Université de Tunis

Institut Supérieur de Gestion

# MASTER RECHERCHE

Spécialité

Sciences et Techniques de l'Informatique Dcisionnelle (STID)

Option

Informatique et Gestion de la Connaissance (IGC)

---

# Automatic large-scale Knowledge Graphs Refinement based on machine learning techniques

---

Nouha Thabet

Nadya Yacoubi Ayadi Maitre Assistant, ISG Tunis Directeur du méemoire

---

Laboratoire RIADI - ENSI

# Acknowledgments

First, I want to start by thanking God for all the help, the strength and the ability to establish this work during my master thesis. Without his blessings, this research study could not be possible and I could not complete it until the end.

I would like to express my sincere gratitude to Pr. Nadia Yaacoubi who gaved me the opportunity to join the RIADI laboratory. I am thankful for her support, guidance and valuables advice.

My greatest and deepest gratitude go to my family for all the support, the continuous encouragement and the overwhelming love that filled me with strength and patience. This accomplishment would not have been possible without them.

I extend my gratitude to my friends and everyone who contributed in the success of this work.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Introduction

Semantic Web knowledge graphs are a structured data and present a source of information for many intelligent systems. However many search engines now use them to enrich search results with structured information also several question answering systems convert a question into some kind of query against a knowledge base and finally information extraction systems will often use a knowledge base as their main source of training data. They are constructed from collections of triples, about people, things, and places in the world, and relationships between them. The data are also stored in a fully machine-readable format that allows many differents application to get the information from the same knowledge base (KB) without any format problem.

Many semantic web knowledge base have present a large-scale crossdomain datasets such as DBpedia, Nell and YAGO. Generating a such structured large dataset is a challenging task. However, we cannot guarantee a knowledge graph which is free of noise and incompleteness. As a result, noise presents the erroneous triples in a KB and incompleteness presents the missing types of entities and the missing links between them. Indeed, it's too hard to detect the noise or the incompleteness manually on a tens or hundreds millions of triples.

The process of reducing incompleteness and noise is called refinement. Developing automatic knowledge graph refinement methods which can do that on a large scale is a good way to reduce these problems there with improving the quality of the data. By improving the quality and coverage of a knowledge graph, many applications which use KBs can indirectly profit from such improvement.

In the literature, there are two majors categories of refinement methods:T-Box refinement and A-Box refinement. T-Box refinement rely on the schema or the ontology and also known as ontology learning while A-Box refinement rely on the TBox-compliant statements.

Many ABox refinement methods do not use TBox information, relying exclusively on the relations between instances. However, sometimes KB lack from schema incompletness. Type prediction, link prediction and error detection are problems of the A-Box refinement in knowledge graphs.

The main objective of this thesis is to develop and improve methods of knowledge graph refinement. We focus on both type prediction and error detection and we evaluate our proposed methods on a large set of popular available knowledge graphs.

Our master thesis is organized into two focal parts:
Part I denotes the theoretical aspects that support the concepts used in our work besides the literature review for related works. This part is contain two chapters presented as follows:

- Chapter 1: presents the semantic web and its components as well as the semantic web knowledge graphs and their issues.
- Chapter 2: defines previous works related to the reduction of the noise and incompleteness.

Part II details our new approaches to reduce incompleteness and noise. This part is decomposed into two main chapters:

- Chapter 3: details our approaches for type prediction and error detection.
- Chapter 4: presents the execution and the experimentation that have been done in the purpose of evaluating the results given by our proposed methods and to compare it with another method.

Finally, a conclusion will summarize our work and will propose future works to improve our methods.

Chapter 1

# Fundamentals and Technicals Background

## Introduction

The increasing diffusion of Linked Open Data (LOD) practices as a standard way to share knowledge in the Semantic Web by different data providers has generated a large number of interconnected datasets which comprise an unprecedented volume of data, commonly represented in Resource Description Framework (RDF). They are named dataset or "Knowledge graph" by Google in 2012, referring to their use of semantic knowledge in Web Search. the number of datasets has increased from 12 datasets in 2007 to 570 datasets in 2014 to end with 1224 interlinked datasets. The number of links across datasets is estimated to 16,133 links in 2018 which allows to build diverse applications such as NL question processing, movies/news recommendation.
This chapter focuses on presenting basic concepts of semantic web and Linked Data. In section 1.2 we introduce the world wide web and why we moved to the semantic web, in section 1.3 and 1.4 we introduce respectively the linked data and the linked open data. we presents the semantic web standards in Sections 1.5, 1.6 and 1.7 which are respectively Resource description framework RDF, Ontology and SPARSQL. We defined the knowledge graph database in section 1.8 and finally we discussed quality issues in Linked Data and knowledge graphs respectively in section 1.9 and 1.10.

## 1.1 The World Wide Web and the Semantic Web

The world wide web, also known by WWW is a common place to share information around the world. It contains a distribution network of web pages that are identified and connected to each others by a global links called Uniform Resource Locators (URLs) through a protocol such as the application level protocol Hypertext Transfer Protocol (HTTP). The content of the world wide web which is understandable by all machines is represented by the Hyper Text Markup Language HTML. Basically, the common formats of publishing data on the web are CSV, XML, or HTML tables and links between web pages are not semantically processable by machines. In other words, if we are interested to know the number of people that have received a Nobel Prize award in physics. We can't get this information from the web only in two cases. The first one if someone have published the result on the web, the second one if there is an available data on the web that have a structured format and can be processed offline. Getting a such information is not guarantee in the two cases and machines must be able to semantically understand data published on the web.
The semantic web is an extension of the world wide web, but it supports the meaning rather the structure of data. Instead of linking web pages like the WWW, it links a data to a specific another data contained in that web page and thus by using global references called Uniform Resource Identifiers (URIs). Moreover, the distributed web of data is presented by a data model called the Resource Description Framework (RDF). The Semantic Web is based on different standards such as RDF, OWL and SPARQL that we will discuss them in the next sections. Furthermore it is represented as Semantic Web Layer Cake as shown in Figure 1. Each layer represents a technical part for its construction and make it as a machine-readable Web that presents the data on the World Wide Web or as a globally linked database.

## 1.2 Linked Data

The term Linked Data coined by the director of the World Wide Web Consortium (W3C) Tim Berners-Lee in a 2006 note about the semantic web project. It refers to a style of publishing and creating typed links between strutured data from different sources on the web. Linked data extends standard web technologies to share information in a way that can be read automatically by computers and thus

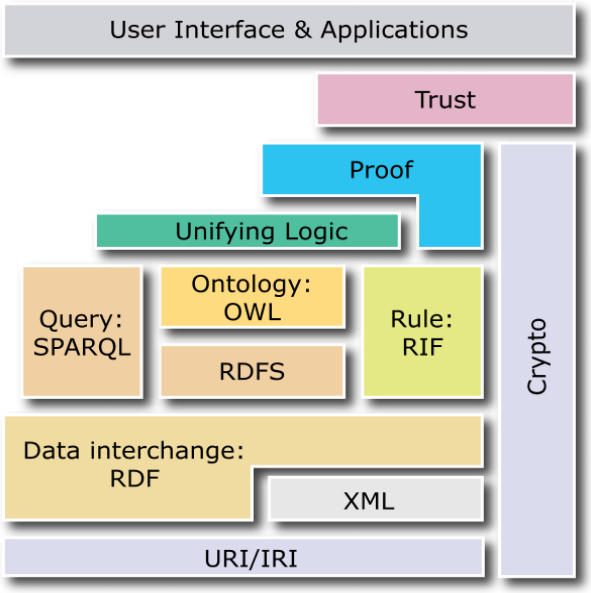


Figure 1.1: Semantic Web Layer Cake

by using RDF to make typed statements that link arbitrary things in the world. A set of rules was defined by Tim Berners-Lee for publishing data on the Web and make it part of a single global data space:

- Use URIs as names for things
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
- Include links to other URIs. so that they can discover more things.

## 1.3 Linked Open Data

Linked Data principles was rapidly adopted and applied in the Linked Open Data Project. This project founded in January 2007 and supported by the W3C Semantic Web Education and Outreach Group. It aims to extend the Web by identifying existing open data sets, converting these to RDF according to the Linked Data principles, publishing them on the Web. and setting RDF links between data

items from different data sources. At the begin, only University research labs and small companies has contributed in the project. Since then, large organisations such as the BBC and Thomson Reuters has participated in the project. However the number of linked open datasets available increased 112 times in twelve years, passed from twelve data sets on 2017 to 1,234 on 2019.Figure 1 and figure 2 shows Linked open Data Cloud diagram respectively in 2007 and in 2019.

Each arc in the two figures refers to the existance of links that connect data in two data sets. Although we notice that there are different shapes of arcs. For example in Figure 1 the arc between DBpedia data set and Geonames dataset is heavy and that means there are greater number of links between the two data sets. Also there is a bidirectional arc between DBpedia and Revyu and this indicates that the outward links to the other exist in the two data set. The dataset currently contains 1,234 datasets with 16,136 links.

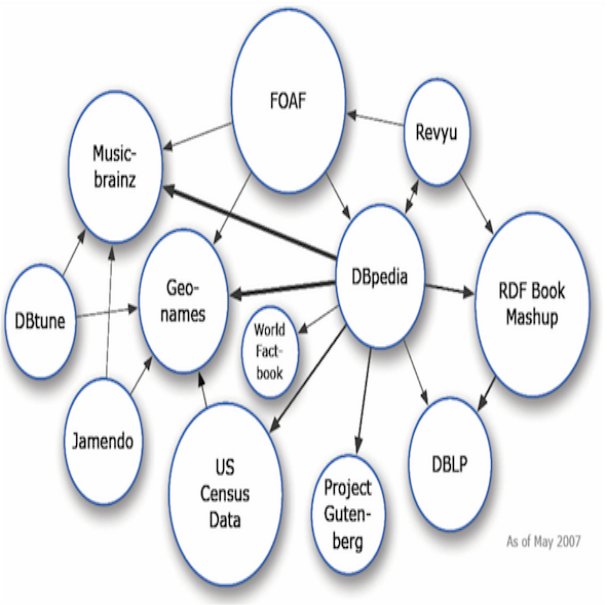


Figure 1.2: The Linking Open Data cloud diagram on 2007

## 1.4 Resource Description Framework

Information is expressed in a standard model to enable applications to process data on the web. This model is the Resource Description Framework RDF, which

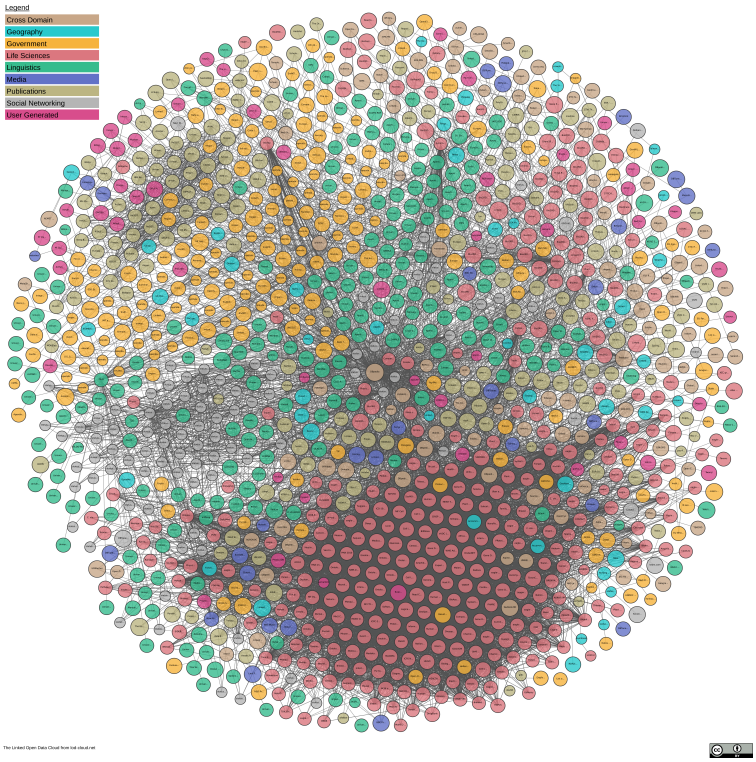


Figure 1.3: The Linking Open Data cloud diagram on 2019

provide an interoperability between these applications. The main goal of RDF is to define a mechanism for describing resources independent of the application domain and allow the exchange of information between applications without loss of meaning. It's a part of W3C recommendation. (?, ?).
The basic model of RDF has three elements such as statement, three types for the statement term and different serialization formats.

### 1.4.1 RDF basic elements

RDF data model has three basic elements which are resource, property and statement.

- Resource : Resources represent things expressed by RDF. A resource could be an entire page web, a specific element within the page web, an entire website or an object that is not accessible on the web like a historical monument.
- Property : Properties are used to describe resources. Each property has a

specific meaning and each one could be a specific aspect, a characteristic, an attribute or a relation.

- Statement : A statement is an RDF triple which is composed from a subject, a predicate and an object. The subject is a resource, the predicate is a named property and the object which is a value of that named property. A statement is also called a triple and has the following format:
  $< resource(subject) >< property(predicate) >< propertyvalue(object) >$
  .

### 1.4.2 RDF Statement Types

As explained previously a statement is a triple containing three RDF terms in the form of $subject - predicate - object$. There are three types of an RDF term. It can be an URI, a literal or a blank node. According to RDF standard the URI can be used in the three RDF terms, the literal can be used only as an object and the blank node can be used as subject or object in an RDF triple. The RDF terms could also be presented as a graph. RDF graph is a set of RDF Triple. A graph is composed by nodes and edges. The set of nodes are the subjects and the objects and the set of edges are the predicates. We will discuss later knowledge graphs. As URI was discussed in previous sections, we now discuss literal and blank node.

- A Literal could be a plain literal or a typed literal and both of them presents a literal value like a string, number or a date. The plain literal is a string of the form ”abc@langTag” where ”abc” is an arbitrary (possibly empty) string, and ”langTag” is either the empty string or a language tag such as English or French. For example ”Lisbon”@en indicates the language tag English. However the typed literal is a string combined with a datatype URI. A datatype URI is defined by the XML schema and indicates dates, integers and floating point numbers, e.g. ”1994 – 08 – 17”$d$xsd:date.

- The RDF terms could be represented as a graph where subject and object are nodes and the predicate is a directed vertex between them, we will discuss later graphs representation. A blank node is used to denote an anonymous resource which is a resource without URI or literal. This is happen in documents that contain RDF description and cannot be referenced outside of their originating scope.

There are two types of RDF triples. When the object is of type literal the triple

is called *literal triples*. However when the object is of type URI the triple is called *RDF links*. In the second type, we can have link between a subject and an object that belong to different data sets as discussed in section 1.4.

### 1.4.3 RDF Serialization Formats

As discussed previously, RDF data are published and exchanged between information systems on the web. For this reason there is a need for a defined syntax and thus by serializing RDF data. There are several formats for serializing RDF RDF/XML, N-Triples , N3, and Terse RDF Triple Language (Turtle). In this thesis we will use Turtle to present RDF triples.
A turtle file have a .ttl extension and it's composed from a sequence of directives, a triple generating statement or blank lines and cpmments are represented by lines which begin by #.

## 1.5 Ontology

Data on the web are published and stored into many databases and most of time there is a need to the machine to compare or combine information from two or more databases. However, two different databases may presents the same information but with different identifiers. In this case the machine needs to know that two data may have the same meaning but not the same identifiers. A solution to this problem is to use ontologies.According to W3C, an ontology defines the terms used to describe and represent an area of knowledge. Basically, the main goal of ontologies is to encode knowledge in a domain and make it reusable to be shared applications. For example in the semantic web it represent the semantics of documents and enable humains and machines to interpret the meaning of the exchanged data.
Each web ontology cannot present all knowledge areas but only a specific domain such as medecine or education. In addition to that it contains classes, also called concepts, and relationships between them. The relationships are used to indicate the hierarchy between classes or to describe various features and attributes of each one. Therefore, ontology represents a set of classes (e.g "Person", "Scientist","Award"), and relationships to indicates hierarchy (e.g. The "Scientist" is a subclass of "Person") and finally a relationship could present a property (e.g.

A ”Scientist” have an award of type ”Award”). Figure 3 explain an example of ontology.

Therefore web ontologies use a web ontology language such as OWL and RDFS. we will discuss both of them in the next subsections.

### 1.5.1 RDFS

RDF Schema is the semantic representation of RDF. It creates vocabulary to describe RDF resources and classified them as classes or properties and a W3C recommendation.

Classes are themselves resources and ofen identified by URIs. There are different classes and each one has a set of resources called instances. Each instance could belong to one or many classes and therefore a resource could have many types. A class may be described using RDF properties. For example the property rdf:type is used to indicate that a resource is an instance of a class (e.g. BMW is an instance of the class ”automobile”).

In addition to that, RDF Schema defines relationships between classes and others between properties. For example relationships between classes is provided throught $rdfs : class$ to represent resources that are RDF classes, rdfs:subClassOf to state that a class is a subclass of another and rdfs:Resource to describe all things described by RDF. Relationships between properties are presented by rdf:property to present properties that are RDF properties, rdf:subPropertyOf to indicate that one property is a subproperty of another, rdfs:range to restrict which classes can be object of a property in an RDF triple and finally rdfs:domain to restrict which classes can be subject of a property in an RDF triple. Many other details about RDFS are described in reference.

### 1.5.2 OWL

The web ontology language OWL is also a semantic language for publishing ontology on the web and a W3C recommendation. It’s an extension of the RDFS but it’s more expressive and works with complex relationships between classes or properties. For example if we need to work with two datasets with two dif-

ferent ontologies in an application, we can find classes with different URIs but present the same thing. For instance the URI ”http://schema.org/Place” and the URI ”http://dbpedia.org/ontology/Place” are different but both of them present the class ”Place”. In this case OWL helps to indicate that they state for the same class using owl:sameAs. Many other details about OWL are described in reference.

## 1.6 SPARQL

SPARQL is the query language of RDF and a recommendation of W3C. The SPARQL language is similar to the SQL language. The only diference that SQL operates on relational database and SPARQL operates on RDF graph. A SPRQL qury can have four forms :

- The SELECT form like in SQL it returns variables and their bindings directly.
- The CONSTRUCT form returns an RDF graph formed by taking each query solution in the solution sequence, substituting for the variables in the graph template, and combining the triples into a single RDF graph by set union.
- The ASK form is used to state if a query has a solution or not.
- The DESCRIBE form returns a graph that contain data about each resource identified in the solution.

In addition to that a SPARQL query consists of triple patterns, conjunctions , disjunctions (logical “or”) and a set of optional patterns such as FILTER to restrict the solutions of a graph.
Now let takes an example of SPARQL query and answer to the question in section ...

## 1.7 Knowledge Graphs

In this section, we focus on knowledge graphs as a graph-based knowledge representation formalism adopting semantic Web vision and linked data principles. In section 1.8.1, we present examples of some large-scale knowledge graphs such as DBpedia and Wikidata.

### 1.7.1 Overview

All terms defined in the previous sections contribute in the creation of a knowledge graph. The term Knowledge Graph was coined by google in 2012 and has been recently also used to refer to Semantic Web knowledge bases. In section 1.5 we have state that an RDF triple has also a graph representation. However the knowledge graph is a set of entities in the form of nodes that are connected by relations between them. The set of relations are the edges of the graph. In addition to that, statements of KG are devided into A-Box statements and T-Box statements. A-Box statements are defined by RDF and the T-Box statements are defined by OWL and RDFS. For example "every scientist is a person" is a T-Box statement however "Albert Einshtein is a scientist" is an A-Box statement.

### 1.7.2 Examples of Well-known Knowledge Graphs and their construction

Knowledge graphs cannot be created manually because of their magnititude. They are often constructed from unstructured or semi-structured knowledge, such as Wikipedia, or harvested from the web with a combination of statistical and linguistic methods to transform data to RDF triples. In this section we present some well-known knowledge graph and how they were constructed.

- DBpedia : DBpedia is the central hub of LOD cloud. It extracts structured information from wikipedia and convert it into multilingual Knowledge Base. A wikipedia page is constructed from free text and also from wiki-markup to identify infobox templates, external pages links, images etc. Infoboxes are the property summarizing tables found on most of the Wikipedia pages. Types of the Wikipedia infoboxes are mapped to the DBpedia ontology, and the keys used in those infoboxes are mapped to properties in the DBpedia ontology. Based on those mappings, the DBpedia knowledge graph is extracted . The english version of Dbpedia contains 4.58 million things which 4.22 million are classified in consistent ontology. There are localized versions of Dbpedia distributed in 125 languages. All these versions describe 38.3 million things and Altogether the DBpedia 2014 release consists of 3 billion pieces of information (RDF triples) out of which 580 million were extracted from the English edition of Wikipedia, 2.46 billion were extracted from other language editions.

- YAGO : Yet Another Great Ontology (YAGO) extracts information from wikipedia ans WordNet. Instead of using information extraction method, YAGO extracts infobox information and category information from wikipedia. The ontology of this knowledge graph is built from the category system in Wikipedia and the lexical resource WordNet, with infobox properties manually mapped to a fixed set of attributes. YAGO has knowledge of more than 10 million entities (like persons, organizations, cities, etc.) and contains more than 120 million facts about these entities.

- Wikidata : is a multilingual, community-based Knowledge Base introduced by the Wikimedia organisation. It provides structured information to Wikipedia. Data is entered and maintained by crowdsourced editors as well as automated bots. After the shutdown of Freebase, the data contained in Freebase was subsequently moved to Wikidata.

| Dataset | DBpedia | YAGO | Wikidata |
|---|---|---|---|
| Version | 2016-04 | YAGO3 | 2016-08-01 |
| instances | 5 109 890 | 5 130 031 | 17 581 152 |
| facts | 397 831 457 | 1 435 808 056 | 1 633 309 138 |
| classes | 754 | 576 331 | 30 765 |
| relations | 3555 | 93 659 | 11 053 |
| AVG(indegree) | 13.52 | 17.44 | 9.83 |
| AVG(outdegree) | 47.55 | 101.86 | 41.25 |

## 1.8 Quality Issues in Linked Data

The quality of knowledge graphs was investigated in several research works through different perspectives. Zaveri et al. () propose a comprehensive conceptual framework for Linked Data Quality Assessment (LDQA) by synthesizing the quality problems in linked data through 18 quality dimensions. Even more, authors propose 64 quality metrics according to each dimension.

In the same research work, Zaveri et al. propose a Triple CheckMate which is a crowdsourcing tool that comprises a manual and a semi-automatic process. The manual process is the first phase that corresponds to the detection of common quality problems and their representation in a quality problem taxonomy. The second phase comprises of the evaluation of a large number of individual resources,

according to the quality problem taxonomy via crowdsourcing. Although, Zaveri's approach covers a large scope of quality problems, the crowdsoucing tool requires user involvement to detect errors.
Other researchers focus on the most common quality problems which are incompletness and noise. In this master thesis, we focus on (semi-)automatic methods for error detection and type prediction in knowledge graphs.

## 1.9 Incompleteness and Noise In Knowledge Graphs

Incompleteness and noise are two major problems of the knowledge graph. Incompleteness refers to missing entities in the graph, missing types for entities and missing relations, whereas noise refers to faulty statements in the graph.
Noisy data in knowledge bases have two main sources: The first one is the extraction process when erroneous statements are generated from correct natural language inputs. The second one refers to the existing errors in the data source just like faulty inputs in the Wikipedia infobox. Manual evaluation on samples of data has demonstrate that YAGO and DBpedia has respectively 12% and 18% error statements while Wikipedia has 4.7% faulty statements. In the other hand it's too complicate to estimate incompleteness in a knowledge base. However, we can demonstrate it by the fact that only 2% of people have father in the Wikidata Graph Database and only 15.87
Decreasing the noise and the incompleteness problems in KG is too complicate especially with datasets which contains millions of triples. In the next subsection we will give an overview on some automatic refinement approaches on A-Box.

### 1.9.1 Internal and External Methods

There are two kinds of methods: the internal methods and the external methods. The former uses the data of the knowledge base and the latter uses external data such as other knowledge graphs that are linking to the actual one, text corpora or crowdsourcing. This kind of methods is not really automatic while internal methods can be automatic and also uses graph features. There are two kind of feature: the latent features and the observed features and we will discuss them in the next section.

### 1.9.2 Latent and Observed Features

Observed feature as indicated are directly observed in the graph. The idea behind using observed feature is to identified similar entities by checking their common neighborhood of node and paths between nodes. In the other hand latent feature are not directly observed in the graph.

## 1.10 Evaluation of Knowledge Graph Refinement Methods

There are three main kinds of evaluation approaches for ABox automatic refinement (149): partial gold standard, silver standard and retrospective evaluation.

- **Partial Gold Standard:** In this evaluation methodology a subset of graph entities or relations are selected and labeled manually. Other evaluations use external data as partial gold standards. For completion tasks, this means that facts that should exist in the knowledge graph are collected, whereas for correction tasks, a set of facts in the graph is manually labeled as correct or incorrect. Crafting a gold standard can be extremely costly, and the fact that the gold standard is created for a relatively small sample can significantly affect the quality of the evaluation.

- **Silver Standard :** Another evaluation strategy is to use the given knowledge graph itself as a test dataset. For completion, that means a subset of the knowledge graph is removed from the training dataset and used as test. For error detection, wrong facts can be generated and added to the knowledge graph, then later be used for reference in the evaluation. Since the knowledge graph is not perfect, it cannot be considered as a gold standard, therefore we call it a silver standard. However, assuming that the given knowledge graph is already of reasonable quality, the evaluation results should be a good approximation of the actual results, with the advantage of being fully automatable.

- **Retrospective Evaluation:** For retrospective evaluations, the output of a given completions or identified errors as correct and incorrect. The quality

metric is usually accuracy or precision, along with a statement about the total number of completions or errors found with the approach, and ideally also with a statement about the agreement of the human judges. Recall and other measures which rely on it cannot be calculated since the number of false positives cannot be calculated. One advantage of retrospective evaluations is that they allow a very detailed analysis of an approach's results. However the annotations are specific for a given method and cannot be reused.

## 1.11 Conclusion

In this chapter we introduced the Linked Data and its fundamental components such as RDF, OWL and SPARQL. We defined the knowledge base with some example of KG and their construction. Finally, we discussed the quality issues in KG and introduced some methods.
In the next chapter we will focus on the problems of type prediction and error detection in a knowledge graph and we will present different methods that decrease the incompleteness and detect errors in a KG.

Chapter 2

# Related Works

## 2.1 Introduction

In the first chapter we talked about issues in knowledge graphs which are not free of errors and missing types. In this chapter we present previous works related to issues in knowledge graphs quality. We will first discuss work about type prediction in section 2.2 then discuss error detection in section 2.3.

## 2.2 Type Prediction

Every resource in a knowledge base can have one or more classes but many knowledge graphs suffer from the lack of this important information. For example 37.3% of DBpedia resources are untyped. Reasoning can be a solution to infer type information on the semantic web. But knowledge graphs are often noisy, and reasoning on noisy data can increase the error rate and then damage the quality of the data. This problem was identified by Ji et al. Many other solutions were proposed to solve the problem of knowledge graph completness. These solutions use machine learning techniques and statistical approaches. We will only discuss the SDType algorithm which has the best results and the most used to complete missing types in the DBpedia website.

### 2.2.1 SDType

SDType is an algorithm is a statistical approaches that uses ingoing and outgoing property of a resource and statistical distribution for each property to assign types for untyped resources. The statistical distribution of each property is the number of occurance of each type appearing as subject instance and object instance for that property. Let take as example the property *dbpedia-owl:location* as shown in Table 2.1.

| Type | Subject % | Object % |
|---|---|---|
| owl:Thing | 100.0 | 88.6 |
| dbpedia-owl:Place | 69.8 | 87.6 |
| dbpedia-owl:PopulatedPlace | 0.0 | 84.7 |
| dbpedia-owl:ArchitecturalStructure | 50.7 | 0.0 |
| dbpedia-owl:Settlement | 0.0 | 50.6 |
| dbpedia-owl:Building | 34.0 | 0.0 |
| dbpedia-owl:Organization | 29.1 | 0.0 |
| dbpedia-owl:City | 0.0 | 24.2 |
| ... | ... | ... |

Table 2.1: Distribution of subject and object types for the property *dbpedia-owl:location*

The percentage value of subject and object of each type cannot sum up to 100% because every resource can have multiple types.

We can assign probabilities from this table. For example if we have a resource x that appears as a subject with the predicate *dbpedia-owl:location*, then we can extract the probability P(:x a dbpedia-owl:Place) = 0.698. More formally, given a resource with a property *prop* that may be an ingoing property or an outgoing property, the conditional probability to measure how a type $t$ is expressed by

$$P(t|(\exists prop.T))$$

Using only statistical distributions can avoid problems of false type prediction with knowledge bases in which the extension of some types are several orders of magnitude larger than that of others. For this reason SDType compute weight $w_{prop}$ for each property which reflects its predictive power and measure the deviation of the property's distribution to the apriori distribution of all types in the knowledge base. The stronger this deviation, the higher there is assetion of the property's predictive power. The weight may not has the same value when the property is an ingoing property or an outgoing property because of the

statistical distributions as shown above. For example with the predicate *dbpedia-owl:location* there are two weights $w_{dbpedia-owl:location}$ and $w_{dbpedia-owl:location^{-1}}$.
Both statistical distributions per property as well as the apriori probabilities P(t) are used to calculate the weight wich has the following formula :

$$w_{prop} := \sum_{alltypest} (P(t) - P(t|(\exists prop.T)))^2$$

The conditional probabilities and the property weight are used to implement a weighted voting approach. This for assigning a likelihood to each type appearing in the property's statistical distribution. Then the overall the the weighted sum of all likelihoods presents the overall predicted type distribution for a resource and has the following formula :

$$\sum_{allpropertiespropofr} P(t(r)|(\exists prop.T)(r))$$

Finally the type is choosen after applying a confidence threshold.

## 2.3 Error Detection Approaches

The error detection problem in knowledge base was researched by semantic web community and classified as a hard problem. Zaveri et al. claim that this problem cannot be au¬tomatized. Ontology reasoning has the ability to detect conflicts or error in knowledge graphs but this requires a rich ontology which is difficult to find it. In fact, the ontology of knowledge bases suffers from the lack of domain and range restrictions and this presents the main cause of such problem. Only few methods were proposed as solutions for the error detection problem.

### 2.3.1 Ontology Enrichment

Enriching the ontology was an approach to detect errors. Töpper et al. (2012) have enriched DBpedia ontology by adding others domain and range restrictions and disjointness axioms. Then they used the resulted ontology for error detection and detect around 60,000 inconsistent statements. But they conclude that in most cases the cases the identified statement itself is actually correct, whereas the ontology should be altered.

### 2.3.2 Crowdsourcing Approach

Using external knowledge from external data sources is another approach to validate statements. Acosta et al. (2013) used crowdsourcing to handle with the quality of Linked Data. In this approach, the common errors in Linked Data source were analyzed and classified into the following error classes: wrong literal values, wrong literal datatypes, and wrong interlinks to other datasets. This classification helps to specify data that have a similar form of crowdsourcing. This elaborated via a paid micro-tasks like the validation of a statement, published by users on Amazon Mechanical Turk. The authors reported maximum precision of 0.90, 0.83, and 0.94 for the three classes which is impressing. However, this approach hasn't a good scalability. They report that the evaluation of 1,037 statements on Amazon Mechanical Turk could be validated in four days which mean that the whole DBpedia knowledge base would take more than 3,000 years to be evaluated.

### 2.3.3 knowledge graph completion (KGC)

Knowledge graph completion (KGC) or link predication methods can also be used on the error detection problem. These methods can be devided into two types using two different kind of graph features. The first type presents the graph-based methods which relies on the observed features like the paths between two nodes of the graph and the second one presents the embedding methods and relies on the latent features which learn continuous representations for entities and relations. We will discuss two approaches of KGC.

#### Path Ranking Algorithm

The Path Ranking Algorithm (PRA) used in the task of automatically infer missing facts from existing ones. It was first introduced at first by Lao and Cohen, 2010, and later slightly modified in various ways (Gardner et al., 2014; Gardner and Mitchell, 2015). The idea is based on the use of paths that connect two entities as feature to predict potential relations between them. The path is equivalent to a sequence of relation $< r1, r2, \ldots, rn >$ between two entities. For example, ( bornIn , capitalOf ) is a path linking SophieMarceau to France , through an intermediate node Paris . Such paths are then used as features to predict the presence of specific relations, e.g., nationality. There are three steps in PRA: feature extraction, feature computation, and relation-specific classification.

The first step is to select path features that are potentially useful for predicting new relation instances. Thus by encoding KG as multi-relation graph. Given a pair of entities (s, o), PRA then finds the paths by performing random walks over the graph, recording those starting from s and ending at o with bounded lengths. More exhaustive strategies like breadth-first (Gardner and Mitchell, 2015) or depth-first (Shi and Weninger, 2015) search could also be used to enumerate the paths. After that a set of paths are selected as features, according to some precision-recall measure (Lao et al., 2011), or simply frequency (Gardner et al., 2014).

Once path features are selected, the next step is to compute their values. Given an entity pair (s, o) and a path $\pi$, PRA computes the feature value as a random walk probability p(o—s, $\pi$), i.e., the probability of arriving at o given a random walk starting from s and following exactly all relations in $\pi$. Computing these random walk probabilities could be at great expense. Gardner and Mitchell (2015) recently showed that such probabilities offer no discernible benefits. So they just used a binary value to indicate the presence or absence of each path. Similarly, Shi and Weninger (2015) used the frequency of a path as its feature value. Besides paths, other features such as path bigrams and vector space similarities could also be incorporated (Gardner et al., 2014).

The last step of PRA is to train an individual classifier for each relation, so as to judge whether two entities should be linked by that relation. Given a relation and a set of training instances (i.e., pairs of entities that are linked by the relation or not, with features selected and computed as above), one can use any kind of classifier to train a model. Most previous work simply chooses logistic regression.
The computation of random walk probabilities associated with each path type and node pair is very intensive, requiring time proportional to the average out-degree of the graph to the power of the path length for each cell in the computed feature matrix.

To this end a new way of generating feature matrices over node pairs in a graph was proposed to improve both the efficiency and the expressivity of the model relative to PRA. It's the subgraph feature extraction.

#### Subgraph Feature Extraction

As discussed before SFE in an improvement of PRA, It was introduced by Gardner and Mitchell (2015) SFE has the same first step of PRA but the second step was improved. The first step of PRA does a series of random walks from each subject and object node. In PRA these random walks are used to find a relatively small set of potentially useful path types for which more specific random walk probabilities are then computed, at great expense. In our method, subgraph feature extraction (SFE), we stop after this first set of random walks and instead construct a binary feature matrix.

#### Knowledge Graph Embedding Models

Knowledge graphs embedding models was useful for knowledge graph completion. Several models were discussed in the literature. RESCAL (Maximilian Nickel et al. 2011) is the first knowledge graph embedding model, it uses tensor factorization on the adjacency tensor of the knowledge graph. However, each pair of entities is represented via the tensor product of their embeddings. The strength of this method figure on its ability to capture complex relational patterns over multiple hops but its quadratic runtime and memory complexity with regard to the embedding dimension enable it to scale to very large knowledge-graphs. RESCAL was improved to TRESCAL which used ontology informations like type and domain and range restrictions to speed up the tensor factorization process.
Other methods represents relations between subject and object as translations. The earliest translation-based embeddings model was TransE which predict missing relationships on a learned low-dimensional embedding of the knowledge graph entities. PTransE (110) extends TransE by considering relation paths as regular relations, which makes the number of relations considered grow exponentially.

### 2.3.4 SDValidate

SDValidate (H.Paulheim and C.Bizer, 2014) is an algorithm that exploit statistical distributions of properties and types to identify faulty statements based of feature type of the objects statements. Thus according to Zaveri et al. (2013), who claim accuracy problems in DBpedia are highly related to the objects being incorrectly or incompletely extracted. The basic idea is to specify the deviation of the types predicted by the statement from the statement's object's actual types set in the knowledge base based on confidence score calculated for each statement.

SDValidate identify statement by first calculating the relative predicate frequency (RPF), assigning a score to each of the selected statements and finally applying a threshold to identify error statements.

In the first step is computing RPF which indicates the frequence of a predicate/object combination in the graph. The RPF has the following formula for a statement*s*:

$$RPF(s) := P(pred|obj) = \frac{|\text{statements with pred}(s) \wedge \text{obj}(s)|}{|\text{statements with obj}(s)|}$$

The main idea behind this approach is that statements with frequent predicate/object combination are more likely to be correct than statements with unfrequent predicate/object combination.

In the second step the score is assigned to each selected statement using the statistical distribution of predicates and types. SDValidate uses vectors that contains probabilities of the property's object and subject and compare them to the respective resources' actual types. The confidence score of the statement is calculated using the cosine similarity of the two vectors. Since objects are more likely to be wrong, the algorithm focuses on the erroneous objects by comparing the distribution for the object type of the predicate with a statement's object's types. The formula of true confidence statement is:

$$conf(s) := \frac{\sum_{\text{all types t}} p(t|prop^{-1}).d(t,o)}{\sqrt{\sum_{\text{all types t}} p(t|prop1)^2}.\sqrt{\sum_{\text{all types t}} d(t,o)^2}}$$

Where prop is a predicate, s is a statement, o is an object and d(t,o) has a binary value which denotes if a resource o has a type or not.

In the last step a threshold is applied to indicate whatever a statement is correct or erroneous.

## 2.4 Conclusion

In this chapter we introduced SDType for type prediction problem and methods for error detection problem. SDType use statistical distribution to assign types for

untyped resources and reported interesting results. For error detection, we defined ontology enrichment method which didn't report good results because many statements identified were semantically correct but they miss ontology informations. The second method is the crowdsourcing approach which has a good accuracy but isn't scalable. The third method is to use methods of knowledge graph completion to detect errors. And the last one is the algorithm SDValidate for error detection which use statistical distribution and detects error statements as outliers. In this master thesis we're going to improve the results of SDValidate.

Chapter 3

# Contribution

## Introduction

In this Chapter, we aim to improve the results of SDValidate by using a well known clustering technique. We will use the types of entities and paths between them as features. And since knowledge graphs suffer from missing types we will first run our approach of type prediction.
This chapter is organized as follow: we start with the reserch motivation and our proposed solution in section 3.2.

## 3.1 Research Motivation

As discussed in the previous chapter, knowledge graphs suffer from incompletness and noise. For noisy data H.Paulheim and C.Bizer introduced SDValidate algorithm to detect erroneous statements and in this master thesis we will present an approach to improve the results obtained by this algorithm. But first of all we will introduce the different types of errors in a knowledge base. Some errors could be detected based on the schema definitions like domain and range violations but this cannot be happen in the absence of schema definitions as defined in the previous chapter in the ontology enrichment. The errors in knowledge graphs could be in type of entities, in the predicate or wrong instances of correct types.

- Erroneous statements could be generated because of wrong typed entities. However the statement is semantically correct but the subject or the object has an erroneous type. For example we have the following triple

$< OlivierLodhe, award, FaradayMedal >$, "Olivier Lodge" has to be of type person and "Faraday Medal" has to be of type Award or a subclass of Award. In Dbpedia "Faraday Medal" has the type person which leads to consider this triple totally wrong. In fact a person cannot have an award of type *"Person"*

- Predicate errors indicate that the property asserted between two entities is wrong. when the property between two entities is wrong. For example, in DBpedia the triple $< I'maLooser, recordedIn, AbbeyRoad >$ is wrong. "I'm a looser" is a song by The Beatles from the album "Abbey Road" the relation recodedIn has domain MusicalWork and range PopulatedPlace. But according to this triple the song was recorded in an album which makes the statement erroneous. To correct it we have to replace the object or the predicate. If the object were "Abbey Road Studio" with the type "Recording studio" it will be also wrong because the latter is not a subclass of Populated Place. Therefore, we have to change the predicate to correct it.

- Erroneous statements might also contain wrong instances of correct types. For example the triple $< Ronaldo, PlayedFor, ManchesterUnited >$ could be wrong because Ronaldo refers to Ronaldo Nuzario instead of Cristiani Ronaldo. However this kind of error is not easy to detect automatically.

Inspired by a work presented in (), we conducted a similar experimentation on the DBprdia dataset to estimate Wrong Domain Rate (WDR) and Wrong Range Rate (WRR). However, Wrong Domain Rate (WDR) indicates the ratio between the number of times a property $p$ is used with a wrong domain to its total number of uses. Same for Wrong Range Rate (WRR) which is the ratio between the number of times the property is used with a wrong range to its total number of uses. The first row of Table 1 shows the number of time of each property used in the previous examples appears with wrong domain or range accompanied with its WDR or WRR. According to () the other properties of the table are properties frequenty used and appears with high value of WDR or WRR.

Table 3.1: Percentage of WDR of some of most relevant properties

| Property | Number of triples | Domain | WDR |
|---|---|---|---|
| dbo:university | 36557 | Person | 0.81 |
| dbo:country | 3963 | Person | 0.66 |
| dbo:training | 46003 | Artist | 0.66 |
| dbo:child | 73826 | Person | 0.60 |
| dbo:birthPlace | 399131 | Person | 0.36 |

Table 3.2: Percentage of WDD of some of most relevant properties

| Property | Number of triples | Range | WRR |
|---|---|---|---|
| dbo:starring | 111974 | Actor | 0.96 |
| dbo:birthPlace | 399131 | Place | 0.78 |
| dbo:country | 233700 | Counrty | 0.59 |
| dbo:award | 135534 | Award | 0.52 |

We notice that the values obtained are too high because of the incompletness of the dataset. In fact, for an RDF triple (s,p,o), if s has not the RDF:type domain of p then (s,p,o) is considered wrong. Also, if o has not RDF:type range of p then (s,p,o) is considred wrong. Indeed, the missing types lead to incorrect detection since KG often comprise of noisy data as well as incomplete data.
This motivate us to propose a type prediction approach to enhance the error detection task.

We presented in chapter 2, the best state-of-art system for error detection in Knowledge graph, SDValidate which is an approach that uses statistical distributions of properties to detect faulty statements for each predicate *p*. However, there are the same issues with this approach because it does not detect all wrong statements when we lack information about entity types. Let take the following sample of statements of the predicate dbp:award:

| Subject | Predicate | Object |
|---|---|---|
| dbr:Owen Chamberlain | dbo:award | dbr:Nobel Prize in Physics |
| dbr:Rei Dan | dbo:award | dbr:Blue Ribbon Awards |
| dbr:Sofia Coppola | dbo:award | dbr:Academy Awards |
| dbr:Lee Je hoon | dbo:award | dbr:2011 |
| dbr:Eichi Negishi | dbo:award | dbr:Nobel Prize in Chemistry |
| dbo:Im Ji kyu | dbo:award | dbr:2008 |
| dbr:Shinya Yamanaka | dbo:award | dbr:Nobel Prize in Physiology or Medicine |
| dbr:Lee Sang hee | dbo:award | dbr:United States |
| dbr:Jean Antoine Verdier | dbo:award | dbr:South Korea |

Table 3.3: Example of statements with the property dbo:award

The property dbo:award has the class ”Person” as domain and the class ”Award” as range. For example the first fact in the table is correct because the subject is a person and the object is an award. But the statements having the objects dbr:2011, dbr:2008, dbr:United States and dbr: South Korea are incorrect because these objects are not of type award. Hence the sample contains four wrong statements, but after running SDValidate only two of them were detected which are < Lee Je hoon , award, 2011 > and < Lee Sang hee, award, United States >. We used a manual evaluation on the DBpedia dataset to compute the number of erroenous statements that not detected by SDValidate. For the properties award, nationality and occupation we found respectively 32%, 28% and 21% of erreneous statements. For example in the case of the given sample below it's clear that there is a similarity between the statement < Lee Je hoon , award, 2011 > and the statement < Im Ji kyu , award, 2008 > because each one of them has an object that represent a year, and the same thing between the two other wrong statements because each one of them has an object that represent a country. However our approach aims to detect the statements that are not detected by SDValidate but similar to those which were detected. To reach this goal we apply the k-means clustering algorithm and compute the similarity between the statements based on theirs objects. We use two kind of features: The first one is the types of each object of a given predicate and the second one is the path between each object and its subject for that predicate. We apply the random walks algorithm to extract paths between entities and we extract types from the dbpedia dataset. Since dbpedia datasets suffer from missing type assertion and since our approach rely on feature type, we will first try to predict missing types using a classifier before applying the k-means clustering. Finally our approach comprises two main contributions :

- Type prediction : To asses missing types for untyped entities and then use them as features in the error detection. The main idea behind this contribution is to use properties that connect two resources as feature to indicate their types and run them into a KNN classifier.

- Error Detection : To improve the result of SDValidate and detect other errenous statements. As indicated previously, the main idea behind this contribution is to detect new erroneous statements based on theirs objects features. However, for a given relation, we use types of objects and paths that connect these objects to theirs subjects and run a k-means clustering to obtain two clusters: one for correct statements and the other for wrong statements.

## 3.2 New Approach for Type Prediction in Knowledge Graph

In our approach of type prediction we use the KNN classifier to predict missing type for a given untyped entity and the typed entities as neighbors. We used a SPARQL query to get the number of typed resource of the smallest version of dbpedia and we found that there are about 60.000 typed resources. Since the number is too large we are going to take only a sample from the set of typed entities. Also we noticed that there are at least 200 types of entities in the dbpedia dataset using SPARQL query. And since every resource often has many types we cannot detect the right type for a given untyped object or subject by directly using the KNN classifier. To solve this problem we run a hierarchical clustering to make clusters of similar types, then the result of the KNN will be a cluster of types that matches to the given untyped resource.
Our approach is divided into three steps. First we use only a sample of the typed entities. However the smallest version of dbpedia Then since every entity has many types we will make cluster of similar types using the hierarchical clustering. Finally we run the KNN classifier to on the sample of typed entities to detect the cluster of types that matches to a given untyped entity. We will discuss the details of these steps in the next subsections.

### 3.2.1 Sampling Method

In order to optimize the time of computation to create cluster, we're going to reduce the space using a sample of the large dataset. We have to choose the

adequate sampling method to garantee a good results, however if anything goes wrong with the sample then it will be directly reflected in the final result. Many sampling methods have been proposed and they are divided into two categories: Probability Sampling and Non-Probability Sampling (Tansey, 2007).

- Probability Sampling : Also known as random sampling. These method use random selection technique to make sure that every entity within the population has an equal chance to be seleted in the sample (Waksberg, 1978).

- Non-Probability Sampling : These methods rely on the reseracher's ability to select the entities for the sample and the probability of all elements to be selected is not the same (Tansey, 2007).

Since non-probability sampling methods need strategy and plan to be applied it will be too complex to use them with large datasets. In this work we're going to focus on probability sampling methods.
To create clusters of types we need to use all types of the dataset and to predict missing types we need to use all ingoing and outgoing properties and paths. We apply Simple Random Sampling based on ingoing properties, outgoing properties, paths and types. The Simple Random Sampling is the basic sampling technique where we select a group of entities (a sample) for study from a larger group (a population). Each individual is chosen entirely by chance and each member of the population has an equal chance of being included in the sample. Every possible sample of a given size has the same chance of selection. Let T be the set of types where $T = \{t_1, t_2, ..., t_{|T|}\}$. $P$ and $P^{-1}$ are respectively the set of ingoing properties and outgoing properties where $P = \{p_1, p_2, ..., p_{|P|}\}$ and $P^{-1} = \{p_1^{-1}, p_2^{-1}, ..., p_{|P^{-1}|}^{-1}\}$. The three sets present the set of features F where $\{T, P, P^{-1}\} \in F$ and $F = \{f_1, f_2, ..., f_{|F|}\}$. For each feature $f_i$ where $i \in \{1..|F|\}$ we take the population of entities that has the selected feature and then we select a sample $s_i$ from the population. Finally we create a global sample S containing all different entities selected in the previous samples.

### 3.2.2 Generate clusters of types

After creating the sample of typed entities we are going to use the hierarchical clustering algorithm to assign a cluster to each group of entities based on the similarity of their types. As discussed in section 3.2 we cannot run the KNN classifier on a on a multiclass entities and have a good results.
There are two types of hierarchical clustering: Agglomerative and divisive. The

former consider each entity as an individual cluster, then at each iteration the clusters having minimum distance merge with other clusters until the formation of one or K clusters. The latter consider all entities as one cluster, then at each iteration the entities having maximum distance between them will be divided into K clusters.
In our approach we use the agglomerative clustering which starts with each element as a unique cluster and then merge them successively into larger clusters to get a final cluster. At each particular stage, the two most similar clusters are merged together.
The hierarchical ascendant algorithm (HAC) is like follows:

**Algorithm 3.1** HAC algorithm

```
Begin
Initialization:
N_E: set of entities
for i from 1 to N_E do
   Create a cluster
   C = {c_1, ..., c_N}
end for
for j from 1 to c_N do
   for i from 1 to c_N do
      distance (c_j, c_i)
      create the similarity matrix
   end for
end for
while |C| > 1 do
   recover the pair (c_n1, c_n2) from the clusters having the maximum similarity
   remove from C these two clusters
   add to C the cluster corresponding to their melting
   update the similarity matrix
end while
End
```

We calculate the distance using the euclidean distance which has the following formula :

$$d(e_i, e_j) = \sqrt{\sum_{i=1}^{n} (e_i - e_j)^2}$$

Then we use the types of the entities of the generated sample as features. Let ES be the set of entities of the sample and T the set of types where:

- $ES_i$ denotes the entity $i$ of the sample and $i \in [1..|ES|]$

- $T_j$ denotes the type $j$ and $j\ in [1..|T|]$

- $ES_i(T_j) \in \{0, 1\}$ to indicate if an entity $ES_i$ has the type $T_j$

The first step is to compute the proximity matrix, and since the number of entities and types is large we have to apply a heuristic. Indeed, the smallest version of DBpedia dataset for types contains at least 500 000 statements having at least about 200 different labels representing types and each entity can have at most about 10% of the existing types. The computation of euclidean distance will take too much time because the vector of features is large. For this reason we have created submodels instead of working on the general model and the entities of each submodel have at least one type in common. Formally for two entities $ES_i$ and $ES_{i'}$ we have $ES_i \bigcap ES_{i'} \neq \oslash$. Let take an example of a small dataset :

| Entity | Type |
|---|---|
| dbr:Albert Einshtein | dbo:Person |
| dbr:Albert Einshtein | dbo:Scientist |
| dbr:Charles Darwin | dbo:Person |
| dbr:Charles Darwin | dbo:Scientist |
| dbr:Barak Obama | dbo:Person |
| dbo:Barak Obama | dbo:President |
| dbr:Beji Kaid Essebsi | dbo: Person |
| dbr:Beji Kaid Essebsi | dbo: President |
| dbr:United states | dbo:Place |
| dbr:United states | dbo:Country |
| dbr:South Korea | dbo:Place |
| dbr:South Korea | dbo:Country |
| dbr:Tunisia | dbo:Place |
| dbr:Tunisia | dbo:Country |
| dbr:Tunis | dbo:Place |
| dbr:Tunis | dbo:Capital |
| dbr:Paris | dbo:Place |
| dbr:Paris | dbo:Capital |
| dbr:Ons Jabeur | dbo:Person |
| dbr:Ons Jabeur | dbo:Athlete |
| dbr:Rafael Nadal | dbo:Person |
| dbr:Rafael Nadal | dbo:Athlete |

Table 3.4: Sample of typed entities

When we apply the heuristic on this sample of data we will obtain two submodels. The first submodel $M1$ contains the entities that have the common type dbo:Person and the second one $M2$ contains the entities that have the common type dbo:Place. Then we apply the hierarchical clustering on each submodel and we obtain a set of cluster C where $C = C_1, C_2, ..., C_{|C|}$:

- C1 : Presents the cluster of scientist person and contain the types dbo:Person, dbo:PopulatedPerson and dbo:Scientist.
- C2 : Presents the cluster of the person who are presidents of countries and contains the types dbo:Person and dbo:President.
- C3 : Presents the cluster of athlete persons and contains the types dbo:Person and dbo:Athlete.

- C4 : Presents the places which are contries and contains the types dbo:Place and dbo:Country.
- C5 : Presents the place which are capitals and contains the types dbo:Place and dbo:Capital.


Figure 3.1: Hierarchical clustering of the first submodel

### 3.2.3 Adding missing types

After creating clusters of types we will apply the k-Nearest-Neighbours classifier to add missing types. However the kNN is a non-parametric classification method, which is simple but effective in many cases. For a data record an entity $E_k$ to be classified where $k \in 1..|E|$, its k nearest neighbours are retrieved, and this forms a neighbourhood of $E_k$. Majority voting among the data records in the neighbourhood is usually used to decide the classification for $E_k$. However, to apply kNN we need to choose an appropriate value for k, and the success of classification is very much dependent on this value. In a sense, the kNN method is biased by k. There are many ways of choosing the k value, but a simple one is to run the algorithm

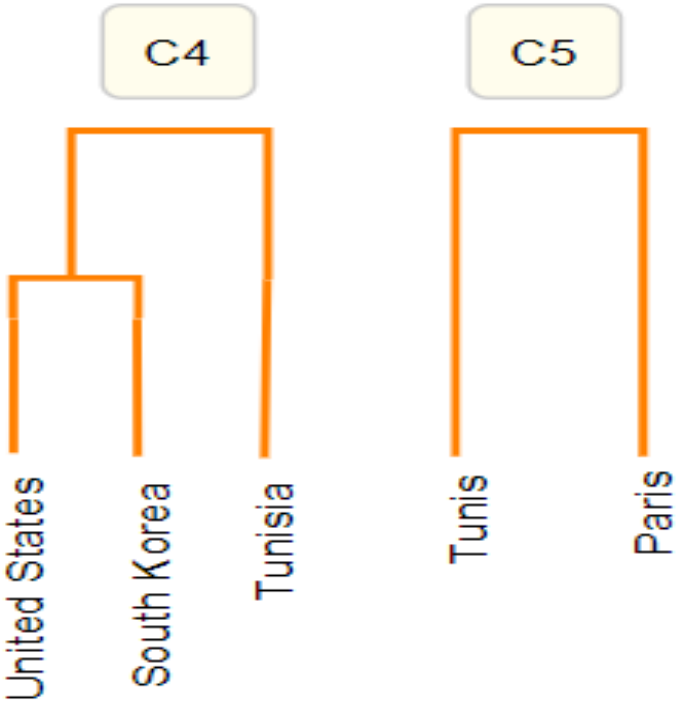


Figure 3.2: Hierarchical clustering of the second submodel

many times with different k values and choose the one with the best performance. For each untyped entity we retrieve the number of each label of ingoing and outgoing properties that appear whith that resource and we choose $G$ properties that have hight frequencies to be features of that resource. For example the resource "dbr:United States" appears with 20 ingoing and outgoing properties. Some properties like $dbo : location^{-1}$ and $dbo : birthPlace^{-1}$ have respectively the frequence 520 and 350 and some others like $dbo : award^{-1}$ and $dbo : locaion^{-1}$ have respectively the frequency 4 and 2. We choose the top G predicates because those who have low frequency have a hight probability to be wrong. However, a statement that has the predicate $dbo : award^{-1}$ with the object "United states" cannot be correct since $dbo : award^{-1}$ must appears with objects that have the type "Award". Indeed, predicting types on noisy data can lead to false type prediction.
After choosing the top properties that appear with the untyped given entity, we use them as features of it. Then for each untyped resource we create a model with typed entities that repect two constraints:

- Each typed entity of the model must have at least one predicate in common with the given resource. More formaly for each untyped entity $E_k$ we create a

model and for each entity of the model $ES_i$ we have the following constraint $E_k \bigcap ES_i \neq \oslash$.

- Each typed entity of the model must have the same nature as the given untyped resource. However if the given entity is an object (has only ingoing properties) the entities of the model must also be only objects and so onif it was subject or both subject and object.

Finally we run the KNN classifier to choose the cluster of types that matches to the untyped resource.

## 3.3 Error Detection

Our seceond contribution is to improve the results of SDValidate and using the k-means algorithm to detect if there is a triple that does not appear in the results of SDValidate but similar to triples resulted by this algorithm. We use two kinds of features: The types of subjects and objects in a one given relation like the approach of SDValidate and the paths features like in the approach of PRA.
The problem with the first approach that solely relying on type feautures do not give performant results when we work with knowledge graphs that suffer from missing type assertion. The second approach has also problems. If we use only path features we can classify correct facts as wrong because of the incompletness in the case of missing properties. However the best way to create our model is to combinate the two kind of features.

### 3.3.1 Features Extraction

We have already indicate that we use two kinds of feature in our model. Before extracting the feature we run SDValidate algorithm and we select the properties that appear in the wrong triples detected by SDValidate. Then we work on each relation solely to get the features of its subjects and objects. We extract the types of entities from the dbpedia dataset. However, we run the random walks algorithms over the graph to find paths between two entities that are connected by a relation between them.
A random walk on a graph is the process of visiting the nodes of the graph in some sequential random order. Since we compute the similarity of statements based on theirs objects, the walk starts at some fixed node which presents an object, and at each step it moves to a neighbor of the current node chosen randomly. We extract the path when the final is a connected subject for that object

with a given predicate. We select the maximum path length and we do not allow a relation to be immediately followed by its inverse. Let $P_{max}$ be the maximum path length and we define a path as a sequence of relations denoted by $r_1 \rightarrow r_2 \rightarrow ... \rightarrow r_n$ and which is connected by entities. A path between a subject s and an object o is denoted by $P(s, o) \Leftrightarrow r(s, e_1) \rightarrow r(e_2, e_3) \rightarrow ... \rightarrow r(e_{n-1}, o)$. The inverse of a relation is denote by $r^{-1}$ where $r(s, o) = r^{-1}(o, s)$. For example we want to extract a paths for between the object and the subject of the triples $< BarackObama, bornIn, Honolulu >$. After running the random walks we find two paths: The first one is the predicate $bornIn^{-1}$ and the second one is $capital/study^{-1}$.

### 3.3.2 Applying the K-means

Once the types and paths have been selected we use them as features to create a model for each property that appears in the output of SDValidate. For a given predicate, we initialize two clusters, one for have the label "Wrong" and contains the erroneous statements resulted by SDValidate and the other have the label "Correct" and presents the rest of statements. We compute the centroids $C_{correct}$ and $C_{wrong}$ of each cluster based of the features selected. Since that error detection method of SDValidate focus on wrong objects for a property we compute the distance between each object appearing with correct statement with the centroid $C_{correct}$ and $C_{wrong}$ to check if that object is really "correct" or not. Let take the following example of statements of property "award".

Table 3.5: Example of statements of property award after affecting the clusters

| Subject | Predicate | Object | Cluster |
|---|---|---|---|
| dbr:Albert Einshtein | dbo:award | dbr:Nobel Prize in Physics | correct |
| dbr:Douglas MacArthur | dbo:award | Medal of Honor | correct |
| dbr:Lee Beom seok | dbo:award | dbr:1963 | correct |
| dbr:Alexander Fleming | dbo:award | dbr:Nobel Prize in Physiology | correct |
| dbr:Paik Sun yup | dbo:award | dbr:2001 | false |
| dbr:Chien Shiung | dbo:award | 1966 | false |

We extract the type and paths feature of each objects in the example and create a binary model:

Table 3.6: Example of model of property award

| Object | dbo:award | dbo:year | award | birthdate/deathdate$^{-1}$ | Cluster |
|---|---|---|---|---|---|
| dbr:Nobel Prize in Physics | 1 | 0 | 1 | 0 | correct |
| Medal of Honor | 1 | 0 | 1 | 0 | correct |
| dbr:1963 | 0 | 1 | 0 | 1 | correct |
| dbr:Nobel Prize in Physiology | 1 | 0 | 1 | 0 | correct |
| dbr:2001 | 0 | 1 | 0 | 1 | false |
| dbr:1966 | 0 | 1 | 0 | 1 | false |

A statement with the property ”award” must have an object of type award and the object dbr:1963 doesn't have this type and wasn't detected as erroneous. If we apply the k-means and we compute the distance between it and each one of the two centroid we find that its close to the cluster ”false”.

We do this step for each objects of the cluster ”correct” and we update each time the clusters centroids.

## 3.4 Conclusion

In this chapter we presented our approach to improve the quality of knowledge base to add missing types and to get better results when we try to improve the output of SDValidate. We used types and paths features and created a binary model to run k-means algorithm and detect wrong statements.
Different steps of the approach are experimentally studied to evaluate the performance of the proposed approach in the next chapter.

Chapter 4

# Experiments

## Introduction

To evaluate the performance of our approach of type prediction as well as error detection in RDF knowledge graphs, we describe in this chapter different experiments with different DBpedia datasets. Then we make a manual evaluations for the results. The rest of this Chapter is organized as follows: Section 4.2. presents the envirement and the datasets used in this work, section 4.3 contains the experiments results for type prediction and section 4.4 contains the experiments results for error detection.

## 4.1 Environment and datasets

Two different test environment were used to deal with this work.
The first is a machine with Intel i7-6700HQ with 8GB of memory using the operating system Windows 10 and eclipse IDE for java developers with the database SQL. This environment where used to add missing types with KNN classifier and to detect erroneous statements with K-Means.
The second is a machine with 64GB of memory using the operating system Windows 10 and eclipse IDE for java developers with relational database H2 to run the Random Walk algorithm to get the paths features.
To evaluate our approach we use three versions of DBpedia: the French version, the German version and the Korean version.

| Dataset | Triples | Typed Entities | Untyped Entities |
|---|---|---|---|
| French | 3 745 190 | 444 202 | 415 390 |
| Koorean | 461 795 | 58 937 | 6944 |

## 4.2 Evaluation Measures

To evaluate the performance of our proposed approach we need to evaluate the quality of the results. The evaluation is done through four metrics bwhich are Accuracy, recall , precision and F-measure based on the confusion matrix.

### 4.2.1 Confusion Matrix

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. It gives an insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

| | Class 1 Predicted | Class 2 Predicted |
|---|---|---|
| Class 1 | TP | FN |
| Class 2 | FP | TN |

According the confusion matrix below Class 1 presents the positive results and Class 2 presents the negative results and the other terms are :

- Positive (P) : Observation is positive (for example: is an apple).
- Negative (N) : Observation is not positive (for example: is not an apple).
- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

The values of these terms will be used to compute the metrics.

### 4.2.2 The Evaluation Metrics

- **Accuracy :** It's the ratio of the correctly labeled subjects to the whole pool of subjects. Accuracy is the most intuitive one. Accuracy answers the following question: How many instances did we correctly label out of all the instances?

$$Accuracy = \frac{(TP+TN)}{(TP+FP+FN+TN)}$$

- **Precision :** It's the ratio of the correctly instance labeled by the model to all instance labeled. Precision answers the following: How many of those who we labeled as erroneous are actually erroneous?

$$Precision = \frac{TP}{TP+FP}$$

- **Recall :** It's the ratio of the correctly instance labeled by our model to all who are erroneous in reality. Recall answers the following question: Of all the instances who are erroneous, how many of those we correctly predict?

$$Recall = \frac{TP}{TP+FN}$$

- **F-Measure :** It considers both precision and recall. It is the harmonic mean(average) of the precision and recall. F1 Score is best if there is some sort of balance between precision (p) recall (r) in the system. Oppositely F1 Score isn't so high if one measure is improved at the expense of the other. For example, if P is 1 R is 0, F1 score is 0.

$$F - measure = 2 * \frac{(Recall*Precision)}{(Recall+Precision)}$$

## 4.3 Type Prediction Evaluation

We apply the sampling method described in section 3.2.1 of chapter 3 to choose a sample of typed entities and to accelerate the computation of the KNN. After creating the sample we run the hierarchical clustering to generate cluster of types as defined in section 3.2.2. Finally we run the KNN algorithm to choose a cluster of types for each untyped entity respecting the several constraints. We explain how we apply these constraints and how we choose the $K$ nearest neighbors in the next subsections.

### 4.3.1 Applying the constraints

Predicting types on noisy data can lead to false type prediction, if we directly apply the KNN without any restrictions we cannot enable the generation of erroneous type prediction and damage the quality of the data. Therefore, we apply three constraints to reduce the rate of false type prediction.

#### First constraint

For every untyped entity we choose a set of typed entity from the generated sample that have at least one feature in common.

#### Second constraint

If the untyped entity presents an object (respectively subject or both object and subject), then every entity of the chosen set must also present an object (respectively subject or both object and subject).

#### Third constraint

For each untyped entity we only select the $p_f$ most frequent predicates as features. However, the more the frequence of that predicate is frequent the more it has a high estimation to be correct.

#### Example

We take the example of the untyped entity dbr:UnitedStates and the set of typed entities $dbr : France$, dbr:AlbertEinshtein , dbr:NoberlPrizeinPhysics, dbr:Tunisia and dbr:Stadium.

- dbr:France(Place, country, populatedPlace)
- dbr:AlbertEinshtein (Person, Scientist)
- dbr:NoberlPrizeinPhysics(Award)
- dbr:Tunisia (Place, country, populatedPlace)
- dbr:Stadium (Place, Settelment)

Table 4.1 presents the predicates of each entity of the example. When we apply the first constraint the set of entity of the model will be : France, Albert Einshtein

, Tunisia, Nobel Prize in Physics. But Albert Einshtein and nobel prize in physics have types which are completely different from United States. So applying only the first constraint is not the best solution.

Table 4.1: Example of model

| United States | France | Albert Einshtein | Nobel Prize | Tunisia | Stadium |
|---|---|---|---|---|---|
| $location^{-1}$ | $city^{-1}$ | $occupation$ | $award^{-1}$ | $country^{-1}$ | $Location - 1$ |
| $birthPlace^{-1}$ | $birthPlace^{-1}$ | $award$ | | $playIn^{-1}$ | |
| $city^{-1}$ | $liveIn^{-1}$ | $bithPlace^{-1}$ | | | |
| $country^{-1}$ | $city^{-1}$ | | | | |
| $award^{-1}$ | | | | | |
| $occupation^{-1}$ | | | | | |

The entity United States have only ingoing predicates, so it presents an object. We apply the second constraint and we choose the entities Tunisia, France and Nobel Prize in Physics which are all objects. Finally we apply the third constraint to choose the most frequent predicates as features. According to table 4.2 the most frequent predicates are: $location^{-1}$, $birthPlace^{-1}$ and $city^{-1}$. We do not choose those who has low frequency because they have a hoght probability to present error triples like $award^{-1}$ and $occupation^{-1}$.

Table 4.2: Features of the entity United States and theirs frequences

| United States | |
|---|---|
| Predicate | Frequence |
| $location^{-1}$ | 520 |
| $birthPlace^{-1}$ | 312 |
| $city^{-1}$ | 240 |
| $country^{-1}$ | 2 |
| $award^{-1}$ | 4 |
| $occupation^{-1}$ | 2 |

After applying both the three constraints the final model for the entity United States will be composed of the entities France, Tunisia and Stadium which are all of types dbo:Place.

### 4.3.2 Choosing the $K$ nearest neighbors

After creating the model we run the KNN classifier and choose the K entities that have minimum distance and affect the types of the dominnat cluster to that entity. Choosing the $K$ nearest neighbors is a challenging task. In our case each entity has its own model. For instance, the dbr:UnitedStates has a model of 145 entities while dbr:NobelPrizeOfPeace has a model of 20 entities. We cannot fix a standard $K$ value, but we need to fix a percentage $k$. We choose 3 value of $k$ and evaluate the results on the koorean and french dataset and we detail the precision as shown in the following tables.

Table 4.3: Evaluation results for k=0.2

| Dataset | Precision |
|---|---|
| Koorean | 0.8913 |
| French | 0 |

Table 4.4: Evaluation results for k=0.3

| Dataset | Precision |
|---|---|
| Koorean | 0.9637 |
| French | 0 |

Table 4.5: Evaluation results for k=0.4

| Dataset | Precision |
|---|---|
| Koorean | 0.9581 |
| French | 0 |

According to Table 4.3, Table 4.4 and Table 4.5 we get a higher precision=0.9637 in the koorean dataset with k=0.3. So we choose 0.3 as a value for k.

## 4.4 Error Detection Evaluation

To test our approach of error detection we run first SDValidate on the given dataset. Then for each predicate we generate a cluster of error objects of the statements containing the objects of the output of SDValidate and a cluster for correct objects of the statements containing the rest of triples of the given predicate. Based on the two generated clusters, we run the k-means algorithm and we compare the similarity of each object regarding the two centroids of clusters. If the distance between the object and the cluster of error objects is less than the distance between that object and the cluster of correct object, we affect that object to the cluster of errors. Finally, after running the k-means several times and based

on the objects of the clusters, we generate the erroneous statements.
We use the Koorean dataset of DBPedia and we run at fist our approach of error detection on the predicates of the koorean dataset, then we add the results of our approach of type prediction to use the predicted types with the other features and we run again our approach of error detection.
For the comparision results we select some predicates with different results to discuss them. In the next subsection we define each predicate semantically to understand the errors.

### 4.4.1 Description of the chosen predicates

To evaluate the result of our approach, we choose the predicates dbo:award, dbo:prodecessor, dbo:owner and dbo:education. We define each one of them and give theirs numbers of triples.

- The predicate **dbo:award** is used in a statement to indicate that a person, a movie or a song has earned a specific prize. For example the triple $< AlbertEinstein, award, NobelPrizeinPhysics >$ indicates that Albert Einstein earned a nobel prize in physics. In DBPedia, it has a range of type dbo:Award.

- The predicate **dbo:predecessor** indicate that a person precede a person in a given post. For example the statement $< BarackObama, predecessor, GeorgeBush >$ indicates that George Bush was the president of the United States before Barack Obama.

- The predicate **dbo:owner** in a statement indicates that a person own a society, industry, a club or any well known brand. For example we can indicate that Bill Gates is the owner of Microsoft by the following triple$< BillGates, owner, Microsoft >$.

- The predicate **dbo:education** indicates that a person has continued her studies in a given university or school. Example: $< EmanuelPastreich, education, HarvardUniversity >$.

- The predicate **dbo:publisher** indicates that a person, a society or a laboratory is a publisher of some work. For example we can use this predicate to indicate that Joanne Kathleen Rowling is the publisher of Harry Potter series.

- The predicate **dbo:location** has always an object of type place and used to indicate the location of a given subject.
  Example: $< OstafyevoInternationalAirport, location, Russia >$.

- The predicate **dbo:region** indicates he region where the thing is located or is connected to. Example: $< MondegoRiver, region, Portugal >$.

We make a manual evaluation on the statements of each predicate to compute the real number of erroneous statements. The Table 4.1 presents the total number of triples and the number of erroneous triples of each property of the Koorean Dataset. Then we apply the SDValidate for error detection and our approach to compare them.

Table 4.6: Results of the manual evaluation on each predicate

| Predicate | Number of Triples | Erroneous Triples |
|---|---|---|
| dbo:award | 793 | 136 |
| dbo:predecessor | 883 | 28 |
| dbo:owner | 540 | 50 |
| dbo:education | 732 | 145 |
| dbo:publisher | 394 | 32 |
| dbo:location | 4210 | 0 |
| dbo:region | 2435 | 12 |

### 4.4.2 Evaluation of the experiments

After the manual evaluation, we use the predicates described above on three experiments which are:

- SDValidate for error detection.

- SDValidate with the result of the KNN approach for type prediction

- SDValidate and our two approach: k-means for error detection and the result of KNN for type completion to add missing types features.

Table 4.2 shows the results of the expériments. We observe three types of results devided as follow:

- **Result 1:** We note that the third approach outperforms the two previous one with the predicates dbo:award, dbo:predecessor and dbo:education. For example for the predicate dbo:predecessor we obtained only 4 erroneous statements out of 28 when we runned SDValidate. However, we can detect the half of erroneous statements when we combining SDValidate with k-means. Then this result was increased by detecting 70% of the faulty predicates when we add the result of KNN to the set of type features.

- **Result 2:** We note that only the second approach outperforms the first one for the predicates dbo:owner and dbo:publisher. Indeed, there is no difference between the results of the second approach and the third one.

- **Result 3:** We note that SDValidate has bad results for the predicates dbo:location. In fact, 418 erroneous statements were detected for the predicate dbo:location while we didn't find any faulty statements using the manual evaluation. When we use the results of KNN with sdvalidate the number of error statements decreases from 418 to 212 but when we apply the k-means it increases from 212 to 2310 which is greater than the half of total triples having the predicate dbo:location. Since DBpedia's correctness is estimated to be 92% and SDValidate detects erroneous triples based on the outliers, the number of erroneous statements per predicate could not be high. If we run our approach and we find that the total number of erroneous statements is greater than the rest of the statements that means that SDValidate had probably bad results for the given predicate and we should not take them.

- **Result 4:** We note that SDValidate has bad results for the predicate dbo:region. In fact, 250 erroneous statements were detected while we found only 12 erroneous statements the manual evaluation. Adding the predicting types by the KNN to the other features had good impact on the results of SDValidate. However, the number of faulty statements decreases from 250 to 7 which is less than then total number of erroneous statements and even after we apply the K-means the number increases from 7 to 10 but still less that the total number of erroneous statements.

For each type of results, we present the comparison results table using the evaluation metrics described in section 4.2.1.

Table 4.7: Experiments on the chosen predicates

| Predicate | SDValidate | SDValidate+KNN | SDValidate+KNN+K-means | Erroneous Triples |
|---|---|---|---|---|
| dbo:award | 63 | 90 | 126 | 136 |
| dbo:predecessor | 4 | 8 | 23 | 28 |
| dbo:education | 75 | 86 | 135 | 145 |
| dbo:owner | 32 | 44 | 44 | 50 |
| dbo:publisher | 9 | 29 | 29 | 32 |
| dbo:location | 418 | 212 | 2310 | 0 |
| dbo:region | 250 | 7 | 10 | 12 |

**Evaluation metrics for Result 1**

In the following tables we present the values of the evaluation measures of the predicates dbo:award, dbo:prodecessor and dbo:education.

Table 4.8: Comparison results on the predicate dbo:award

| Algorithm | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| SDValidate | 0.9079 | 0.9000 | 1.000 | 0.9478 |
| SDValidate + KNN | 0.9419 | 0.9345 | 1.000 | 0.9661 |
| SDValidate + K-means + KNN | 0.9873 | 0.9850 | 1.000 | 0.9924 |

Table 4.9: Comparison results on the predicate dbo:prodecessor

| Algorithm | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| SDValidate | 0.9012 | 0.8994 | 1.000 | 0.9470 |
| SDValidate + KNN | 0.9214 | 0.9194 | 1.000 | 0.9580 |
| SDValidate + K-means + KNN | 0.9824 | 0.9799 | 1.000 | 0.9898 |

Seeing these results we can affirm that using KNN for type prediction and k-means for error detection with the SDValidate approach has a good impact on the results. In fact, we notice that the values of the accurancy, the recall and the F-measure have increased throught the three approaches. That means that both types and paths features helped to detect new error statements.

Table 4.10: Comparison results on the predicate dbo:education

| Algorithm | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| SDValidate | 0.9043 | 0.8934 | 1.000 | 0.9832 |
| SDValidate + KNN | 0.9863 | 0.9832 | 1.000 | 0.9915 |
| SDValidate + K-means + KNN | 0.9863 | 0.9947 | 1.000 | 0.9915 |

**Evaluation metrics for Result 2**

In the following tables we present the values of the evaluation measures of the predicates dbo:award, dbo:owner and dbo:publisher.

Table 4.11: Comparison results on the predicate dbo:owner

| Algorithm | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| SDValidate | 0.9749 | 0.9722 | 1 | 0.9859 |
| SDValidate + KNN | 0.9981 | 0.9979 | 1 | 0.9989 |
| SDValidate + K-means + KNN | 0.9981 | 0.9979 | 1 | 0.9989 |

Table 4.12: Comparison results on the predicate dbo:publisher

| Algorithm | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| SDValidate | 0.9368 | 0.9402 | 0.9972 | 0.9666 |
| SDValidate + KNN | 0.9898 | 0.9945 | 0.9972 | 0.9958 |
| SDValidate + K-means + KNN | 0.9898 | 0.9945 | 0.9972 | 0.9958 |

Seeing these results we can affirm that solely using KNN for type prediction had improved the results. In fact, we notice that the values of the accurancy, the recall and the F-measure have increased only throught the second approach. That means that solely adding types features helped to detect new error statements.

**Evaluation metrics for Result 3**

In the following tables we present the values of the evaluation measures of the predicates dbo:location. We only compare the two first approaches since SDValidate detected correct statements as wrongs and applying k-means has increased

this wrong detection. As explained above, we will not take the results of k-means because the number of error triples generated is greater than the rest of them.

Table 4.13: Comparison results on the predicate dbo:location

| Algorithm | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| SDValidate | 0.8962 | 1.000 | 0.8769 | 0.9344 |
| SDValidate + KNN | 0.9496 | 1.000 | 0.9496 | 0.9741 |

Seeing these results we can affirm missing types of entities lead to false error detection. In fact, when we add the types features predicted by KNN we notice that the values of the accurancy, the recall and the F-measure have increased.

**Evaluation metrics for Result 4**

In the following tables we present the values of the evaluation measures of the predicates dbo:region.

Table 4.14: Comparison results on the predicate dbo:region

| Algorithm | Accuracy | Precision | Recall | F-measure |
|---|---|---|---|---|
| SDValidate | 0.8880 | 0.9945 | 0.8970 | 0.9434 |
| SDValidate + KNN | 0.9979 | 1.000 | 0.9992 | 0.9995 |
| SDValidate + K-means + KNN | 0.9991 | 1.000 | 0.9998 | 0.9998 |

Seeing these results we can affirm that applying KNN for type prediction has changed a lot the results towards the best. In fact, there are an outstanding change between the first and the second approach on the values of the accurancy, the precesion, the recall and the F-measure. Also applying the k-means has increased the results.

# Conclusion

In this chapter, we have detailed the implementation of our approach with a detailed description of the entire process. In addition, we have illustrated the different simulations on real databases.

With the use of evaluation measures, we have compared our methods with SDValidate. The results obtained shows that our method is relevant and effective.

# Conclusion