PR(>F)
C(treatments)	0.052825	2.0	5024.033075	5.835585e-105
Residual	0.000552	105.0	-	-

of the alternative ($P < 0.05$). That is, if the ANOVA assumptions are valid (independence, normality and homoscedasticity of the residuals), we will be able to conclude that there are significant differences among the classifiers when it comes to accuracy, precision, recall, and consequently F1-score.

The independence assumption should be guaranteed on the design of the experiment. In our case, because the data is collected using the cross-validation results and under constant experimental conditions, we have a completely randomized design. Thus, the independence assumption holds true. Moreover, each classifier is trained and tested separately, which also guarantees independence.

The normality assumption can be tested using the Shapiro-Wilk test coupled with a normal QQ plot of the residuals. Table 4.6 shows the results of the Shapiro-Wilk test performed over residual data.

Table 4.6: Results of Shapiro-Wilk test

	W	P-Value
Accuracy	0.9683	0.0111
Precision	0.9708	0.0179
Recall	0.9683	0.0111
F1-Score	0.9673	0.0093

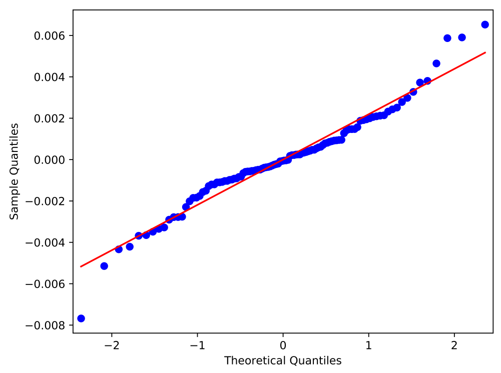
Although the P-values are significant and we rejected the null hypothesis, a closer analysis of the distribution of the residual using the QQ plots of Figure 4.2 allow us to see that the residual is normally distributed for the most part of samples. The portion of the data which is not normally distributed is fairly small and not enough to violate the assumption of normality. Moreover, ANOVA is robust to normality violations, especially if the sample size is large enough [43].

The homoscedasticity assumption can be verified by the Fligner-Killeen test, together with plots of residuals by fitted values, which are shown in Table 4.7 and Figure 4.3, respectively. As we can see, there is a difference between variances across groups. However, ANOVA is also relatively robust to violations of homoscedasticity, especially if the samples are all the same size, like in our case.

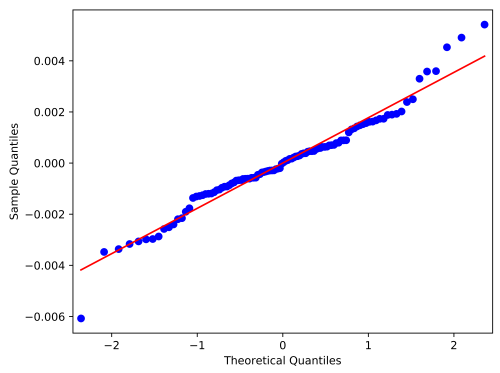
Table 4.7: Results of Fligner-Killeen test

	Fligner Killeen statistic	P-Value
Accuracy	16.6186	0.0002
Precision	14.7287	0.0006
Recall	16.6186	0.0002
F1-Score	17.0741	0.0001

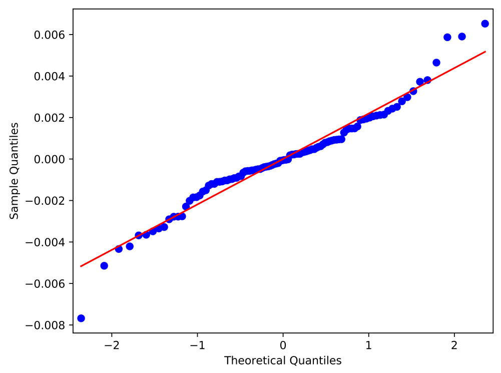
Because we have reasonable ground to believe that the results of ANOVA are trustworthy, we can conclude that there is at least one treatment that is significantly different from the other. However, this test does not allow us to conclude which ones are significantly different from the others. To figure this out we can perform multiple comparisons between two algorithms, which is essentially run a series of t-tests, with some slight modifications (*all vs.*



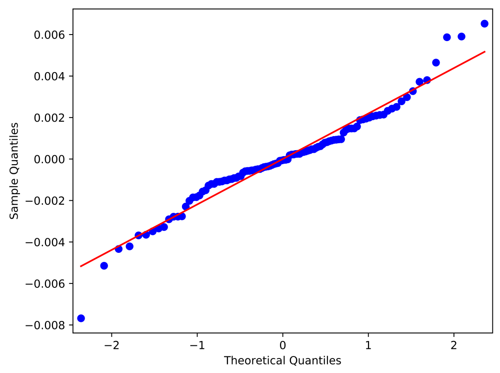
(a) Accuracy



(b) Precision



(c) Recall



(d) F1-Score

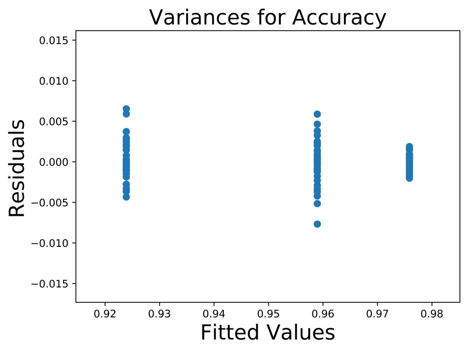
Figure 4.2: Normal QQ plot of the residuals that can be used to verify the ANOVA assumption of normality.

all comparison). The number of comparisons K can be calculated as $K = a(a-1)/2$. Since $a = 3$ (3 algorithms), $K = 3$ compassion.

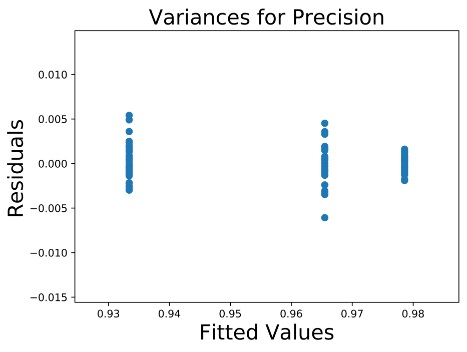
If we are going to perform multiple tests on the same data set, the probability of a type I error on each test is α. If we want to keep our overall error rate controlled at a given level, we will need to correct the value used for each test. Assuming K planned comparisons, the Bonferroni method provides a simples way to adjust α:

$$\alpha_{adj} = \frac{\alpha}{K} = 0.01667 \tag{4.1}$$

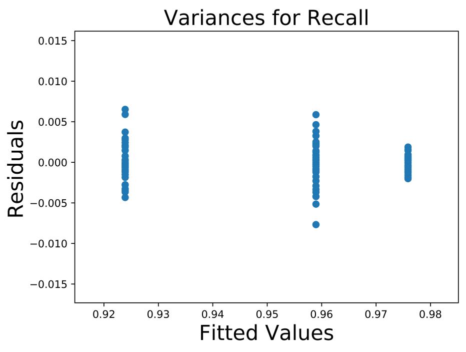
For performing *all vs. all* multiple comparisons, we used Tukey's Honest Significant Difference (HSD) approach, since it provides a slightly higher power when compared to the other methods. Tables 4.8, 4.9, 4.10, and 4.11 show



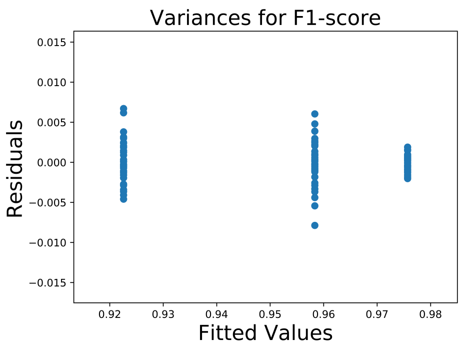
(a) Accuracy



(b) Precision



(c) Recall



(d) F1-Score

Figure 4.3: Plot of residuals by fitted values allows the comparison of the variance across groups

the results for *all vs. all* test regarding the accuracy, precision, recall, and F1-score of the algorithms, respectively. In all cases, the null hypothesis could be rejected with 95% confidence, which means that all three classifiers are different from each other considering these four metrics.

From table 4.8 we can see that when it comes to accuracy, the highest average difference among algorithms is between Nearest Centroids and Naive Bayes. Naive Bayes also has a mean accuracy lower than kNN. Nearest centroids also performs slightly better than kNN when it comes to accuracy. These results corroborate the hypothesis suggested by Figure 4.1a and Table 4.1 that Nearest Centroids achieved the highest average accuracy in assigning the correct type to DBpedia resources.

Figure 4.1b suggested that Naive Bayes reached the lowest average results for precision values. Results from Table 4.9 confirm this hypothesis. On av-

Table 4.8: Multiple Comparison of Means (Accuracy) - Tukey HSD, FWER=0.02

Group 1	Group 2	Mean Diff	P-value	Lower	Upper	Reject
Naive Bayes	kNN	0.0351	< 0.001	0.0336	0.0365	True
Naive Bayes	Nearest Centroid	0.052	< 0.001	0.505	0.0534	True
kNN	Nearest Centroid	0.01619	< 0.001	0.0155	0.0184	True

erage, this classifier has a precision 3.21% lower than kNN and 4.52% lower than the Nearest Neighbors. Considering the 95% confidence level of the experiment, we can also conclude that the nearest centroid algorithm has 1.31% more precision than kNN.

Table 4.9: Multiple Comparison of Means (Precision) - Tukey HSD, FWER=0.02

Group 1	Group 2	Mean Diff	P-value	Lower	Upper	Reject
Naive Bayes	kNN	0.0321	< 0.001	0.0309	0.0333	True
Naive Bayes	Nearest Centroid	0.0452	< 0.001	0.044	0.0464	True
kNN	Nearest Centroid	0.0131	< 0.001	0.0119	0.0143	True

The differences among the recall of the algorithm are shown in Table 4.10 and show that, once again, the Nearest Centroids algorithm performs better than the other two classifiers.

Table 4.10: Multiple Comparison of Means (Recall) - Tukey HSD, FWER=0.02

Group 1	Group 2	Mean Diff	P-value	Lower	Upper	Reject
Naive Bayes	kNN	0.0351	< 0.001	0.0336	0.0365	True
Naive Bayes	Nearest Centroid	0.052	< 0.001	0.0505	0.0534	True
kNN	Nearest Centroid	0.0169	< 0.001	0.0155	0.0184	True

Although F1-score is simply a combination of precision and recall, we also performed the Tukey's HSD test using this metric to simplify the comparison of the overall performance of the classifiers. Results are shown in Table 4.11.

Table 4.11: Multiple Comparison of Means (F1-Score) - Tukey HSD, FWER=0.02

Group 1	Group 2	Mean Diff	P-value	Lower	Upper	Reject
Naive Bayes	kNN	0.0358	< 0.001	0.0343	0.0373	True
Naive Bayes	Nearest Centroid	0.0531	< 0.001	0.0516	0.0546	True
kNN	Nearest Centroid	0.0174	< 0.001	0.0158	0.0189	True

These tests allow us to conclude that the Nearest Centroid classifier is significantly better to our problem, therefore, the remaining experiments on this document were performed using this classifier.

4.3 Retrospective evaluation of classifiers on the Gold Standard

Motivated by the good performance of the binary classifiers (as shown in Table 4.1), we decided to extend our experiments to test whether the proposed supervised approach could detect incorrect type assignments among other entities of the 35 types in our gold standard. For this, we used all samples in the gold standard to train another set of binary classifiers using the best performing algorithm (Nearest Centroid). We then used these classifiers to evaluate 364,791 entity-type pairs belonging those 35 types for which classifiers were trained[2]. Human evaluators performed a retrospective evaluation to verify the result of the classification for a random sample of 3699 predictions.

One of the main advantages of the retrospective evaluation is that it allows a detailed analysis of the classification results, providing the opportunity to inspect the errors made by the classifiers. This often reveals a good insight about the advantages and disadvantages of the proposed solution [32]. Our findings on the errors identified during the retrospective evaluation are presented in Chapter 6.

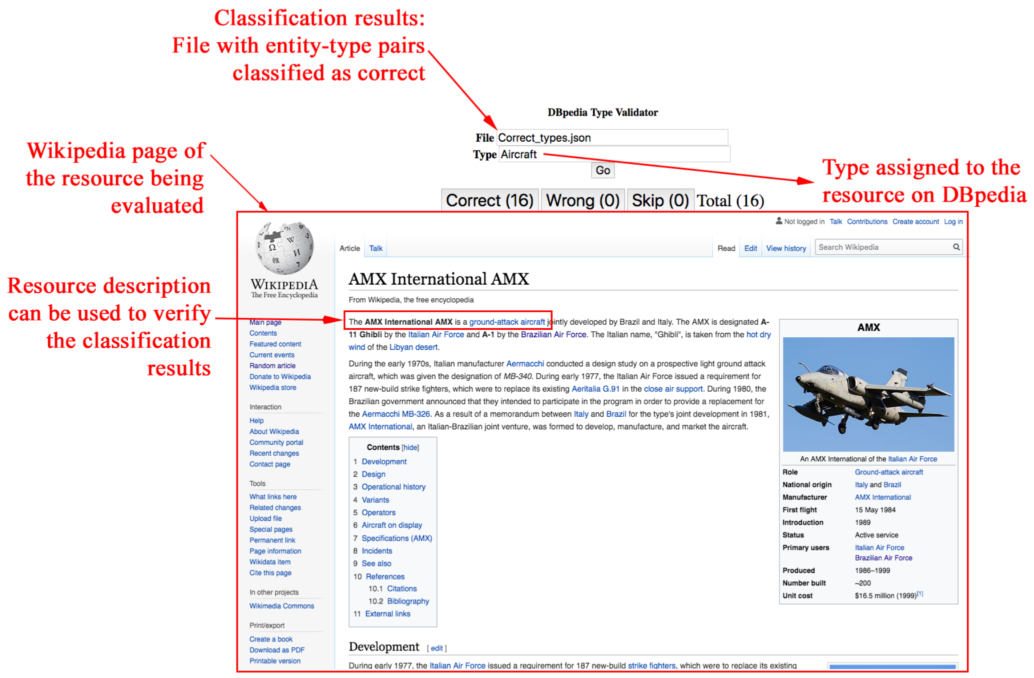
To perform the retrospective evaluation the human evaluators separated the output of the classifier for each entity-type pair into one of the following groups:

[2]Some entities had multiple types.

- True positives: Incorrect entity-type pairs **correctly** flagged as incorrect by the classifiers.
- True negatives: Correct entity-type pairs **correctly** labeled as correct by the classifiers.
- False positives: Correct entity-type pairs **incorrectly** flagged as incorrect by the classifiers.
- False negatives: Incorrect entity-type pairs **incorrectly** labeled as correct by the classifiers.

The correct types for an entity are determined by the human evaluator, using the description of the entity on Wikipedia. That is, the decision about whether or not the type predicted by our method is correct is at the sole discretion of the evaluator. Evaluators were instructed to use their own judgment to resolve inconsistencies and report their assumptions. For example, they reported that no distinction was made between *arenas* and *stadiums*, since the definitions of these terms are vague and in practice, there is not much difference between them. On the other hand, they decided that, if the classifiers failed to detect that a *town* was assigned with the type *township* (or vice versa) the output of the classification should be considered incorrect. Details about all assumptions made by the evaluators during the retrospective evaluation can be found in Appendix C. Evaluators were instructed to skip the evaluation of an entity-type pair if they were unsure about the classification results since some domain-specif entities-type pairs are difficult to be evaluated by people outside the field of knowledge. Figure 4.4 shows a screenshot of the screen presented to the evaluators. Each entity-type pair was evaluated by one evaluator only and, therefore, we did not verify the agreement between the evaluators.

By inspecting the entity-type pairs tagged as false negative by the evaluators, we noticed some discrepancies in the way entities are classified in DBpedia. The most notable example concerns the type `dbo:Animal`, which is used as label for *species* (e.g., `dbr:American_black_bear`) and *individual* racehorses (e.g., `dbr:Fusaichi_Pegasus`). While most the entities with type `dbo:Animal`


Classification results: File with entity-type pairs classified as correct
Wikipedia page of the resource being evaluated
Resource description can be used to verify the classification results
Type assigned to the resource on DBpedia

DBpedia Type Validator

File Correct_types.json

Type Aircraft

Go

Correct (16) Wrong (0) Skip (0) Total (16)

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

In other projects
Wikimedia Commons

Print/export
Create a book
Download as PDF
Printable version

AMX International AMX

From Wikipedia, the free encyclopedia

The **AMX International AMX** is a ground-attack aircraft jointly developed by Brazil and Italy. The AMX is designated **A-11 Ghibli** by the Italian Air Force and **A-1** by the Brazilian Air Force. The Italian name, "Ghibli", is taken from the hot dry wind of the Libyan desert.

During the early 1970s, Italian manufacturer Aermacchi conducted a design study on a prospective light ground attack aircraft, which was given the designation of *MB-340*. During early 1977, the Italian Air Force issued a requirement for 187 new-build strike fighters, which were to replace its existing Aeritalia G.91 in the close air support. During 1980, the Brazilian government announced that they intended to participate in the program in order to provide a replacement for the Aermacchi MB-326. As a result of a memorandum between Italy and Brazil for the type's joint development in 1981, AMX International, an Italian-Brazilian joint venture, was formed to develop, manufacture, and market the aircraft.

Contents [hide]
1 Development
2 Design
3 Operational history
4 Variants
5 Operators
6 Aircraft on display
7 Specifications (AMX)
8 Incidents
9 See also
10 References
10.1 Citations
10.2 Bibliography
11 External links

AMX

An AMX International of the Italian Air Force

Role	Ground-attack aircraft
National origin	Italy and Brazil
Manufacturer	AMX International
First flight	15 May 1984
Introduction	1989
Status	Active service
Primary users	Italian Air Force Brazilian Air Force
Produced	1986–1999
Number built	~200
Unit cost	\$16.5 million (1999)[1]

Development [edit]

During early 1977, the Italian Air Force issued a requirement for 187 new-build strike fighters, which were to replace its existing

Figure 4.4: Screenshot of the interface used for retrospective evaluation.

have properties such as `dbo:family`, `dbo:genus`, `dbo:kingdom`, `dbo:order`, `dbo:phylum`, and `dbo:conservationStatus`, racehorses have properties like `dbo:honours`, `dbo:owner`, `dbo:sex`, `dbo:trainer`, and `dbp:earnings`, which do not belong to type `dbo:Animal` nor to its supertype (`dbo:Eukaryote`)[3]. Therefore, we argue that the individual instances of racehorses should not belong to this type in the ontology. Another evidence that supports our claim is that no other instances of individual animals (e.g., individual dog actors) are labeled as animals on DBpedia. Because of that, we removed from our analysis animals that are also an instance of type racehorse. We further discuss this issue in Chapter 6.

Table 4.12 shows the classifier performance in terms of accuracy, precision, recall, and F1-score (see Appendix A.1 for details) after removing the instances of the special case discussed above and considering the assumptions described in the Appendix C. The performance of the classifier varies across types: for example, both false positive and false negative rates for entities tagged as `dbo:HorseTrainer` is 0%. On the other hand, the false positive rate for the

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

type `dbo:President` is 13%. That is probably because `dbp:President` is a more generic type, which can include presidents of countries, universities, companies, institutes, associations, councils, societies, sports clubs, etc.

Table 4.12: Estimated performance of Nearest Centroid classifiers on unseen entity-type pairs.

	Accuracy	Precision	Recall	F1-Score
Nearest Centroids	0.8883	0.7553	0.9563	0.8439

The manual inspection showed that the proposed approach has a very low false negative rate, which is very encouraging. Moreover, the method is correct about 75% of the times it claims a type assignment is wrong, for a false negative rate below 25%. This results could perhaps be improved with the creation of a better and more comprehensive gold standard. However, this is an expensive task. An alternative approach for collecting data samples to train the classifiers is presented in Section 5.1.

Chapter 5

Unsupervised approach for error detection

Although the gold standard allows us to efficiently compare the classifiers and evaluate our approach, it does not consist of a scalable solution for selecting the reference resources, that is, we cannot achieve our goal of cleaning the entire DBpedia using this method since it would be unfeasible to manually collect enough samples for all types in the ontology.

Secondly, it is virtually impossible to guarantee that the human annotators will not introduce their own bias in the dataset when selecting the reference entities. This can happen, for example, when the annotators select entities which they are familiar with. A biased dataset can largely impact the generalization power of the models.

Finally, some types are very generic, such as `dbo:Person`, which has 175 subtypes in the ontology. Furthermore, there is some inconsistency in the usage of the DBpedia ontology, as illustrated by the example with `dbo:Animal` and *racehorses* on Section 4.3. These inconsistencies make the creation of the gold standard a very demanding task for humans, who are unlikely to be able to think about all possible variations on the usage of the types.

In face of these drawbacks, we developed an alternative approach for automatically selecting the reference resources which are used to perform the type classification.

5.1 Collecting Representative Entities

Our solution relies on the fact that Wikipedia2vec creates embeddings for entities and words simultaneously, and places similar words and entities close to one another in a continuous vector space. This allows us to compute the similarity between an entity and a word. We hypothesize that entities close related to the name of the type are more likely to belong to that type. For example, the similarity between `dbr:Barack_Obama` and `dbo:Politician`, `dbo:Person` or `dbo:President` should be greater than the similarity between `dbr:Barack_Obama` and `dbo:City` or `dbo:Album`. This is because the name of the type and the mention to the entity are likely to occur together in a Wikipedia article (i.e., a Wikipedia article mentioning `dbo:Barack_Obama` it is likely to contain the word *president*) and share a lot of context (i.e., articles that mention *Barack Obama* tend to have semantic similarity with articles that mentions *Politicians*). If this assumption is correct, we can collect reference entities by simply selecting the DBpedia resources whose embeddings closely resemble the word embedding of the name of the type.

The hypernym of the resources (which are often included among the properties of the resource in DBpedia) can also be used as a source of information for collecting the reference resources. The hypernym represents a *type-of relationship* between two terms. For example, the hypernym for `dbr:Canada` is `dbr:Country`. This can be an evidence that `dbr:Canada` is a good example of a resource with type `dbo:Country`. However, the hypernym for many DBpedia resources are also incorrect. For instance, the hypernym for `dbr:Subway_(restaurant)` is `dbr:American`; the hypernym for `dbr:Aspen_University` is `dbr:United` (a resource derived from a disambiguation page). Moreover, although the hypernym for `dbr:Canada` is `dbr:Country`, that does not mean that `dbr:Canada` can only be used as reference for `dbo:Country`. It could also be used to train the classifiers for `dbo:Place`, `dbo:Location`, and `dbo:PopulatedPlace`.

The final selection of reference entities is based on a similarity score between the resource r and the type y, as shown in Equation 5.1. First, for each candidate resource r with type y in DBpedia, we compute the average similarity between the embedding of the resource and the embedding of the n words in the name of the type, we call this average term S_{ry}. To account for the hypernym of the resource, we compute the average similarity between the embeddings of n words on the name of the type and the embeddings of the m words on the hypernym, we call this term S_{yh}. The final similarity score between the resource r and the type y is computed as $ST_{ry} = S_{ry} + \epsilon \cdot S_{yh}$, where ϵ is a given constant.

$$ST_{ry} = \underbrace{\frac{\sum_{i=1}^{n} \text{ similarity } (w_i, r)}{n}}_{S_{ry}} + \epsilon \cdot \underbrace{\frac{\sum_{i=1}^{n} \sum_{j=1}^{m} \text{similarity}(w_i, h_j)}{n+m}}_{S_{yh}} \tag{5.1}$$

To collect samples for a certain type y we simply select the resources with the higher similarity ST_{ry} with the type. In our experiments we empirically defined $\epsilon = 0.75$. Table 5.1 shows a few examples of samples that were automatically selected using this approach.

To quantify the quality of this method, we selected for each type name the most similar 160 entities. We then randomly selected 90 out of the 160 entities for each type. The random selection is purely to promote diversity among the reference resources. We manually verified a random sample of 876 entity-type pairs and found that 93.72% of them were correct. This indicates that, although this approach is not perfect, it can be used to select very high-quality reference entities which can be used for type classification. We evaluate the results achieved by the classifier that used the automatically select resources and found that their performance is very similar to the performance of the classifiers that used the manually curated gold standard (see Section 5.2 for details).

One may argue that we could improve the selection of reference entities by using a different kind of embedding in which information about the properties of types and entities were included. We tried that by creating type2vec embeddings for DBpedia types, in a similar way that we created the resource2vec embeddings described in Section 3.1.3. The first 500 components

Table 5.1: Examples of data samples automatically selected.

Type	Examples
Aircraft	VL_Kotka; Airco_DH.3; Sukhoi_T-3; IAR_823; Stargate_YT-33; Avtek_400A; Gotha_G.X;
Airport	St._Paul_Island_Airport; Carcassonne_Airport; Satna_Airport; LaGuardia_Airport; Brisbane_Airport; Roxas_Airport; Kawama_Airport; Manchester_Airport; Xishuangbanna_Gasa_Airport; Boone_County_Airport_(Arkansas).
Bacteria	Zoogloea; Beggiatoa; Clostridium; Amycolatopsis; Alphaproteobacteria; Acetobacter; Rhodocyclus; Leptospira; Hydrogenothermaceae; Nitrosomonas; Morganella_morganii; Meiothermus; Spirillum;
Country	Bhutan; South_Sudan; Niger; Afghanistan; Dominica; Antigua_and_Barbuda; Botswana; Cyprus; Canada; Sweden;
PopulatedPlace	Hayti_Heights,_Missouri; Gerster,_Missouri; Weston,_Maine; Parker_Strip,_Arizona; Petersville,_Alaska; Paradise_Heights,_Florida; Tilden,_Illinois; Vista,_Missouri; Sims,_Illinois; Cobalt,_Missouri;
RaceHorse	Circular_Quay_(horse); Russia_(horse); Glencoe_II; Alan-a-Dale_(horse); Gimcrack; Silic; Subzero_(horse); Mayano_Top_Gun; Belmar_(horse); Moifaa;
University	Harvard_University; Tianjin_Normal_University; Gansu_Agricultural_University; Wenzhou_University; Hiroshima_City_University; Chang'an_University; Keiwa_College; University_of_Lusaka; Xiangtan_University; Kwassui_Women's_University;

of the embeddings are obtained by averaging the word embeddings (trained with Wikipedia2vec) of the words in the name of the type. The next 300 components are based on the properties of the type, as described in the DBpedia ontology[1]. In this way, we could compare resources to types in DBpedia, and select as reference the resources whose resource2vec embedding close resem-

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

ble the type2vec embeddings, instead of simply comparing word embeddings against Wikipedia2vec embeddings. However, we found that this approach increase in about 10% the number of incorrectly selected samples.

Although it is not clear why the results are worse, we believe that inconsistencies in the DBpedia ontology are playing an important role in lowering the quality of the reference entities. For example, out the 685 types of the ontology, 115 do not have any properties defined (e.g., `dbo:RacingDriver`, `dbo:Province`, `dbo:BiologicalDatabase`, `dbo:Stadium`). Also, several other types have only shared properties, for example, all properties for `dbo:Town` are also present on `dbo:City`, `dbo:Settlement`, `dbo:CityDistrict`, `dbo:Village`, and `dbo:HistoricalSettlement`. The same thing happens with other 326 types. This probably means that introducing the information about the properties of the types not only does not help to enrich the Wikipedia2vec embeddings but also introduces a significant amount of ambiguity to the embeddings, hence adding noise among the reference entities. For this reason, we decided to rely only on Wikipedia2vec embeddings for automatically collect data samples.

5.2 Retrospective evaluation of classifiers on automatically selected reference entities

Because of the apparent good results of the experiments described in Section 4.3, we decided to expand our tests to the entire DBpedia using the classifiers trained using the automatically selected reference resources, as described in the previous section. We trained 557 binary classifiers, one for each type in DBpedia with at least 200 entities. These classifiers were used to judge 2,945,054 entity-type pairs, out of which 1,984,868 were classified as *correct* and 960,186 were classified as *incorrect*.

To asses the performance of these classifiers, once again, human evaluators performed a retrospective evaluation. In total, the classification output for 6607 randomly entity-type pairs of 83 types in the ontology were manually checked. The 83 types used in the evaluation are the same 35 types on the Gold Standard plus 47 types selected manually by the evaluators. These 47 new types were chosen to add some inter-type overlap to the evaluation, allowing us to assess how the classifiers would perform in a more realistic scenario. Another criterion used for selecting the types for evaluation was the number of entities in each types. Because we need at least 160 entities for selecting the reference resources to train the classifiers, only types with more than 200 entities in DBpedia were considered for this evaluation (see Appendix B for the complete list of types included in this analysis).

We tried to compare our results against an alternative also fully automatic approach that consists of comparing of the type of the resource in DBpedia to the type in Wikidata. The motivation for this attempt is that Wikidata is a collaboratively edited knowledge graph, and in principle, it is much cleaner than DBpedia. Moreover, because the vast majority of DBpedia resources are linked to an equivalent in Wikidata (although about half the links may be incorrect [10]), in theory, we can find Wikidata type for the resource.

Despite the challenges associated with the misalignment of the ontologies (as discussed in Section 1.2), for the sake of a baseline for our approach, we tried to verify $3,427,297$ entity-type pairs from DBpedia by comparing them to their equivalent in Wikidata. Out of those, $2,736,658$ comparisons failed because the type in DBpedia was not linked to a type in Wikidata. In total only $690,639$ comparisons were successful.

When comparing the type for a resource in DBpedia and the type for its equivalent in Wikidata we consider that even if the types in the different knowledge graphs are different they both may still be correct. This is particularly true if the type assigned to a resource in one of the knowledge graphs is a subtype of the type assigned to the equivalent resource in the other. Figure 5.1 shows an example. The resource `dbr:Journal_of_Biomedical_Informatics` has type `dbo:AcademicJournal` in DBpedia, and the corresponding type in Wikidata is `academic journal (Q737498)`. But the Wikidata resource equivalent to that journal has type `scientific journal (Q5633421)`.

In other words, even though the types in DBpedia and Wikidatada do not match perfectly, because `scientific journal (Q5633421)` is a subtype of `academic journal (Q737498)` in the ontology of Wikidata, the type in

DBpedia is considered correct. For our comparison, we defined that if the type assigned to a resource in DBpedia is a subtype or a supertype of the type assigned to the equivalent resource in Wikidata by a maximum difference of two levels it should be considered correct.

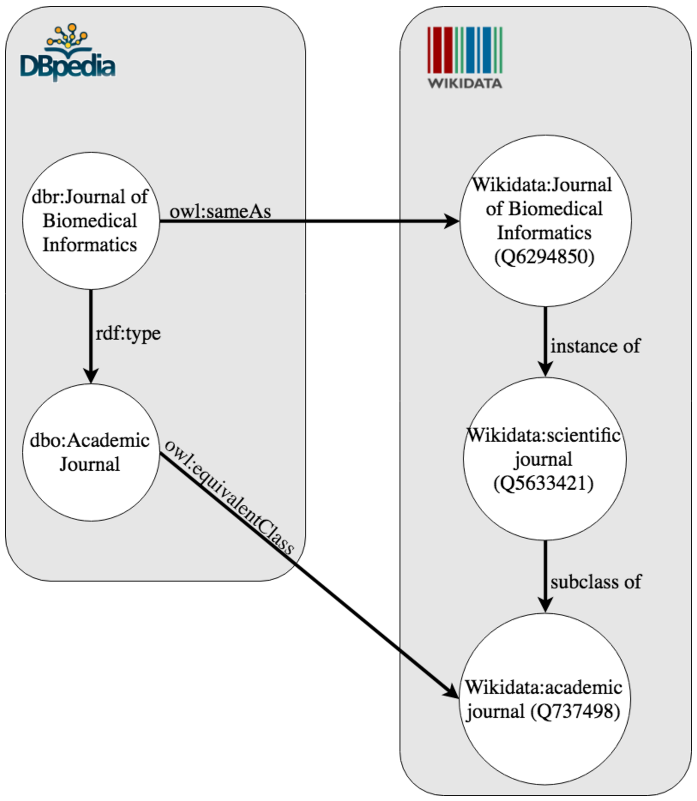

DBpedia
WIKIDATA
dbr:Journal of Biomedical Informatics
owl:sameAs
Wikidata:Journal of Biomedical Informatics (Q6294850)
rdf:type
instance of
dbo:Academic Journal
owl:equivalentClass
Wikidata:scientific journal (Q5633421)
subclass of
Wikidata:academic journal (Q737498)

Figure 5.1: Example of how Wikidata can sometimes be used to verify DBpedia type assignment.

To asses how efficient would be a method to correct DBpedia using wikidata if the ontologies were perfectly aligned, we manually inspected 4, 159 randomly selected entity-type pairs out of the 690, 639 pairs for which the comparison could be performed. Once again, we separated them in four groups:

- True positives: Incorrect entity-type pairs **correctly** flagged as incorrect during the comparison with Wikidata.
- True negatives: Correct entity-type pairs **correctly** labeled as correct during the comparison with Wikidata.

- False positives: Correct entity-type pairs **incorrectly** flagged as incorrect during the comparison with Wikidata.
- False negatives: Incorrect entity-type pairs **incorrectly** labeled as correct during the comparison with Wikidata.

In this way, we could compute accuracy, precision, recall, and F1-score for the comparison with Wikidata. The results are shown in Table 5.2.

Table 5.2: Comparison of different approaches for detecting error in the DBpedia ontology.

	Accuracy	**Precision**	**Recall**	**F1-Score**
Wikidata comparison	0.7169	0.6470	**0.9547**	0.7713
Our Method trained with reference entities selected automatically	**0.8460**	**0.8020**	0.9348	**0.8633**

We can see that the Wikidata comparison achieve a high recall. This is expected because if both DBpedia and Wikidata assigned the same type to an entity, it is very likely that the type is correct. Therefore the false negative rate is very low. On the other hand, the false positive rate is considerably higher than the one achieved by our binary classifiers. Because of that, the Wikidata has the lowest precision.

When investigating the reasons for the high number of false positives resulting from the comparison against Wikidata we notice differences in how the entities are assigned to a type in the ontology in each knowledge graph. For instance, rivers in DBpedia are usually assigned to the type `dbo:River`, which is equivalent to the `river (Q4022)` in Wikidata. However, rivers in Wikidata are usually assigned to the type `stream (Q47521)`. Although both `river (Q4022)` and `stream (Q47521)` have a common ancestor in the Wikidata ontology, they are in different branches (see Figure 5.2). Therefore the comparison of rivers will always generate a false positive. In this particular case, because rivers and steams are not exactly the same thing, one may also argue that the type of Wikidata is incorrect and that is the reason for the false positive. In any case, this is a clear example of the limitations of using an external knowledge graph for error detection.

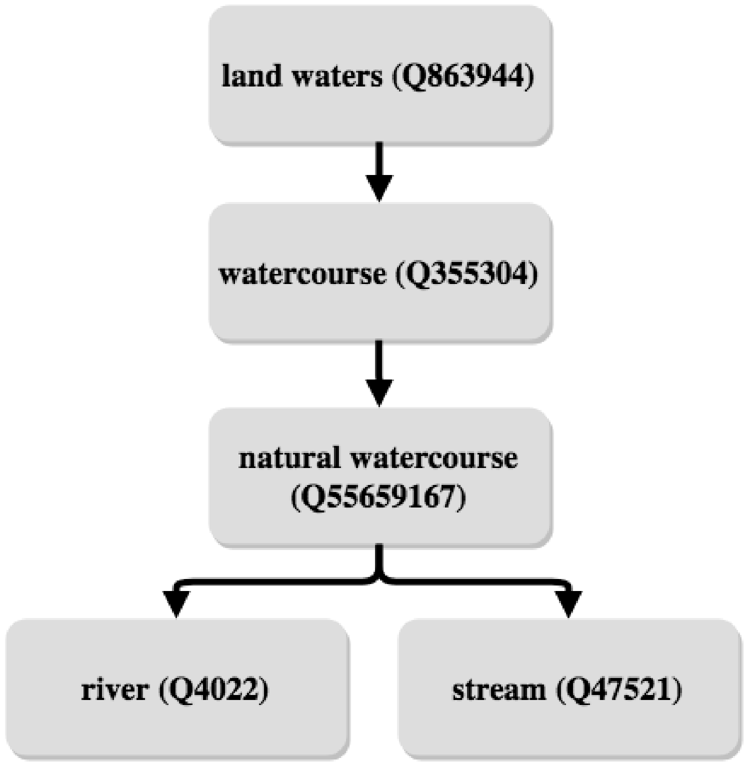


Figure 5.2: Partial Wikidata ontology illustrates a common scenario in which false negatives are likely to happen when using Wikidata to refine DBpedia

Another interesting finding of this experiment resides in the similar performance achieved by the classification using the manually curated gold standard (Table 4.12) and the classification using the automatically selected reference entities. This serves as evidence that although roughly 6% of the representative entities are incorrect (as described in Section 5.1), the overall performance of the classifiers is comparable with their performance when using human-annotated data. One possible explanation for this result is that humans tend to insert their own bias in the dataset when collecting data samples, by selecting resources that they are familiar with (for example, selecting cities in their country of origin). Meanwhile, our proposed method for automatic data collection is capable of selecting a more diverse set of samples. Moreover, even manually curated dataset are susceptible to errors, which means that it is unlikely that 100% of the resources in the gold standard are correct.

Overall, the result from Table 5.2 indicates that our proposed solution can outperform the usage of another knowledge graph for refining DBpedia.

Chapter 6

Error analysis

During the retrospective evaluations described in Sections 4.3 and 5.2 we also investigated the most common errors made by the classifiers. In this Chapter, we present some of them and discuss their possible reasons.

6.1 Errors with rare entities

The first thing that we notice is that rare entities (i.e., entities that are too specific or are not frequently linked in other Wikipedia pages) and entities that are derived from Wikipedia articles that are too short, are more likely to lead to incorrect classifications. This is expected because the Wikipedia2vec embeddings are essentially a representation of the semantic context of the entities. If the semantic context of the entity is poor, the quality of the embedding is low. We compared the Wikipedia articles of entities that were correctly classified against the articles of entities that were incorrectly classified. The results are shown in Table 6.1. On average, the entities correctly classified have longer articles and are more frequently mentioned in other pages.

The only solution that we could envision for improving the classification of rare entities is improving the Wikipedia articles about them. However, because they are rare and attract little public interest, it is unlikely for Wikipedia's editor to be willing to enhance these pages. Nevertheless, due to the same reason, errors on the type assignment of these entities are less visible, since tend to affect fewer applications and users.

We also hypothesized that entities with more properties in DBpedia are

Table 6.1: Comparison between articles of entities correctly and incorrectly classified.

	Average article length	**Average number of references**	**Average number of properties on DBpedia**
Correct classifications	899.25 words	179.29	11.58
Incorrect classifications	783.16 words	156.85	10.52

more likely to be correctly classified. This intuition is based on the fact that the DBpedia2vec components of the resource2vec embeddings are created based on the properties of the resource in DBpedia. However, that does not seem to be the case. When comparing the average number of properties between correct and incorrectly classified entities we can see that, on average, correctly classified resources have only one property more than the incorrectly classified ones, which is probably not enough to justify the errors of the classifiers.

At this point, we are unable to provide a definitive explanation about why the number of properties for the resource in DBpedia does not significantly impact the likelihood of the resource being misclassified. The most probable cause is that, although the number of properties in incorrectly classified resources is the same, the quality of the properties is lower (i.e., they are not strongly related to the type). For example, most of the cities which were correctly classified have properties such as `dbo:populationDensity`, `dbo:areaCode`, `dbo:country`, `dbo:elevation`, `dbo:leaderTitle`, `dbo:leaderName`, and `dbo:populationDensity`. Meanwhile, cities that were misclassified by our classifiers, like `dbr:South_Kannanur` and `dbo:Albanopolis` have less specific properties, such as `dbp:populationAsOf`, `dbp:pushpinMap`, `dbp:pushpinMapCaption`, `dbp:settlementType`, `dbp:latd`, and `dbp:latm`, which are not as useful for categorizing a city and, therefore, make the classification more difficult.

Table 6.2: A few examples of the performance of the binary classifiers for each type during the manual retrospective evaluation, described in section 5.2.

Type	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Aircraft	0.9670	0.9302	1.0000	0.9638
Airport	0.9285	0.8928	1.0000	0.9433
Animal	0.7818	0.7525	1.0000	0.8588
Country	0.9416	1.0000	0.8571	0.9230
Diseases	0.9712	1.0000	0.9200	0.9583
Language	0.8764	0.7954	0.9200	0.9459
PopulatedPlace	0.5049	0.5000	1.0000	0.6666
President	0.6612	0.5833	0.7777	0.6666

6.2 Errors with high-level and ambiguous types

Another observation made during the error analysis is that some of the binary classifiers tend to perform better than others. Table 6.2 shows a few examples. We noticed that classifiers for well-defined and self-contained types (usually those are the lower-level types, without subtypes), such as `dbo:Aircraft`, often have a higher performance than classifiers for more generic types, such as `dbo:President` (which can include presidents of countries, universities, companies, institutes, associations, councils, societies, sports clubs, etc.), or `dbo:PopulatedPlace` (which includes countries, cities, town, villages, states, provinces, territories, etc.).

Using agglomerative hierarchical clustering we can see how resources with the same type can sometimes have very different embeddings. The dendrogram of figure 6.1 shows an example for the type `dbo:Language`. The figure shows that there are at least two very distinct clusters. By sampling entities from each cluster it is easy to see that entities in the first cluster (in green) are languages and dialects used as a method of human communication (e.g., `dbr:English_language`, `dbr:Portuguese_language`, `dbr:French_language`). The second cluster contains programming languages and a lot of noise (i.e., entities of other types than are not language of any kind).

In this particular situation, the classifier was able to achieve reasonable performance on separating languages and programming languages from other

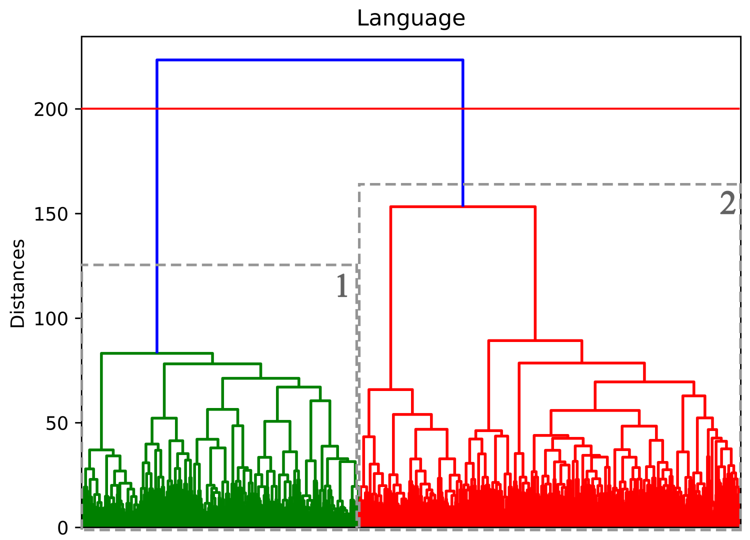


Figure 6.1: Dendrogram for type Language

entities incorrectly labeled as `dbo:Language`. However, this is not always the case. The proposed approach performed poorly on detecting erroneous type assignment for the type `dbo:President`, whose dendrogram is shown on Figure 6.2. In this case, we can see two clusters. The first cluster, in green, contains all different types of politicians (e.g., prime ministers, governors, vice-presidents), which are frequently misclassified as `dbo:President` in DBpedia. The second cluster (red) is mostly other people, but it also contains presidents of different organizations.

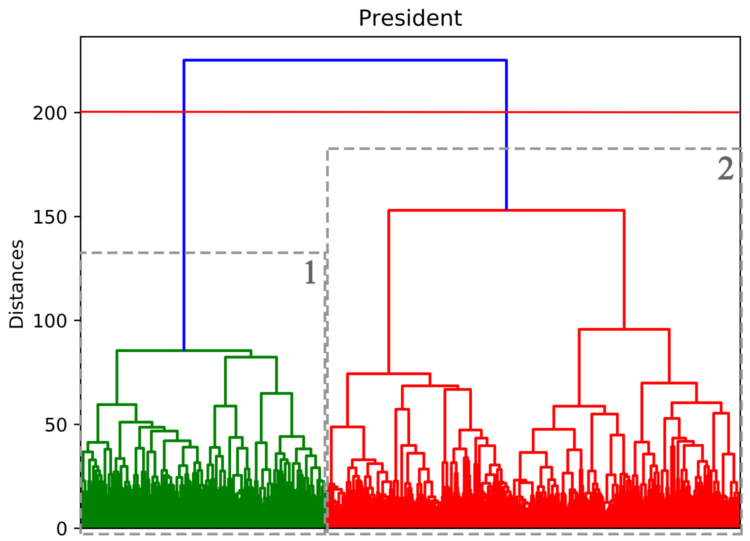


Figure 6.2: Dendrogram for type President

Because there is a lot of semantic similarity between presidents of countries and other politicians, the classifier was unable to distinguish between the entities in the first cluster. In many occasions, it also failed to distinguish the entities in the second cluster. That is because `dbo:President` is a very generic type, and anyone who was president of anything at any point in their lives could be associated to this type, even when the presidency was not the most relevant part of their lives. For example, someone who was class president could be classified as `dbo:President`, but there is very little or none information on their embeddings that would allow the classifiers to separate a president from another generic person.

A similar phenomenon can be observed with the type `dbo:PopulatedPlace`. Figure 6.3 shows that there are two very distinct groups of entities assigned to this type. By analyzing entities in each of the clusters we noticed that cluster 1 (green and red) includes countries, former sovereign states (e.g. Kingdom of Iceland), and former colonies (e.g., Upper Canada), states and provinces, and other larger populated areas (e.g., Vancouver Island). The second cluster (magenta and aqua) includes basically smaller regions, such as cities, towns (e.g., Maarssen, Siderno), districts, and villages. Both clusters contain some noise. The classifiers are able to distinguish between the two clusters, but frequently miss-classifies entities in the same clusters. For example, many cities were classified as towns and vice versa.

6.3 Errors due to inconsistencies in DBpedia

Sometimes, classifiers for lower-level and more specific types also do not perform as well as expected. That is the case of the classifier for `dbo:Animal`, which achieved only 78% of accuracy. Although this type has a few subtypes (such as `dbo:Amphibian` and `dbo:Insect`) its properties are very specific, including, for example, `dbp:binomial`, `dbp:conservationStatus`, `dbp:kingdom`, and `dbp:genus`. We would expect that these properties, combined with the semantic information about the entities, would be enough to distinguish this type from the others. But as we already discussed in Section 4.3, that did

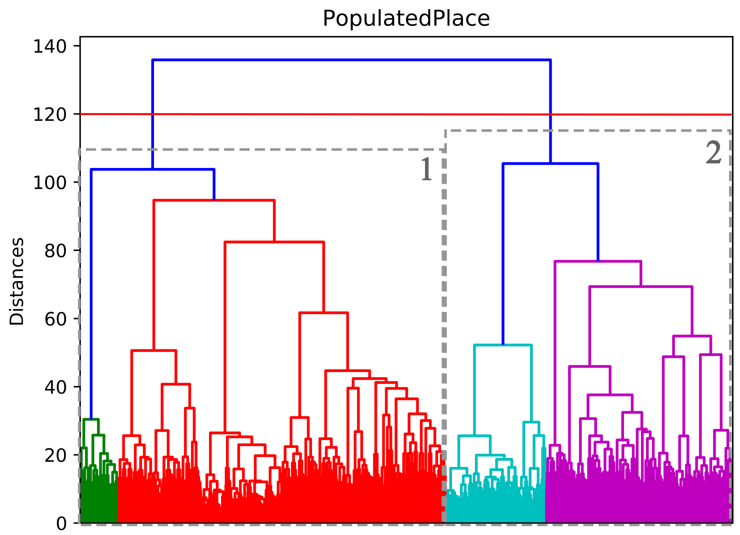


Figure 6.3: Dendrogram for type PopulatedPlace

not happen. During the retrospective evaluation, we noticed that all false-positives involving this type were racehorses, which are in fact animals, but were flagged as *not an animal* by the classifiers. However, a closer analysis reviewed that the properties of racehorses are very different from the properties for the other animals, and are not defined as belonging to the type `dbo:Animal` or its supertype `dbo:Eukaryote` (examples include `dbo:honours`, `dbo:owner`, `dbo:sex`, `dbo:trainer`, and `dbp:earnings`).

The dendrogram of Figure 6.4 shows how the entities of `dbo:Animal` were clustered and how the embeddings of the racehorses (second cluster, in red) are distant from the embeddings of the other animals. The first cluster (green) contains mostly species of animals (e.g., `dbr:Tanzanian_woolly_bat`, `dbr:Echis_carinatus`, `dbr:Armenian_lizard`). The majority of the entities in the third cluster are animal families (e.g., `dbr:Ulidiidae`, `dbr:Cyrtonyx`).

When comparing the representative entities of the with the entities in each cluster, we found that the average cosine similarity between the entities of cluster 1 and entities labeled `dbo:Animal` among the representative ones is 0.3865. The average similarity between cluster 3 and the representative entities is 0.36063. Between cluster 2 and representative entities the similarity drops to 0.1996. In fact, none of the representative entities samples was a racehorse.

Whether or not racehorses should be classified as animals in DBpedia is

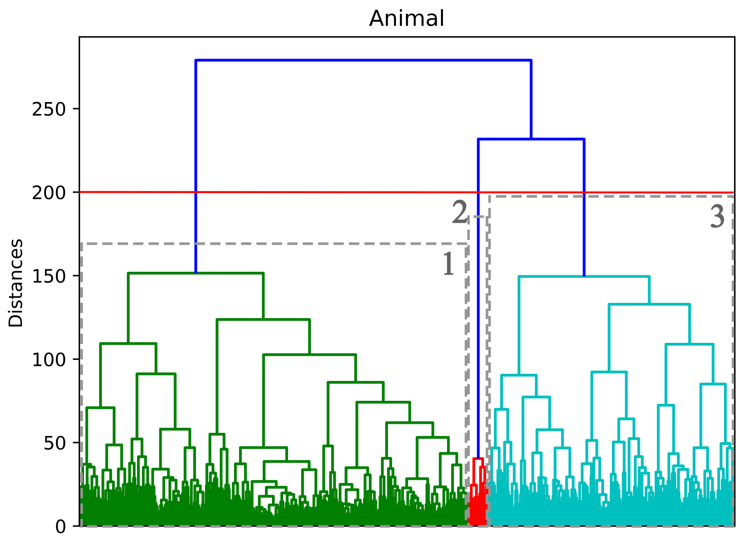


Figure 6.4: The dendrogram for type Animal allows us to see three distinct clusters. The red one, racehorse, is responsible for 100% of the false positive classification for this type.

debatable. While some may believe that this classification represents an inconsistency on the usage of the ontology (since its properties do not belong to the type), others argue that the classification is correct and perhaps the properties for the type should include things like `dbo:owner`, `dbo:sex`, `dbo:trainer`, etc. In either case, this example illustrates how challenging it can be to verify the type for some of the resources, even for simple types like `dbo:Animal`.

This is not the only type in the DBpedia ontology that seems to have problems. Throughout our error analysis, we found several other cases of what we consider to be inconsistencies in the ontology. For example, we noticed that, at the time of writing, 196 out of the 685 types do not have any resources associated to them, including very important ones that could have millions of entities, such as `dbo:MusicComposer`, `dbo:Humorist`, `dbo:TelevisionDirector`, and `dbo:VicePresident`. Although our proposed solution can be used to assign types to an entity even if it has no type, it will not be able to assign an entity to a type that have not been used yet on the knowledge graph, because in these cases, no classifier was trained (i.e., no reference entities can be collected). The only way to overcome this limitation of the proposed solution would be manually collecting a few reference entities for the type without resources, so the classifier for the type could be trained.

There are also several types in DBpedia that do not make sense as types and should be resources instead, such as `dbo:Altitude` (height) and `dbo:GrossDomesticProduct`. Other types seem to not fit well in the ontology, for example `dbo:Type`, which has subtypes `dbo:DocumentType` and `dbo:GovernmentType`. Although we can easily identify entities belonging to the subtypes (e.g., `dbr:Democracy`), it is difficult to imagine an application in which classifying an entity simply as `dbo:Type` would be useful.

On the other hand, we missed a few types in the ontology. For example, DBpedia makes no distinction between *ship* and *boat*. It also does not distinguish between libraries, institutions that lend books and provide information, and software libraries. Although those entities are fairly different from each other, they belong to the same type in the ontology. There is also no specific type that could be used to assign a financial institution other than banks. Because of that many companies in the financial market (e.g., stockbrokers) end up being classified as `dbo:Bank`.

These examples illustrate how challenging it is to assign resources to types in the ontology, specially when there are a lot of problems with the ontology itself.

Chapter 7

Conclusion

Because knowledge graphs (KGs) allow knowledge to be stored in a machine-readable format, they have been widely used in many Natural Language Processing (NLP) applications, such as question-answering systems and conversational agents. Moreover, they have been used to support a wide range of NLP tasks, including but not limited to data integration, named entity recognition, topic detection, and document ranking.

Various approaches have been applied to construct KGs. Each of them has its own advantages and disadvantages. In general, manually curated KG tend to suffer from limited coverage, while graphs constructed automatically are more likely to contain errors. That means that a trade-off between completeness and correctness is usually necessary for the construction of KGs. Because of this, a lot of research effort has been deployed on refining KGs. Unlike KG construction techniques, refining approaches assume that the graph has already been built and it can be improved by adding missing knowledge or correcting erroneous information.

In this work, we presented a fully automatic approach for KG refining, which detects types erroneously assigned to the entities of the graphs. Our solution creates a semantic representation (embeddings) of the entities by combining information from Wikipedia and DBpedia. These embeddings are used to train binary machine learning classifiers which are able to distinguish between the resources of a specific type from the resources of all other types.

To test our solution we first tried to select representative entities of each types using a external KG, such as Google's KG or Wikidata, which are meant to be less noisy than DBpedia. This would allow us to quickly collect a large number of data samples. However, this approach has several disadvantages. First of all, the external KG can also be incorrect or incomplete. Google, for example, is much more conservative than DBpedia when assigning types to the entities in their KG. Sometimes, only very high-level types, such as *Thing*, are assigned. Secondly, the ontologies of different KGs are usually not aligned. This means that some types in one KG may not have an equivalent type on the other. Finally, finding equivalent resources on different KGs may also be a problem. It has been shown that up to 50% of the interlinks connecting resources on DBpedia to equivalent resources on other KGs are incorrect [10].

Because of the challenges on using an external KG, we had to create a manually curated partial gold standard to assess the viability of our solution and to compare the different algorithms for creating the classifiers. We collected 3876 entities of 35 different types of the DBpedia ontology. The initial thought was that if the approach works well in this simplified setting, it may be worth scaling the solution to consider all types in DBpedia. The results of this experiment indicated that the proposed approach can achieve more than 97% of accuracy, precision, and recall.

Motivated by the good results of the experiments in the gold standard, we decided to test the classifiers on all DBpedia resources that belong to one of the 35 types of the gold standard. In total 364,791 entity-type pairs were checked by the classifiers. Human evaluators then verified the output of the classifiers for a random sample of 3699 predictions. In this experiment the proposed approach achieve more than 88% of accuracy and almost 85% of F1-score.

Even though these two experiments demonstrated the overall quality of the solution, the necessity of manually selecting the representative entities poses a significant limitation on the applicability of the method. For training classifiers for all 489 types in DBpedia with at least one entity, it would be necessary to collect roughly 97,800 data samples (200 per type), which would be expensive, and time-consuming. Because of this, we introduced an alternative approach to automatically select the representative entities from each type by retrieving the resources with a higher probability of being of the type. Our experiments showed that about 94% of the data samples collected this way are correct, and the overall performance of the classifiers is similar to their performance when trained using the manually curated gold standard.

During our error analysis, we found that some of the binary classifiers perform better than others. In general, generic and high-level types, such as `dbo:PopulatedPlace` and `dbo:President`, lead to more classification errors. However, in some cases, the errors made by the classifiers are actually due to inconsistencies in the ontology of the KG. This includes types that are ambiguous and end up being assigned to completely different resources, like `dbo:Library`, and types with resources whose properties do not belong to the type, like racehorses being labeled as `dbo:Animal`.

7.1 Contributions

The main contributions of this work can be summarized as follows:

- We propose an approach for enriching the semantic representation (embeddings) of the entities of a KG using the properties of the entities on the graph.
- We show how these embeddings can be used to associate the entities to types in the ontology.
- We proposed a fully automatic approach for selecting entities that represent a type in the ontology. These entities can be used as training data for machine learning models.
- We create a gold standard for evaluating our approach, which can also be used for other researchers working on similar problems.

7.2 Threats to validity

As with any empirical study, our results need to be interpreted as inherently approximate. Every effort was made to avoid overt biases when selecting types for our gold standard or when select more types for the larger experiment with our unsupervised method. Nevertheless, it is possible that our results do not carry over to other types. Moreover, the fact that we only analyzed a randomly selected subsample of the classifications, and each classification is verified by only one evaluator may be another threat to the validity of our conclusions.

7.3 Future work

The lower performance of the classifiers for high-level types can be explained by a limitation in our approach for selecting representative entities, which is not able to capture all different kinds of resources for a high-level type. For example, most of the training samples for `dbo:PopulatedPlace` are cities, and examples of other types of populated place are rare among the representative entities. A simple possible solution that we plan to test and evaluate in a future work, would be applying the algorithm for selecting representative entities recursively, to encompass all subtypes for a type. For example, data for training a classifier for `dbo:PopulatedPlace` can be obtained by applying the approach described in section 5.1 for the types `dbo:Agglomeration`, `dbo:Community`, `dbo:Continent`, `dbo:Country`, etc.

However, this approach may not be enough to solve the problem for all types. Some of them, such as `dbo:President` are generic even though they have no subtype. This technique would not solve the problem for types that are being inconsistently used, like `dbo:Animal` (as discussed in section 4.3). Even including samples for all subtypes of `dbo:Animal`, no racehorse would appear among the representative entities for the type.

It is clear that the approach for collecting reference entities still needs to be improved in order to increase the overall performance of the classifiers, and this should be the focus at a future work. Another prospective continuation of this work involves testing the proposed solution on a different KG, such as Wikidata. Although Wikidata does not contain the same amount of errors of DBpedia, it can still be benefited because our method can also be used to assign types to the entities if the type is missing.

In the future, we also want to investigate if the resource2vec embeddings could be used to identify other issues in the KG. For example, we believe that by comparing resource2vec embeddings created with the DBpedia properties against the resource2vec embeddings created with the properties on a target KG, we may be able to identify incorrect interlinks between resources in DBpedia and the target KGs. Similarly, by analyzing the embeddings of two entities connected by a relation in a KG, we can try to predict whether or not the relationship is correct. The resource2vec embeddings can also probably be used to identify problems in the ontology itself, for instance, ambiguous types (e.g., `dbo:Library`) that should be split into two or more types.

References

[1] A. P. Aprosio, C. Giuliano, and A. Lavelli, "Automatic expansion of dbpedia exploiting wikipedia cross-language information," in *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, P. Cimiano, Ó. Corcho, V. Presutti, L. Hollink, and S. Rudolph, Eds., ser. Lecture Notes in Computer Science, vol. 7882, Springer, 2013, pp. 397–411. DOI: 10.1007/978-3-642-38288-8_27. [Online]. Available: https://doi.org/10.1007/978-3-642-38288-8%5C_27. 18

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 2003. [Online]. Available: http://jmlr.org/papers/v3/blei03a.html. 18

[3] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, J. T. Wang, Ed., ACM, 2008, pp. 1247–1250. DOI: 10.1145/1376616.1376746. [Online]. Available: https://doi.org/10.1145/1376616.1376746. 2, 12

[4] M. Cannaviccio, D. Barbosa, and P. Merialdo, "Accurate fact harvesting from natural language text in wikipedia with lector," in *Proceedings of the 19th International Workshop on Web and Databases, San Francisco, CA, USA, June 26, 2016*, ACM, 2016, p. 9. DOI: 10.1145/2932194.2932203. [Online]. Available: https://doi.org/10.1145/2932194.2932203. 18

[5] J. Cohen, Ed., *Dedication.* Academic Press, 1977, p. v, ISBN: 978-0-12-179060-8. DOI: https://doi.org/10.1016/B978-0-12-179060-8.50003-7. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780121790608500037. 36

[6] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, and R. Ghani, Eds., ACM,

2014, pp. 601–610. DOI: `10.1145/2623330.2623623`. [Online]. Available: `https://doi.org/10.1145/2623330.2623623`. 15

[7] A. Efrati. (2012). Google gives search a refresh, [Online]. Available: `https://www.wsj.com/articles/SB10001424052702304459804577285C-1842851136290` (visited on 07/03/2019). 15

[8] M. Färber, B. Ell, C. Menne, and A. Rettinger, "A comparative survey of dbpedia, freebase, opencyc, wikidata, and yago," *Semantic Web Journal, July*, pp. 1–26, 2015. [Online]. Available: `http://www.semantic-web-journal.net/system/files/swj1141.pdf`. 1, 6, 11, 12

[9] Y. Gao, J. Liang, B. Han, M. Yakout, and A. Mohamed, "Building a large-scale, accurate and fresh knowledge graph," in *SigKDD*, ACM, 2018. 15

[10] H. Halpin and P. J. Hayes, "When owl: Sameas isn't the same: An analysis of identity links on the semantic web," CEUR Workshop Proceedings, vol. 628, C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, Eds., 2010. [Online]. Available: `http://ceur-ws.org/Vol-628/ldow2010%5C_paper09.pdf`. 6, 17, 51, 64

[11] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *14th International Conference on Computational Linguistics, COLING 1992, Nantes, France, August 23-28, 1992*, 1992, pp. 539–545. [Online]. Available: `https://www.aclweb.org/anthology/C92-2082/`. 18

[12] J. Hipp, U. Güntzer, and G. Nakhaeizadeh, "Algorithms for association rule mining - A general survey and comparison," *SIGKDD Explorations*, vol. 2, no. 1, pp. 58–64, 2000. DOI: `10.1145/360402.360421`. [Online]. Available: `https://doi.org/10.1145/360402.360421`. 18

[13] S. Jang, Megawati, J. Choi, and M. Y. Yi, "Semi-automatic quality assessment of linked data without requiring ontology," in *Proceedings of the Third NLP&DBpedia Workshop (NLP & DBpedia 2015) co-located with the 14th International Semantic Web Conference 2015 (ISWC 2015), Bethlehem, Pennsylvania, USA, October 11, 2015.*, H. Paulheim, M. van Erp, A. Filipowska, P. N. Mendes, and M. Brümmer, Eds., ser. CEUR Workshop Proceedings, vol. 1581, CEUR-WS.org, 2015, pp. 45–55. [Online]. Available: `http://ceur-ws.org/Vol-1581/paper5.pdf`. 21

[14] J. Kim, E. Kim, Y. Won, S. Nam, and K. Choi, "The association rule mining system for acquiring knowledge of dbpedia from wikipedia categories," in *Proceedings of the Third NLP&DBpedia Workshop (NLP & DBpedia 2015) co-located with the 14th International Semantic Web Conference 2015 (ISWC 2015), Bethlehem, Pennsylvania, USA, October 11, 2015.*, H. Paulheim, M. van Erp, A. Filipowska, P. N. Mendes, and M. Brümmer, Eds., ser. CEUR Workshop Proceedings, vol. 1581, CEUR-WS.org, 2015, pp. 68–80. [Online]. Available: `http://ceur-ws.org/Vol-1581/paper7.pdf`. 19

[15] T. Kliegr, "Linked hypernyms: Enriching dbpedia with targeted hypernym discovery," *J. Web Semant.*, vol. 31, pp. 59–69, 2015. DOI: `10.1016/j.websem.2014.11.001`. [Online]. Available: `https://doi.org/10.1016/j.websem.2014.11.001`. 18

[16] J. Lee, S. Seo, and Y. S. Choi, "Semantic relation classification via bidirectional LSTM networks with entity-aware attention using latent entity typing," *Symmetry*, vol. 11, no. 6, p. 785, 2019. DOI: `10.3390/sym11060785`. [Online]. Available: `https://doi.org/10.3390/sym1106%5C-0785`. 18

[17] J. Lehmann and L. Bühmann, "ORE - A tool for repairing and enriching knowledge bases," in *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part II*, P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, and B. Glimm, Eds., ser. Lecture Notes in Computer Science, vol. 6497, Springer, 2010, pp. 177–193. DOI: `10.1007/978-3-642-17749-1\_12`. [Online]. Available: `https://doi.org/10.1007/978-3-642-17749-1%5C_12`. 21

[18] J. Lehmann, D. Gerber, M. Morsey, and A. N. Ngomo, "Defacto - deep fact validation," in *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, Eds., ser. Lecture Notes in Computer Science, vol. 7649, Springer, 2012, pp. 312–327. DOI: `10.1007/978-3-642-35176-1\_20`. [Online]. Available: `https://doi.org/10.1007/978-3-642-35176-1%5C_20`. 21

[19] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015. DOI: `10.3233/SW-140134`. [Online]. Available: `https://doi.org/10.3233/SW-140134`. 2, 10, 13

[20] Y. Lin, S. Shen, Z. Liu, H. Luan, and M. Sun, "Neural relation extraction with selective attention over instances," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, The Association for Computer Linguistics, 2016. [Online]. Available: `https://www.aclweb.org/anthology/P16-1200/`. 18

[21] S. Liu, M. d'Aquin, and E. Motta, "Towards linked data fact validation through measuring consensus," in *Proceedings of the 2nd Workshop on Linked Data Quality co-located with 12th Extended Semantic Web Conference (ESWC 2015), Portorož, Slovenia, June 1, 2015.*, A. Rula, A. Zaveri, M. Knuth, and D. Kontokostas, Eds., ser. CEUR Workshop

Proceedings, vol. 1376, CEUR-WS.org, 2015. [Online]. Available: http://ceur-ws.org/Vol-1376/LDQ2015%5C_paper%5C_04.pdf. 21

[22] Y. Ma, H. Gao, T. Wu, and G. Qi, "Learning disjointness axioms with association rule mining and its application to inconsistency detection of linked data," in *The Semantic Web and Web Science - 8th Chinese Conference, CSWS 2014, Wuhan, China, August 8-12, 2014, Revised Selected Papers*, D. Zhao, J. Du, H. Wang, P. Wang, D. Ji, and J. Z. Pan, Eds., ser. Communications in Computer and Information Science, vol. 480, Springer, 2014, pp. 29–41. DOI: 10.1007/978-3-662-45495-4_3. [Online]. Available: https://doi.org/10.1007/978-3-662-45495-4%5C_3. 4, 19

[23] F. Mahdisoltani, J. Biega, and F. M. Suchanek, "YAGO3: A knowledge base from multilingual wikipedias," in *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, www.cidrdb.org, 2015. [Online]. Available: http://cidrdb.org/cidr2015/Papers/CIDR15%5C_Paper1.pdf. 14

[24] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval.* Cambridge University Press, 2008, ISBN: 978-0-521-86571-5. DOI: 10.1017/CBO9780511809071. [Online]. Available: https://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf. 30, 31

[25] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, Eds., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality. 22

[26] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*, K. Su, J. Su, and J. Wiebe, Eds., The Association for Computer Linguistics, 2009, pp. 1003–1011. [Online]. Available: https://www.aclweb.org/anthology/P09-1113/. 2

[27] C. Newton. (2012). Google's knowledge graph tripled in size in seven months, [Online]. Available: https://www.cnet.com/news/googles-knowledge-graph-tripled-in-size-in-seven-months/ (visited on 07/03/2019). 15

[28] A. Nikolov, V. S. Uren, E. Motta, and A. N. D. Roeck, “Overcoming schema heterogeneity between linked semantic repositories to improve coreference resolution,” in *The Semantic Web, Fourth Asian Conference, ASWC 2009, Shanghai, China, December 6-9, 2009. Proceedings*, A. Gómez-Pérez, Y. Yu, and Y. Ding, Eds., ser. Lecture Notes in Computer Science, vol. 5926, Springer, 2009, pp. 332–346. DOI: `10.1007/978-3-642-10871-6\_23`. [Online]. Available: `https://doi.org/10.1007/978-3-642-10871-6%5C_23`. 6

[29] X. Niu, X. Sun, H. Wang, S. Rong, G. Qi, and Y. Yu, “Zhishi.me - weaving chinese linking open data,” in *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part II*, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, and E. Blomqvist, Eds., ser. Lecture Notes in Computer Science, vol. 7032, Springer, 2011, pp. 205–220. DOI: `10.1007/978-3-642-25093-4\_14`. [Online]. Available: `https://doi.org/10.1007/978-3-642-25093-4%5C_14`. 19

[30] A. G. Nuzzolese, A. Gangemi, V. Presutti, F. Draicchio, A. Musetti, and P. Ciancarini, “Tìpalo: A tool for automatic typing of dbpedia entities,” in *The Semantic Web: ESWC 2013 Satellite Events*, P. Cimiano, M. Fernández, V. Lopez, S. Schlobach, and J. Völker, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 253–257, ISBN: 978-3-642-41242-4. 18

[31] H. Paulheim, *Browsing linked open data with auto complete*, Unpublished manuscript, 2012. [Online]. Available: `https://madoc.bib.uni-mannheim.de/33082/1/dbpediaatc.pdf`. 18

[32] ——, “Knowledge graph refinement: A survey of approaches and evaluation methods,” *Semantic Web*, vol. 8, no. 3, pp. 489–508, 2017. DOI: `10.3233/SW-160218`. [Online]. Available: `https://doi.org/10.3233/SW-160218`. 1–4, 6, 10, 13–15, 17, 21

[33] H. Paulheim and C. Bizer, “Type inference on noisy RDF data,” in *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, Eds., ser. Lecture Notes in Computer Science, vol. 8218, Springer, 2013, pp. 510–525. DOI: `10.1007/978-3-642-41335-3\_32`. [Online]. Available: `https://doi.org/10.1007/978-3-642-41335-3%5C_32`. 3, 17

[34] ——, “Improving the quality of linked data using statistical distributions,” *Int. J. Semantic Web Inf. Syst.*, vol. 10, no. 2, pp. 63–86, 2014. DOI: `10.4018/ijswis.2014040104`. [Online]. Available: `https://doi.org/10.4018/ijswis.2014040104`. 5, 17, 21

[35] F. C. F. Pinto, *Design and analysis of experiments: Analysis of variance*, Sep. 2016. [Online]. Available: `https://github.com/fcampelo/Design-and-Analysis-of-Experiments/blob/master/10-Analysis%5C-OfVariance/Chapter10.pdf`. 78

[36] D. Pogue. (2012). Going beyond search, into fetch, [Online]. Available: `https://www.nytimes.com/2012/05/24/technology/personaltech/google-and-microsoft-feature-do-it-all-search-pages-state-of-the-art.html` (visited on 07/03/2019). 15

[37] A. Singhal. (2012). Introducing the knowledge graph: Things, not strings, [Online]. Available: `https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html` (visited on 07/2019). 13, 15

[38] J. Sleeman and T. Finin, "Type prediction for efficient coreference resolution in heterogeneous semantic graphs," in *2013 IEEE Seventh International Conference on Semantic Computing, Irvine, CA, USA, September 16-18, 2013*, IEEE Computer Society, 2013, pp. 78–85. DOI: `10.1109/ICSC.2013.22`. [Online]. Available: `https://doi.org/10.1109/ICSC.2013.22`. 17

[39] J. Sleeman, T. Finin, and A. Joshi, "Topic modeling for RDF graphs," in *Proceedings of the Third International Workshop on Linked Data for Information Extraction (LD4IE2015) co-located with the 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 12, 2015.*, A. L. Gentile, Z. Zhang, C. d'Amato, and H. Paulheim, Eds., ser. CEUR Workshop Proceedings, vol. 1467, CEUR-WS.org, 2015, pp. 48–62. [Online]. Available: `http://ceur-ws.org/Vol-1467/LD4IE2015%5C_Sleeman.pdf`. 18

[40] R. Socher, D. Chen, C. D. Manning, and A. Y. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, Eds., 2013, pp. 926–934. [Online]. Available: `http://papers.nips.cc/paper/5028-reasoning-with-neural-tensor-networks-for-knowledge-base-completion`. 19

[41] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, Eds., ACM, 2007, pp. 697–706. DOI: `10.1145/1242572.1242667`. [Online]. Available: `https://doi.org/10.1145/1242572.1242667`. 2, 14

[42] R. Tibshirani, T. Hastie, B. Narasimhan, and G. Chu, "Diagnosis of multiple cancer types by shrunken centroids of gene expression," *Proceedings of the National Academy of Sciences*, vol. 99, no. 10, pp. 6567–6572, 2002, ISSN: 0027-8424. DOI: 10.1073/pnas.082099299. eprint: https://www.pnas.org/content/99/10/6567.full.pdf. [Online]. Available: https://www.pnas.org/content/99/10/6567. 30

[43] M. L. Tiku, "Power function of the f-test under non-normal situations," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 913–916, 1971, ISSN: 01621459. [Online]. Available: http://www.jstor.org/stable/2284254. 38

[44] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999. 8, 24

[45] G. Töpper, M. Knuth, and H. Sack, "Dbpedia ontology enrichment for inconsistency detection," in *I-SEMANTICS 2012 - 8th International Conference on Semantic Systems, I-SEMANTICS '12, Graz, Austria, September 5-7, 2012*, V. Presutti and H. S. Pinto, Eds., ACM, 2012, pp. 33–40. DOI: 10.1145/2362499.2362505. [Online]. Available: https://doi.org/10.1145/2362499.2362505. 21

[46] A. Uyar and F. M. Aliyu, "Evaluating search features of google knowledge graph and bing satori: Entity types, list searches and query interfaces," *Online Information Review*, vol. 39, no. 2, pp. 197–213, 2015. DOI: 10.1108/OIR-10-2014-0257. [Online]. Available: https://doi.org/10.1108/OIR-10-2014-0257. 14, 15

[47] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," *Communications of the ACM*, vol. 57, pp. 78–85, Sep. 2014. DOI: 10.1145/2629489. 2, 13

[48] L. Wang, Z. Cao, G. de Melo, and Z. Liu, "Relation classification via multi-level attention cnns," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, The Association for Computer Linguistics, 2016. [Online]. Available: https://www.aclweb.org/anthology/P16-1123/. 18

[49] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 12, pp. 2724–2743, 2017. DOI: 10.1109/TKDE.2017.2754499. [Online]. Available: https://doi.org/10.1109/TKDE.2017.2754499. 18

[50] D. Wienand and H. Paulheim, "Detecting incorrect numerical data in dbpedia," in *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin,

S. Staab, and A. Tordai, Eds., ser. Lecture Notes in Computer Science, vol. 8465, Springer, 2014, pp. 504–518. DOI: 10.1007/978-3-319-07443-6_34. [Online]. Available: https://doi.org/10.1007/978-3-319-07443-6%5C_34. 21

[51] Wikipedia contributors, *K-nearest neighbors algorithm — Wikipedia, the free encyclopedia*, [Online; accessed July-2019], 2019. [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm. 32

[52] P. Xu and D. Barbosa, "Connecting language and knowledge with heterogeneous representations for neural relation extraction," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds., Association for Computational Linguistics, 2019, pp. 3201–3206. [Online]. Available: https://www.aclweb.org/anthology/N19-1323/. 18

[53] I. Yamada, A. Asai, H. Shindo, H. Takeda, and Y. Takefuji, "Wikipedia2vec: An optimized tool for learning embeddings of words and entities from wikipedia," *CoRR*, vol. abs/1812.06280, 2018. arXiv: 1812.06280. [Online]. Available: http://arxiv.org/abs/1812.06280. 22, 23

[54] I. Yamada, H. Shindo, H. Takeda, and Y. Takefuji, "Joint learning of the embedding of words and entities for named entity disambiguation," in *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, Y. Goldberg and S. Riezler, Eds., ACL, 2016, pp. 250–259. [Online]. Available: https://www.aclweb.org/anthology/K16-1025/. 22

[55] A. Zaveri, D. Kontokostas, M. A. Sherif, L. Bühmann, M. Morsey, S. Auer, and J. Lehmann, "User-driven quality evaluation of dbpedia," in *I-SEMANTICS 2013 - 9th International Conference on Semantic Systems, ISEM '13, Graz, Austria, September 4-6, 2013*, M. Sabou, E. Blomqvist, T. D. Noia, H. Sack, and T. Pellegrini, Eds., ACM, 2013, pp. 97–104. DOI: 10.1145/2506182.2506195. [Online]. Available: https://doi.org/10.1145/2506182.2506195. 5, 12, 15, 16

[56] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, S. Auer, and P. Hitzler, "Quality assessment methodologies for linked open data," *Submitted to Semantic Web Journal*, 2013. 15, 16

[57] M. Zhang, Y. Zhang, and G. Fu, "End-to-end neural relation extraction with global optimization," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, M. Palmer, R. Hwa, and S. Riedel, Eds., Association for Computational Linguistics, 2017, pp. 1730–1740. [Online]. Available: https://www.aclweb.org/anthology/D17-1182/. 18

[58] Y. Zhao, S. Gao, P. Gallinari, and J. Guo, "Knowledge base completion by learning pairwise-interaction differentiated embeddings," *Data Mining Knowledge Discover*, vol. 29, no. 5, pp. 1486–1504, 2015. DOI: `10.1007/s10618-015-0430-1`. [Online]. Available: `https://doi.org/10.1007/s10618-015-0430-1`. 19

[59] H. Zhou, A. Zouaq, and D. Inkpen, "Dbpedia entity type detection using entity embeddings and n-gram models," in *Knowledge Engineering and Semantic Web - 8th International Conference, KESW 2017, Szczecin, Poland, November 8-10, 2017, Proceedings*, P. Rózewski and C. Lange, Eds., ser. Communications in Computer and Information Science, vol. 786, Springer, 2017, pp. 309–322. DOI: `10.1007/978-3-319-69548-8\_21`. [Online]. Available: `https://doi.org/10.1007/978-3-319-69548-8%5C_21`. 20, 33

Appendix A

Background Material

A.1 Evaluation metrics

The simplest metric that can be used to evaluate a classifier is the accuracy. It represents the percentage of inputs in the test dataset that were correctly classified. Notwithstanding, it is important to take into consideration the frequencies of the individual class labels in the test dataset. Accuracy results usually are not enough for the evaluation and comparison of classification models. Since the number samples in each classes can be unbalanced. Therefore, we also relied in the notions of precision, recall, and a combination of both named F1-Score.

Given the classification results in term of absolute number of true positives, true negatives, false positives, and false negatives (as defined on sections 4.3 and 5.2), the precision, recall, and F1-score of the classifiers can be computed as following:

Precision

The precision of the classification is defined as the percentage of correct error detections over the total entity-type pairs labeled as incorrect, therefore:

$$Precision = \frac{TP}{TP + FP} \tag{A.1}$$

Recall

In our case, recall represents the percentage of incorrect entity-type pairs in the test sample that the classifier correctly identified as incorrect:

$$Recall = \frac{TP}{TP + FN} \tag{A.2}$$

In most cases the relationship between Precision and Recall is inverse and, therefore, the cost of increasing one of these values results in the decrease of the other. The balance of the two measures of performance depends on the specific objectives of the study.

F1-Score

We can combine the precision and recall to give a single score, the F1-score. It is defined to be the harmonic mean of the precision and recall:

$$\frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{A.3}$$

A.2 Comparison of Algorithms: ANOVA

The ANOVA (Analisis of Variance) is a statistical test that can be used to analyze the differences in the means of multiple groups (populations). The inferences about the population means are made by analyzing variance, hence the name of the test. In this section we provide a brief explanation about the test which was adapted from Campelo (2018) [35].

Each sampled value for each evaluation measure (accuracy, precision and recall) can be modeled as:

$$y_{ij} = \underbrace{\mu_i + \epsilon_{ij}}_{\textit{Means model}} = \underbrace{\mu + \tau_i + \epsilon_{ij}}_{\textit{Effect model}} \begin{cases} i = 1, \dots, a \\ j = 1, \dots, n \end{cases}$$

where μ is the overall mean, τ_i represents the effect of the i-th level (in our case each level is one classifier) and ϵ_{ij} is the residual (random error, or unmodeled variability). a and n represent the number of algorithms being tested and number sampled data for each algorithm, respectively.

In the derivation of the statistical test for the existence of differences in the group means, we will employ the effects model, and initially consider a few assumptions about the residuals:

$$y_{ij} = \mu + \tau_i + \epsilon_{ij} \begin{cases} i = 1, \ldots, a \\ j = 1, \ldots, n \end{cases} , \quad \text{with } \epsilon_{ij} \overset{\text{i.i.d.}}{\sim} \mathcal{N}\left(0, \sigma^2\right)$$

If these assumptions are correct, the populations are expected to be distributed as shown on figure A.1

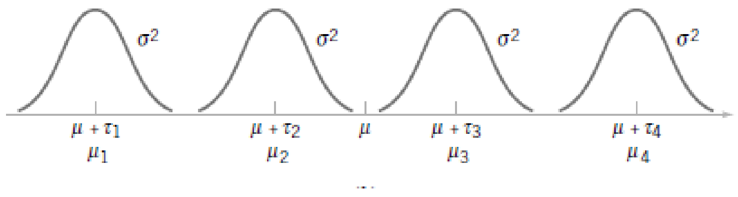


Figure A.1

Since we are interested in testing our data for differences in the mean values of each population, the test hypotheses can be described as:

$$\begin{cases} H_0 : \tau_i = 0, \forall i \in \{1, 2, \ldots, a\} \\ H_1 : \exists \tau_i \neq 0 \end{cases} \tag{A.4}$$

This approach to modeling the mean effects of the algorithm is known as the fixed effects model. If there is no difference between the algorithms, they will perform similarly, therefore the algorithm effect is null:

$$\sum_{i=1}^{a} \tau_i = 0 \tag{A.5}$$

In the derivation of the statistical test for the existence of differences in the group means, we will employ the effects model, and initially consider that the residual is normally distributed ($\mu = 0$ and variance σ^2).

The total variability of the data can be expressed by the *total sum of squares*, which represents the sum of the squared deviations between each observation and the overall sample mean:

$$SS_T = \sum_{i=1}^{a} \sum_{j=1}^{n} \left(y_{ij} - \bar{y}_{\bullet\bullet}\right)^2, \tag{A.6}$$

where $\bullet$ indicates the summation over an index, and $^{-}$ indicates an averaging operation.

The SS_T can be divided into two terms, representing the within-group SS_E, and the between-group $SS_{\scriptscriptstyle Levels}$ variability:

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{n}\left(y_{ij} - \bar{y}_{\bullet\bullet}\right)^2 = \underbrace{n\sum_{i=1}^{a}\left(\bar{y}_{i\bullet} - \bar{y}_{\bullet\bullet}\right)^2}_{SS_{Levels}} + \underbrace{\sum_{i=1}^{a}\sum_{j=1}^{n}\left(y_{ij} - \bar{y}_{i\bullet}\right)^2}_{SS_E} \quad \text{(A.7)}$$

Dividing the sums of squares by their respective number of degrees of freedom yields quantities known as *mean squares.* The relevant means squares for our test will be the *levels mean square* and the *residual mean square*:

$$MS_E = \frac{SS_E}{a\left(n-1\right)} \qquad MS_{\scriptscriptstyle Levels} = \frac{SS_{\scriptscriptstyle Levels}}{a-1} \quad \text{(A.8)}$$

The expected values of these quantities are:

$$E\left[MS_E\right] = \sigma^2 \qquad E\left[MS_{\scriptscriptstyle Levels}\right] = \sigma^2 + \frac{n\sum_{i=1}^{a}\tau_i^2}{a-1} \quad \text{(A.9)}$$

Notice that MS_E is an unbiased estimator for the common variance of the residuals, while $MS_{\scriptscriptstyle Levels}$ is biased by a term that is proportional to the squared values of the τ_i coefficients.

However, under H_0 we have that $\tau_i = 0$ for all i, that is,

$$E\left[MS_{\scriptscriptstyle Levels}\right] = E\left[MS_E\right] = \sigma^2 \quad \text{(A.10)}$$

It can be shown that, if H_0 is true, the statistic $F_0 = \frac{MS_{Levels}}{MS_E}$ is distributed according to an F distribution with $a-1$ degrees of freedom for the numerator and $a(n-1)$ for the denominator. The usual notation is $F_{(a-1),a(n-1)}$.

If H_0 is false, the expected value of $MS_{\scriptscriptstyle Levels}$ is larger than that of MS_E, which results in larger values of F_0 and defines the critical region for our test:

Reject H_0 at the α significance level if $F_0 > F_{\alpha;(a-1),a(n-1)}$

The rejection of the null hypothesis leads to the conclusion that there is at least one level with an effect significantly different from zero, but it does not allow us to conclude which ones are different. To complete the analysis we, therefore, need verify if the assumptions of the test are valid and determine which means are different from which, and by how much.

The ANOVA model is based on three assumptions on the behavior of the residuals: *Independence*, *Homoscedasticity* (i.e., equality of variances across groups), and *Normality*. The residuals of the model can be obtained as:

$$e_{ij} = y_{ij} - \hat{y}_{ij} = y_{ij} - (\hat{\mu} + \hat{\tau}_i) = y_{ij} - \bar{y}_{i\bullet}$$

The normality assumption can be tested using the Shapiro-Wilk test coupled with a normal QQ plot of the residuals. The ANOVA is relatively robust to moderate violations of normality, as long as the other assumptions are verified (or the sample size is large enough). The homoscedasticity assumption can be verified by the Fligner-Killeen test, together with plots of residuals by fitted values. ANOVA is relatively robust to modest violations of homoscedasticity, as far as the sample is balanced. As usual, the independence assumption should be guaranteed (to the best of the experimenter's knowledge) on the design phase, as well as on the analysis. This includes avoiding pseudoreplication and ordering effects, among others.

If the ANOVA assumptions are verified (i.e., if we have solid grounds for trusting the result of the test), we usually need to determine which levels of the factor are significantly different. Pairwise comparisons of the *all vs. all* type is used in this work because we are simply interested in detecting which levels are significantly different from which, without any prior information or special interest in one specific level or ordering. In these cases, the number of comparisons is $K = a(a-1)/2$, where a is the number of levels.

To perform all vs. all multiple comparisons, a common approach is to use Tukey's Honest Significant Difference (HSD) approach. This method provides a slightly higher power than performing multiple t-tests with adjusted values of α.

Appendix B

List of types with samples in the manual evaluation

Following is the list of the 83 types with more than 200 entities which were used for the retrospective evaluation described in section 5.2:

- AcademicJournal
- Actor
- Aircraft
- Airline
- Airport
- Album
- AmericanFootballPlayer
- Animal
- Archipelago
- Automobile
- Award
- Bacteria
- Band
- Bank
- Book
- BroadcastNetwork
- Building
- Casino
- Castle
- City
- Company
- Country
- Currency
- Disease
- Drug
- Event

- FictionalCharacter
- Food
- Galaxy
- GovernmentAgency
- Horse
- HorseTrainer
- Hospital
- Hotel
- Island
- Lake
- Language
- Library
- Locomotive
- MilitaryConflict
- Mountain
- Murderer
- Museum
- Musical
- MusicalArtist
- MusicGenre
- MythologicalFigure
- Newspaper
- Park
- Person
- Planet
- Plant
- Poem
- PoliticalParty
- PopulatedPlace
- President
- Profession
- ProgrammingLanguage
- Publisher
- RailwayLine
- River
- School
- Sea
- Settlement
- Ship
- ShoppingMall
- Software
- Song
- Spacecraft
- SpaceStation
- Species

- Sport
- Stadium
- Station
- Swimmer
- Theatre
- TimePeriod
- Town
- Train
- University
- VideoGame
- Village
- Weapon

Appendix C

Assumptions on the retrospective evaluation of the classifiers

This section lists the assumptions made by the human evaluators during the retrospective evaluation described in sections 4.3 and 5.2. For the remaining types used for the evaluation but not listed in this section, no special assumption was made.

Actor

- Animal actors were not considered to belong to this type.

Award

- Award shows were not considered to belong to this type.

Band

- Rap groups that do not use any musical instruments were not considered to belong to this type.

BroadcastNetwork

- Individual TV channels were not considered to belong to this type.

Casino

- Casino hotels were considered to belong to this type.

- Empires, caliphates, and dynasties were not considered to belong to this type.

Country

- Kingdoms were considered to belong to this type.

Currency

- Cryptocurrencies were considered to belong to this type.

Language

- Human languages and dialects were consider to belong to this type.
- Programming and modeling languages were consider to belong to this type.

PoliticalParty

- Alliances and coalitions were not consider to belong to this type.

ProgrammingLanguage

- Frameworks, software libraries, extensions, algorithms, and paradigms were not consider to belong to this type.

Publisher

- Any organization that publishes something (video games, softwares, newspapers, magazines, books, papers, etc.) were consider to belong to this type.

River

- Streams and creeks were consider to belong to this type.
- Arroyos and springs were not consider to belong to this type.

Settlement

- Any place where a group of people live was consider to belong to this type (towns, city, populated island, village, township, etc.).

Ships

- Boats were also consider to belong to this type, since the distinction between boats and ships is not always clear. Moreover, the type *boat* does not exist on DBpedia, therefore, this is probably the best type to assign boats to.
- Submarines and U-boats were not considered as ships.

Shopping Mall

- Flea markets, outdoor markets, plazas, outdoor pedestrian malls, outlet malls, and large stores were not considered to belong to this type.

Software

- Computer games and mobile applications were considered to belong to this type.

Spacecraft

- Satallites, rovers and probes were considered to belong to this type.
- Space station modules were not consider spacecrafts.

Stadium

- Areas and baseball venues were also considered as stadiums.

Stations

- Public transport station (eg. railway station, metro station, bus station) were considered to belong to this type.
- Broadcast stations (radio and television) were considered to belong to this type.
- Power stations(dams) were considered to belong to this type.

- Military airports were not considered to belong to this type.

Theater

- Operas were also consider as theaters.
- We excluded from our evaluation buildings that contains theaters but also contains other services (i.e., we did not judge these cases).
- Theater companies were not consider theaters.

Town

- Townships were not accepted in this type.

Train

- Both named trains and trains services were consider to belong to this type.
- Electric multiple units and diesel multiple unit were considers trains.

Villages

- Only resources whose description explicitly says that they are villages were considered to belong to this type.

Weapons

- Tanks and artillery vehicles were considered weapons.