Master in Computer Science
Rheinische Friedrich-Wilhelms-Universität Bonn

Scalable Numerical Outlier Detection in Knowledge Graphs

Supervisor: Dr. Hajira Jabeen
First Examiner: Prof. Dr. Jens Lehmann
Second Examiner: Dr. Steffen Lohmann

Rajjat Dadwal
Matriculation Number: 2892280
18-07-2018

Statutory Declaration

I hereby certify that all work presented in this master thesis is my own, no other than the sources and aids referred to were used and that all parts which have been adopted either literally or in a general manner from other sources have been indicated accordingly.

Author: Rajjat Dadwal
Date: 18-07-2018

....
(Signature)

Acknowledgements

I would like to thank my supervisor Dr. Hajira Jabeen for her utmost support, understanding and confidence in me throughout the thesis period. I am immensely grateful to Prof. Dr. Jens Lehmann and Dr. Steffen Lohmann for letting me pursue master thesis at the Smart Data Analytics (SDA) group in the University of Bonn (at the Department of Computer Science). I am also obliged to Mr. Gëzim Sejdiu for sharing his knowledge about Spark and Scala and helped me to resolve technical blockers.
Furthermore, I would like to thank my parents for their endless support and love.

Abstract

Outlier detection finds unusual patterns in the data and can be applied to varied types of datasets. Outliers may exist due to the erroneous record in the data, but sometimes correct instances also deviate from the usual patterns. The outlier detection technique has a wide range of applications in different fields like security, finance, etc. Our thesis is focused on one of the areas known as Semantic Web.

The Semantic Web provide Resource Description Framework (RDF), a standard data format for exchanging the information on the web. Various organizations and academia are continually working on creating RDF datasets, which leads to the enormous growth of the size of the data. A single isolated machine is inadequate to process the massive and heterogeneous data, and it needs to use a parallel processing system like Spark, Hadoop or Flink. Also, the graph like representation and multimodal nature of RDF makes the outlier detection technique more challenging. In our Master thesis, we have implemented a generic approach to find the numerical outliers in Knowledge Graphs using the Spark framework and Scala as a programming language. We have successfully found outliers in RDF data and manually classified into natural as well as real outliers.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

The Internet has exploded with a large and wide variety of data over the last few years. The extensive data available over the web is human readable, but it is difficult for a machine to understand the semantics and syntax of the data. There is a need for defining the meaning on the web of data so that machines can easily understand the information behind it. Semantic web knowledge graphs has filled this gap by introducing Resource Description Framework (RDF)[1]. It is a W3C[2] recommendation for describing the semantics of the data present on the web.

Many industries and academia are continuously creating and publishing RDF data over the Internet. The industries like DBpedia, Wikidata, Yago, etc. are generating RDF data and maintaining their database for further researches (like improving RDF data quality). For example, " Altogether the DBpedia 2016-04 release consists of 9.5 billion (2015-10: 8.8 billion) pieces of information (RDF triples) out of which 1.3 billion (2015-10: 1.1 billion) were extracted from the English edition of Wikipedia, 5.0 billion (2015-04: 4.4 billion) were obtained from other language editions and 3.2 billion (2015-10: 3.2 billion) from DBpedia Commons and Wikidata[3] ". It is hard to create and manage such a bulk amount of data without mistakes.

For instance, Wikipedia pages are maintained manually, and the input is neither restricted nor validated [17], hence are not error-free. It is also possible that, while converting Wikipedia pages to RDF data, the extraction tool may lead to incorrect RDF triples. For example, RDF databases like DBpedia extracts information by using heuristic information methods from Wikipedia and may extract the incorrect values in the output. These incorrect values sometimes act as outliers, showing anomalous behavior when compared with the rest of the data. For example, a village population may be mistakenly recorded in millions or billions in the RDF database and considered as an outlier when it is compared with the population of the other villages. It is an example of a real outlier. There may also be natural outliers present in the data- A natural outlier is a correct value, but still it shows

[1]`https://www.w3.org/RDF/`
[2]`https://www.w3.org/`
[3]`http://wiki.dbpedia.org/dbpedia-version-2016-04`

an anomalous behavior when compared to the rest of the data. For example, the population of India and China acts as natural outliers (India and China have an exceptionally high population compared to the rest of the world).

The outliers present in the RDF dataset need to be identified to improve the quality of the data. However, we have focused our research on finding numerical outliers exclusively. Due to the sheer size of the RDF data, it is challenging for a standard machine with limited memory and processing speed to process it, which leads to the necessity of distributed computation to find outliers. Several open source distributed frameworks available over the Internet like Hadoop, Spark, etc. All these frameworks help in processing the bulk data in an efficient and scalable manner. However, the high processing speed of Spark, i.e., 100x faster than Hadoop, and its in-memory processing[4], make Spark beneficial for our research. So, we have used the Spark framework with Scala programming to find the numerical outliers in the RDF Data.

1.2 Challenges

RDF data is heterogeneous, multimodal and its size is growing rapidly. The traditional techniques to find the outliers on RDF data are unsuitable to produce the correct results. For example, height property is associated with multiple domains like person, building, etc. The outlier detection technique applied on height property will lead to incorrect results. It necessitates to build a scalable algorithm that can find the outliers in an efficient manner. To the best of our knowledge, this is the first scalable and generic approach for unsupervised outlier detection on knowledge graphs. The past researches on this domain are restricted to detecting numerical outlier only on a subset of RDF data. For example, [17] has focused on a subset of properties of RDF triples like DBpedia-owl:populationTotal, DBpedia-owl:height, and DBpedia-owl:elevation. On the other hand, our method is generic and works for all the properties of the RDF data. However, due to the volume of the data, the creation of cohorts[5] of classes made our task more challenging. We have implemented different approaches like the cartesian product, mapPartitions, and minHashLSH methods to solve the cohorting problem and will compare them in subsequent chapters. We attempt to find a solution to the problem, i.e., "detecting numerical outlier in knowledge graphs in a scalable manner" by distributing the data with the assistance of the Spark framework.

[4]`https://spark.apache.org/`

[5]`http://www.dictionary.com/browse/cohort`

1.3 Outline

This thesis is subdivided into six chapters. In chapter two, we give some background understanding of RDF, Big Data, DBpedia, etc. The past work related to our research is described in chapter three. Chapter four is associated with the various methodologies that are implemented to obtain the results. The evaluation and results of our thesis are explained in chapter six. The last chapter is dedicated to the conclusion of the entire research.

2 Technical Details

In this chapter, we establish the background of the thesis work by formally explaining the concept of Semantic Web, Linked Data, Resource Description Framework (RDF), different RDF Databases and importance of the Big data for the detection of outliers in RDF data.

2.1 Semantic Web

Semantic is a synonym of 'meaning' and therefore Semantic Web is defined as a web with meaning. The primary purpose of the Semantic Web is to understand the meaning behind the information present on the Internet. [1] has defined the Semantic Web as "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation". According to the W3C, "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries[1]". The Semantic Web is recognized as an integrator that combines different content, information applications, and systems.

The figure 2.1 describes the semantic web stack, also known as the Semantic Web Layer cake[1]. Every layer is assigned to a particular task and helps in making the web information machine-understandable. For example in the figure 2.1, RDF is the standard data format for exchanging information on the web suggested by W3C. There is another layer called an RDF Schema (RDFS). RDFS is a semantic extension of RDF. It provides mechanisms for specifying groups of related resources and the connections between these resources. These resources are used to determine characteristics of other resources, such as the domains and ranges of the properties [2]. SPARQL, a querying language helps in querying RDF database. OWL (Web Ontology Languages) provides a larger vocabulary for the things. OWL extends its vocabulary from RDFS vocabulary. We can perform Set operations, define equivalence relations between RDF databases and do many more things with the help of OWL vocabulary.

[1] `https://en.wikipedia.org/wiki/Semantic_Web`
[2]`https://www.w3.org/TR/rdf-schema/`

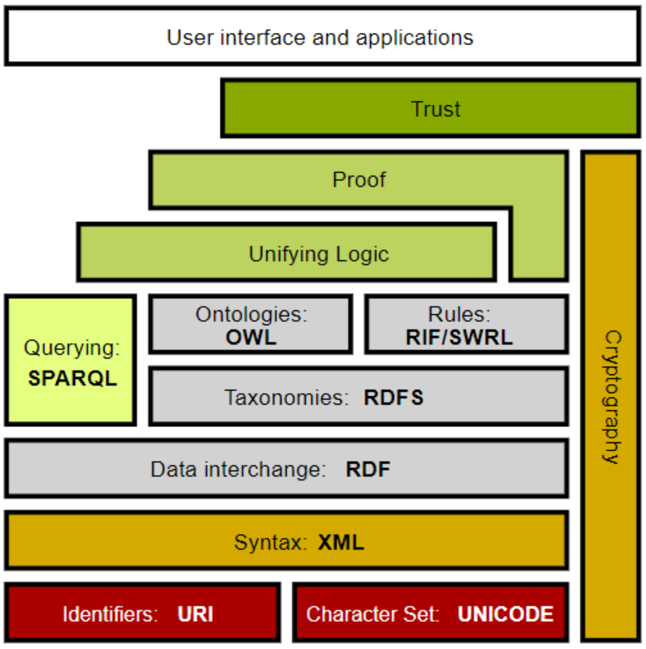


Figure 2.1: The Semantic Web Layer[3]

2.2 Linked Data

Linked Data, as the name symbolizes, means the interlinking of data on the web. It aims not only to serve the pages for human readers but also to extend web pages in such a way that machines can also understand them automatically[4]. There are a set of guidelines to publish data on the web mentioned by Tim Berners-Lee[5]:-

1. The Uniform Resource Identifier (URI) is used to identify the things.
2. HTTP URIs is used so that people can look up those names.
3. Through URIs, one can get the useful information with the assistance of RDF and SPARQL.
4. URIs are linked to other URIs in Linked data and are helpful in discovering new things.

[3]https://en.wikipedia.org/wiki/Semantic_Web
[4]https://en.wikipedia.org/wiki/Linked_data
[5]https://www.w3.org/DesignIssues/LinkedData.html

The common example of Linked data is DBpedia database that provides the data in RDF triples in various formats like Ntriples, Turtle, XML, JSON, etc. We have used the DBpedia database to detect the numerical outliers in a scalable manner. The figure 2.2 represents the LOD as per published in August 2017.

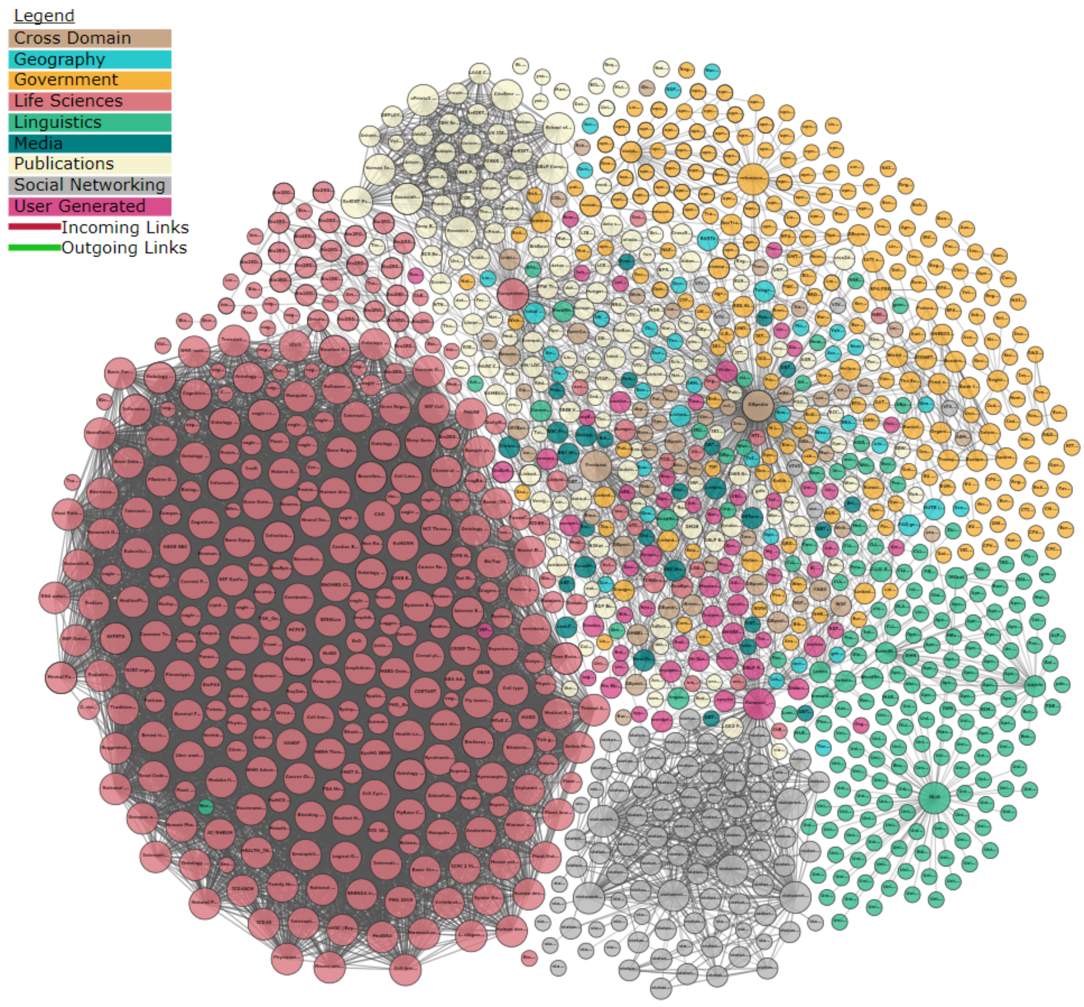


Figure 2.2: The Linked Open Data as per August 2017[6]

[6]http://lod-cloud.net/

2.3 RDF

Wikipedia is one of the largest reference websites used by millions of people across the globe. There are about 71,000 active contributors are working on more than 47,000,000 articles in 299 languages[7]. Such a huge amount of information present over the Internet lacks in the semantics before the introduction of RDF. The information present on the web can be expressed in the form of triples which is the combination of a subject, a predicate, and an object. These collections of triples are known as Knowledge Graphs (KGs). Knowledge graphs represent real-world entities and their interrelations, organized as a graph. A KG defines classes and relations among these classes in a schema allows for linking arbitrary entities with defined relationships and, cover a variety of domains.

Subject and object of the triples are the resources, represented through unique URIs. The predicate depicts the relationship between the resources. For example, the Wikipedia page about India consists of information about India like its capital, population, states, etc. The sentence "India has capital Delhi" is described in RDF as shown in figure 2.3 :-

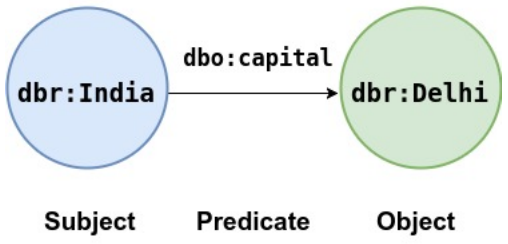


Figure 2.3: An example of the RDF triple

In figure 2.3, dbr:India acts as a subject, with property dbo:capital and dbr:Delhi acts as an object for the given sentence. RDF can be recognized as a directed graph with two vertices (resources) join by an arrow representing a relationship between them. The subject, predicate, and object of a triple are called RDF nodes. There are three different types of nodes:-

1. URI used to distinguish resources uniquely worldwide. Within RDF, URI references may be used to identify any object, including Web resources such as HTML documents, real-world things such as products, organizations, and persons, and abstract concepts such as terms, classes, or property types.
2. Blank nodes are unnamed nodes and represented with "underscore". The term "blank" refers to the fact that blank nodes do not have identifiers. Blank

[7] `https://en.wikipedia.org/wiki/Wikipedia:About`

nodes are helpful for describing multi-component structures, reification[8] and offer protection of the inner information[9].

3. Literals are used to model property value and can be interpreted through datatype like integer, double, date or date event. Literals without datatype treated as strings. Literals cannot act as a starting point of RDF graph and properties can never be labeled with literals.

There are different techniques to represent RDF data, named as RDF data serializations. These are defined as follows:-

1. N-Triples[10]:- Plain text format for encoding an RDF graph.
2. N-Quads[11]:- Similar to N-Triples but they allow encoding multiple graphs
3. Turtle[12]:- Compatible with N-Triple, uses abbreviations for URL
4. JSON-LD[13]:- A JSON-based format to serialize Linked Data
5. N3 or Notation3[14]:- Superset of RDF, extends RDF by adding formulae, variables, etc.
6. RDF/XML[15]:- Syntax based on XML.

We can represent the RDF in one or other manner as explained above according to the requirement. We have adopted RDF N-triples data serialization in our master thesis.

```
@prefix dbo:    <http://dbpedia.org/ontology/> .
@prefix dbr:    <http://dbpedia.org/resource/>
dbr:Bonn    ns37:areaTotal     "141.06"^^ns33:squareKilometre ;
            dbo:areaCode       "BN" , "0228" ;
            dbo:areaTotal       141060000.0 ;
            dbo:elevation       60.0 ;
            dbo:leaderTitle     "Lord Mayor"@en ;
            dbo:postalCode      "53111\u201353229" ;
            dbp:location        "Bonn"^^rdf:langString ;
            dbp:singleLine      "yes"^^rdf:langString ;
            dbp:source           dbr:Deutscher_Wetterdienst ;
            dbp:type            "city"^^rdf:langString .
```

[8] https://www.w3.org/TR/rdf-primer/#reification
[9] https://en.wikipedia.org/wiki/Blank_node
[10] https://www.w3.org/TR/n-triples/
[11] https://www.w3.org/TR/n-quads/
[12] https://www.w3.org/TR/turtle/
[13] https://www.w3.org/TR/json-ld/
[14] https://www.w3.org/TeamSubmission/n3/
[15] https://www.w3.org/TR/rdf-syntax-grammar/

Listing 2.1: Example for Turtle Data Serialization

The listing 2.1 is an example of Turtle RDF data serialization.

2.4 Resource Description Framework Schema (RDFS)

The RDF Schema (RDFS) language gives a data modeling vocabulary for RDF data[16]. The RDFS help to define the semantics over RDF data. For example, we have formulated the following illustration in listing 2.2 to observe the importance of RDFS in RDF data model.

```
@prefix dbo:    <http://dbpedia.org/ontology/> .
@prefix dbr:    <http://dbpedia.org/resource/> .
dbr:India    dbo:capital    dbr: New_Delhi.
```

Listing 2.2: RDF triple for subject India

However, there is a difficulty in analyzing the above listing due to the lack of vocabulary knowledge about the subject, predicate, and object. RDFS make it easy by providing a vocabulary for RDF triples. RDFS groups similar resources into a "class" and describes a relation between them. Each resource is an instance of a class, and a class can be an instance of itself as well. The membership of any entity or resource to its class is defined by rdf:type property and each resource can belong to many classes. Each property is associated with a domain and range. The listing 2.3 is extension of the listing 2.2 and provides vocabulary about the subject, property, and object.

```
@prefix dbo:    <http://dbpedia.org/ontology/> .
@prefix dbr:    <http://dbpedia.org/resource/> .
dbr:India      rdf:type       dbr:Country.
dbr:Delhi      rdf:type       dbr:City.
dbo:capital    rdf:type       rdf:Property .
dbr:India      dbo:capital    dbr:New_Delhi .
```

Listing 2.3: Listing 2.2 with RDFS knowledge

Classes can be arranged in a hierarchy by using rdfs:subclassOf property. Table 2.1 and 2.2 are the example of RDFS classes and properties. The knowledge about

[16]https://www.w3.org/TR/rdf-schema/

RDF and RDFS Classes	
Class	Description
rdfs:Resource	Superclass of all the class
rdfs:Classs	class of resources that are RDF classes
rdfs:Literal	the class of literal values such as strings and integers
rdfs:Datatype	the class of datatypes
rdf:XMLLiteral	the class of XML literal values
rdf:Property	the class of RDF properties
rdf:HTML	represents the class of HTML literal value

Table 2.1: A List of RDF and RDFS Classes

RDF and RDFS Properties	
Class	Description
rdfs:range	state the values of a property are instances of one or more classes.
rdfs:domain	domain of subject property
rdf:type	used to define that subject is an instance of a class
rdfs:subClassOf	define hierarchical relationship between classes
rdfs:subPropertyOf	instance of rdf:property and creates relation between properties
rdfs:label	provide a human-readable version of a resource's name.
rdfs:comment	provide a human-readable description of a resource.

Table 2.2: A List of RDF and RDFS Properties

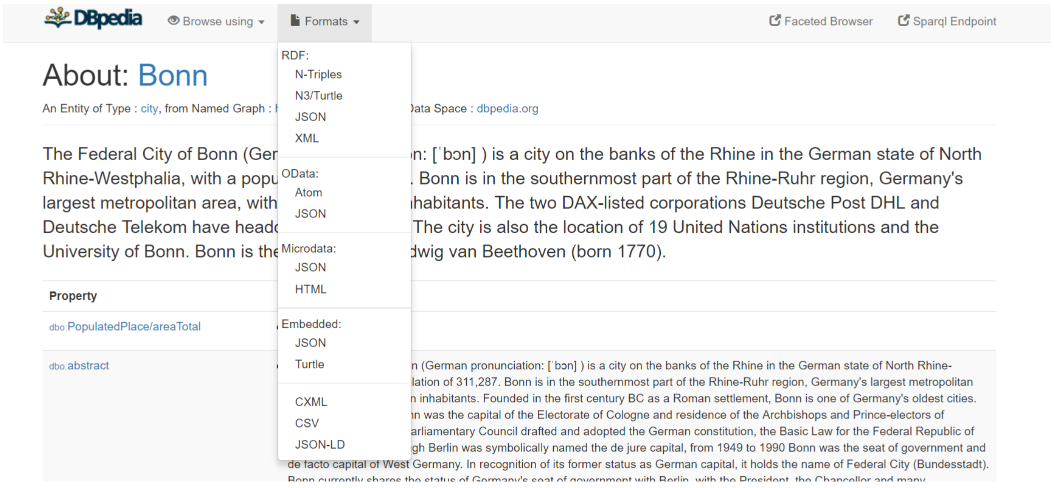
RDFS vocabulary is important for our task. We have grouped the similar subjects into classes based on their types and find the numerical outliers.

2.5 RDF Database

Creating RDF triples from web resources is a vital task and many organizations like DBpedia, Wikidata, Freebase, Yago, etc. are continually creating RDF triples. We have described some of the databases in the following subsections.

2.5.1 DBpedia

DBpedia (from "DB" for database) is a project aiming at obtaining structured content from the information created in the Wikipedia project[17]. DBpedia released its first dataset in 2007 by the developers of University of Mannheim and University of Leipzig. Wikipedia consists of information like images, numerical attributes (e.g., population, height), links to external web pages, etc. DBpedia extracts all these information, forms an RDF graph and represent them in different RDF data serialization format as shown in the figure 2.4. The aim to create RDF data out



About: Bonn

An Entity of Type : city, from Named Graph : h ata Space : dbpedia.org

The Federal City of Bonn (Ger n: [ˈbɔn]) is a city on the banks of the Rhine in the German state of North
Rhine-Westphalia, with a popu . Bonn is in the southernmost part of the Rhine-Ruhr region, Germany's
largest metropolitan area, with habitants. The two DAX-listed corporations Deutsche Post DHL and
Deutsche Telekom have head The city is also the location of 19 United Nations institutions and the
University of Bonn. Bonn is the dwig van Beethoven (born 1770).

Property	
dbo:PopulatedPlace/areaTotal	
dbo:abstract	n (German pronunciation: [ˈbɔn]) is a city on the banks of the Rhine in the German state of North Rhine- lation of 311,287. Bonn is in the southernmost part of the Rhine-Ruhr region, Germany's largest metropolitan n inhabitants. Founded in the first century BC as a Roman settlement, Bonn is one of Germany's oldest cities. n was the capital of the Electorate of Cologne and residence of the Archbishops and Prince-electors of arliamentary Council drafted and adopted the German constitution, the Basic Law for the Federal Republic of gh Berlin was symbolically named the de jure capital, from 1949 to 1990 Bonn was the seat of government and de facto capital of West Germany. In recognition of its former status as German capital, it holds the name of Federal City (Bundesstadt).

Figure 2.4: DBpedia User Interface

of Wikipedia is to enhance the intelligence of the web. DBpedia is available in a different version of languages, e.g., Dutch, German, French, Russian, etc.

2.5.2 Wikidata

Wikidata is also an RDF database with a primary aim to collect structured data to provide support for Wikipedia, Wikimedia Commons, the other wikis of the Wikimedia movement[18]. An RDF resource is known as an item, and their notation starts with 'Q' appended by some number. An item is a subject or object of an RDF triple. For example, an item can be a place, person or animal. Statements define attributes of an item and consist of a property and a value[18]. The property of an item is represented as P and also followed by a number. For example, New Delhi is a capital of India is formulated in Wikidata as shown in the table 2.3.

[17]https://en.wikipedia.org/wiki/DBpedia
[18]https://www.wikidata.org/wiki/Wikidata:Introduction

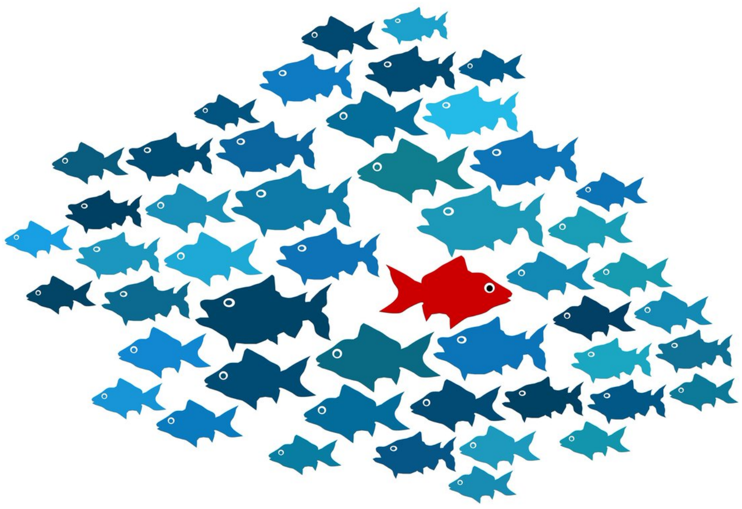


Figure 2.5: Outlier Example[22]

RDF Triple		
Subject	Predicate	Object
New Delhi	Capital	India
Q987	P36	Q668

Table 2.3: Wikidata Dataset Example

In table 2.3, A subject "New Delhi" is represented as Q987[19], a property "Capital" is described as P36[20] and an object "India" as Q668[21] in Wikidata. Wikidata offers database dumps in JSON, RDF and XML formats.

2.6 Outlier Detection

Outlier or anomaly detection is a technique for "finding patterns in data that do not conform to the expected normal behavior [3]." Anomalies detection has numerous applications in areas such as security, finance, health-care, and many more. For example, any abnormal banking transaction may be considered as a fraud or an outlier.

[19]`https://www.wikidata.org/wiki/Q987`
[20]`https://www.wikidata.org/wiki/Property:P36`
[21]`https://www.wikidata.org/wiki/Q668`

The leading cause of outliers in a dataset may be because data processing error, data capturing error or human errors (where users enter the wrong value unintentionally). We have focused our thesis on numerical outliers present in the RDF dataset. The techniques like Mean and Standard Deviation Method, Inter Quartile Range (IQR) and Median Absolute Deviation (MAD) are the valid options for the detection of numerical outliers. We will select one method out of these three methods and apply in our algorithm. The figure 2.5 shows the comprehensive understanding of outliers, i.e., a group of fishes is moving in one direction whereas red fish is going in the opposite direction, showing an anomalous behavior.

2.6.1 Selection of Outlier Detection Method

[12] surveyed different methods used by the researchers to find the outliers. Mean, and Median are the two candidates for finding the outliers in a univariate data. As compared to median, mean is more sensitive to the outliers and is explained with the help of breakdown point. The estimator's breakdown point is the maximum proportion of observations that can be infected (i.e., set to infinity) without forcing the estimator to result in a false value (infinite or null in the case of an estimator of scale) [12]. For example, if any value is set to infinity in the data, the whole mean shifts to infinity. Hence the means has zero breakdown point that makes it unsuitable for the calculation of outliers. Whereas Median breakdown value is about 50%, meaning that the median can resist up to 50% of outliers. MAD has also the same breaking point, whereas IQR has a breakdown point of 25%. We can choose between MAD and IQR to find the outliers in the RDF data.

2.6.2 IQR

IQR method is based on finding the first quartile (Q1), Median (M) and third quartile (Q3) of the given numerical dataset. Q1 is the median of all the values smaller than the median M whereas the Q3 is the median of all the values higher than the median M. IQR is the difference between Q3 and Q1. The data points which are smaller than Q1-1.5*IQR and greater than Q3+1.5*IQR is considered as outliers. The constant value 1.5 depends upon the distribution of the data and can be adjusted accordingly.

For example, we have dataset D which equals to:

$$D = 1, 5, 7, 9, 11, 12, 50 \tag{2.1}$$

[22]https://towardsdatascience.com/a-brief-overview-of-outlier-detection-techniques-1e0b2c19e561

$$Q1 = 5, \quad Q3 = 12 \tag{2.2}$$

$$IQR = Q3 - Q1 = 12 - 5 = 7 \tag{2.3}$$

$$x = Q1 - 1.5 * IQR = 5 - 1.5 * 7 = -5.5 \tag{2.4}$$

$$y = Q3 + 1.5 * IQR = 12 + 1.5 * 7 = 22.5 \tag{2.5}$$

Any data point that is less the x and greater than y is considered as an outlier. In the above example, the value 50 is an outlier since it is greater than 22.5.
In our case, we have used a DBpedia RDF dataset which is usually in the form of N-triples. We have considered only those triples which have literals either integer or double and applying IQR on these numerical literals.

2.6.3 MAD

MAD is another way to detect numerical outliers in a univariate data. The MAD measure the variability of a univariate sample of quantitative data in statistics[23]. It is more resilient to outliers in a data set than the standard deviation method[24]. The MAD is defined as the median of the absolute deviations from the data's median for a univariate data $X_1, X_2, ..., X_n$ as follows:
$MAD = b * median\ (\ |X_i - \ median(X)|\ )$
For example, we have a univariate dataset D.

$$D = \{1, 5, 7, 9, 11, 12, 50\} \tag{2.6}$$

$$Median = 9 \tag{2.7}$$

$$X_i - \ median(X) = \{-8, -4, -2, 0, 2, 3, 41\} \tag{2.8}$$

$$|X_i - \ median(X)| \ = \{8, 4, 2, 0, 2, 3, 41\} \tag{2.9}$$

Arranging the datasets in ascending order to find the median for the next iteration.

$$|X_i - \ median(X)| \ = \{0, 2, 2, 3, 4, 8, 41\} \tag{2.10}$$

$$median\ (\ |X_i - \ median(X)|\ ) = 3 \tag{2.11}$$

For normally distributed data, b equals 1.4826[23].

$$MAD = b * median\ (\ |X_i - \ median(X)|\ ) = 3 * 1.4826 = 4.4478 \tag{2.12}$$

[23]https://en.wikipedia.org/wiki/Median_absolute_deviation
[24]https://en.wikipedia.org/wiki/Standard_deviation

$$A = Median + 2.5 * MAD = 9 + 2.5 * 4.4478 = 20.1195 \tag{2.13}$$

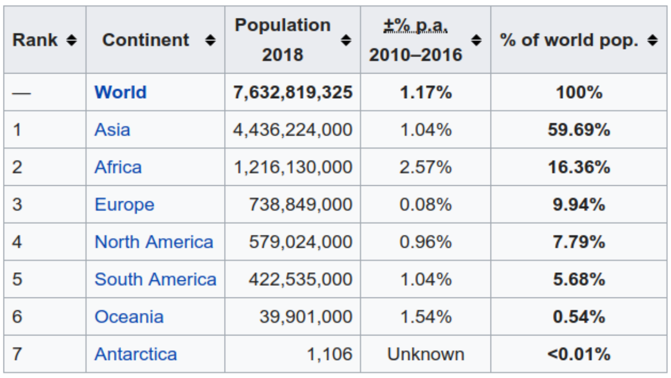
$$B = Median - 2.5 * MAD = 9 - 2.5 * 4.4478 = -2.1195 \tag{2.14}$$

Any value in set D which is greater than A and less than B is considered as an outlier. In the above example, the dataset D has only one outlier, i.e., 50 (greater than A).

2.7 Anomalies in RDF Data

Anomalous behavior of data is widespread, and it leads to the necessity of detecting anomalies to make the data more qualitative. In RDF dataset, anomaly behavior can be observed at various positions in the triples. There are different types of anomalies in the RDF N-triples:-

1. **Numerical Anomaly**:- In this type of anomaly, numerical literals present at object position in N-triples show an unusual behavior when compared with other similar literals. These outliers have categorized into natural outlier and real outlier.



Rank ⬍	Continent ⬍	Population 2018 ⬍	±% p.a. 2010–2016 ⬍	% of world pop. ⬍
—	**World**	**7,632,819,325**	**1.17%**	**100%**
1	Asia	4,436,224,000	1.04%	**59.69%**
2	Africa	1,216,130,000	2.57%	**16.36%**
3	Europe	738,849,000	0.08%	**9.94%**
4	North America	579,024,000	0.96%	**7.79%**
5	South America	422,535,000	1.04%	**5.68%**
6	Oceania	39,901,000	1.54%	**0.54%**
7	Antarctica	1,106	Unknown	**<0.01%**

Figure 2.6: Population of seven Continents[25]

[25] https://en.wikipedia.org/wiki/List_of_continents_by_population

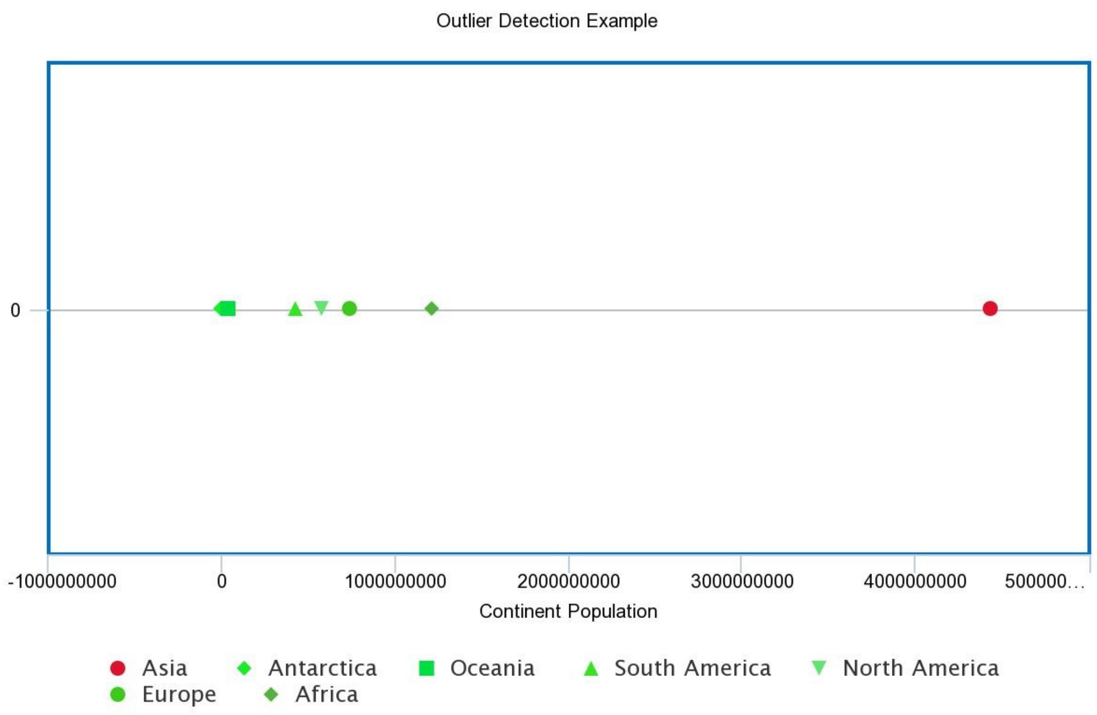


Figure 2.7: Population of Asia acts as a natural outlier in a group of Continents

In figure 2.7, the population of Asia is considered as an outlier and is an example of a natural outlier. The other continents have a comparable population which leads to isolation of Asia from the rest of the continents.

2. **Domain/Range Anomaly**:- It occurs when the domain and range of the properties are modified in the RDF data. Every property associated with domain and range value defined in the RDFS. It might be the case that the domain and range value in the RDF data does not match with the RDF schema which leads to unusual behavior.

```
@prefix dbo:     <http://dbpedia.org/ontology/> .
@prefix dbr:      <http://dbpedia.org/resource/>.
dbr:India  rdf:type dbr:PopulatedPlace.
ns1:areaTotal      rdfs:domain    dbo:PopulatedPlace .
ns44:areaTotal      rdfs:range      ns5:squareKilometre .
dbr:India       ns44:areaTotal      "3287591.04790256"^^ns43:string.
```

Listing 2.4: Anomaly behavior based on Domain/Range of RDF triples

The above-formulated listing 2.4 shows that property areatotal has range equals squareKilometre in RDF Schema whereas in RDF data showing that India has areatotal 3287591.04790256 with datatype string. It can lead to the incorrect results in our method as we have considered only literals with datatype integer, double and nonNegativeInteger. Although syntactically areatotal has a double value in the RDF data, it is not considered in the final result due to the wrong assignment of the datatype, i.e., string.

2.8 Big Data

Big data is a term which means a large volume of data that is difficult to handle by a single machine with traditional techniques. Since the onset of the digital age, the enormous amount of data is captured from numerous devices every day. Devices like phones, cameras, laptops, automobiles are the primary source which creates such voluminous data. Often, Big data is characterized by four V's- Volume, Velocity, Variety, and Veracity. Volume here refers to the size of data which may be in gigabytes, terabytes or more than that. The rate at which the data is growing continuously can be expressed as Velocity. The abnormality and noise in the data called Veracity. The noisy data produces unusual behavior in the data and need to be cleaned before the processing step. Variety refers to the heterogeneity in the data collected from numerous devices that can be structured,semi-structured and non-structured. Structured data can be processed quickly and requires no cleaning steps. However, semi-structured and unstructured data need preprocessing (cleaning the data) before applying big data algorithms for extracting useful information or trends from them. There are two modes of processing the Big data- offline or online mode. In an offline mode, the data is saved for a particular period, accompanied by the processing step. This type of mode is useful for historical or archive records. We have worked with such a type of data in our master thesis. Online mode can be described as real-time data processing and gives a steady output of data.

Since big data is characterized by four V's as shown in figure 2.8, serial computation over these large datasets is unable to work at all. There is a need for a distributed or parallel computing framework which can distribute the datasets across the cores or networks and makes the processing faster. Currently, there are some open source frameworks available on the web namely Hadoop, Spark or Flink. The application of any of these frameworks depends on the type of data and as well as on the output. We have used the Spark framework for processing the data with Hadoop Distributed File System (HDFS) as a storage unit. To get more insight

[26] `http://www.bluecoppertech.com/big-data-the-new-gold/`

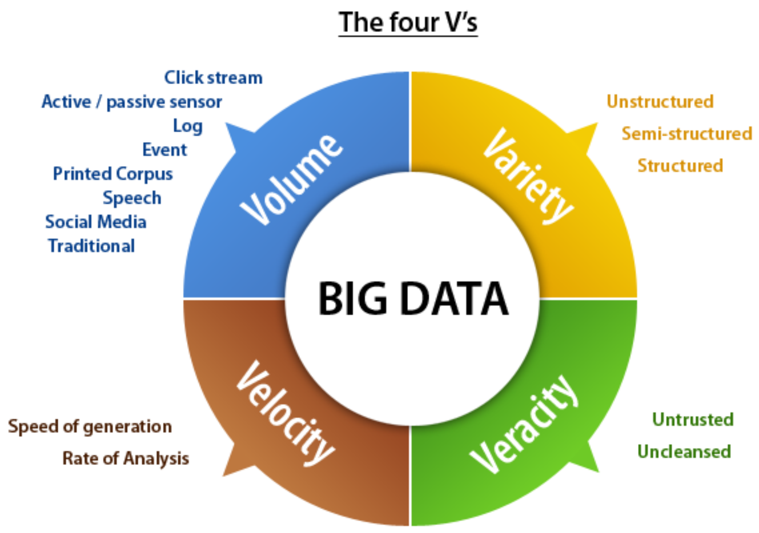


Figure 2.8: Four V's of Big Data[26]

about these frameworks, Spark and HDFS are discussed in details in next sections.

2.8.1 Spark

Apache Spark is a fast and general-purpose cluster computing system and accessible through Java, Python, and Scala[27]. The Spark project is consolidated under one stack, consists of several independent elements. Each element is designated for the specific task. The description of the spark components are as follows:-

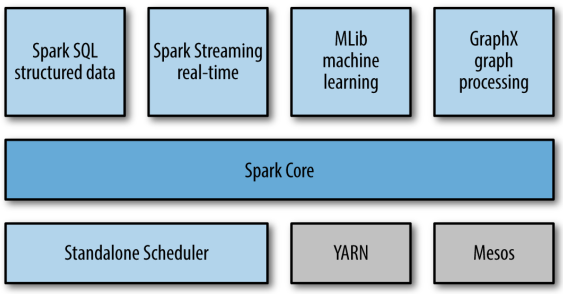


Figure 2.9: The Spark Stack[28]

[27]https://spark.apache.org/docs/latest/
[28]https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch01.html/

1. **Spark Core**:- It Includes basic functionalities of spark like task scheduling, memory management, fault recovery and interacting with the storage system. It is also home to Resilient Distributed Dataset (RDD) API.
2. **Spark SQL**:- This component of the spark stack deals with structured data. The basic SQL operations can be performed on data the data via SQL.
3. **Spark Streaming**:- It is responsible for the processing of live stream data.
4. **MLib**:- Spark comes with a Machine learning library which contains various Machine learning algorithms related to clustering, classification, regression, etc.
5. **Graphx**:- This component of spark is for processing graphs, and manipulating them with the help of graph operations.
6. **Cluster Managers**:- Spark is designed for distributed computing where parallel operations run on various computer nodes. Spark uses cluster managers like Apache Mesos, Hadoop yarn, and standalone scheduler.

Resilient Distributed Dataset (RDD)

RDD is a collection of objects which are immutable and distributed across the clusters. In spark, everything is expressed as RDD. We can create RDD, and manipulate it by using transformations and actions. There are two ways by which one can create RDD:-

1. By loading an external dataset. For example, reading RDF N-triple file from the external storage system.
2. By distributing a collection of objects.

Two types of operations that are performed on RDD -Transformation, and Action. Transformation on RDD means creating a new RDD from previous ones. An action is defined as computing a result based on RDD and returning it to the driver program to show it on the console or save it to the external storage system. Lazy evaluation of Spark achieves fault tolerance and also optimizes the performance. In lazy evaluation, execution of RDD will not start until it encounters an action. Spark makes a lineage graph for all the transformations and executes when an action is triggered. This lineage graph is useful when some partition is lost, and spark can recover with the assistance of lineage graph.

Spark has inbuilt functions for transformations and actions that run on the cluster in a distributed way to make the computation faster. We can apply inbuilt functions on RDD and get desired results. There are many spark functions available, but table 2.4 shows some of the standard transformations and actions supported by

Apache Spark. Spark supports HDFS for reading and writing data. We give a short introduction of HDFS in the next section.

2.8.2 Hadoop Distributed File System (HDFS)

The HDFS is a file storage system which is designed especially to store large data in a distributed fashion. It follows the master-slave architecture -A single Namenode which acts as a master and number of slaves are known as Datanodes. Usually, a file is split into many blocks of default size 64MB or 128 MB and stored in the data nodes. The data is also replicated in various data nodes to make the system more fault tolerant. The HDFS file has a default replication factor of three. It means that each block is replicated in three data nodes. All the data nodes send their heartbeat as well as Blockreport messages to Namenode. A Blockreport contains the list of data blocks that a DataNode is hosting. Namenode stores the metadata information about the data stored at the data nodes.

HDFS Goals[29]

1. **Hardware Failure**:- Since HDFS works in a cluster of nodes, there is a probability of failure of any node at any instant. The detection of these failed nodes and their quick recovery should be the prime aim of the HDFS.
2. **Large Datasets**:- HDFS is designed for storing large files of size vary from gigabytes to petabytes. So, it should support scalability, i.e., many nodes can be added to store such huge files.
3. **Streaming Data Access**: HDFS is designed for batch processing rather than real-time processing. High throughput of data access is preferred rather than low latency of data access.

2.9 Scala

Spark framework supports Scala, Python, Java and R. Scala being a functional as well as object-oriented language is beneficial for implementing a scalable outlier detection algorithm. Scala is a high-level programming language which stands for "scalable language". It runs on JVM and also compatible with Java libraries. Not only scalability which makes Scala popular but some others factors that are discussed as follows:-

[29] `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`
[30] `https://hadoop.apache.org/docs/r1.2.1/images/hdfsarchitecture.gif`

Function Name	Description
`map`	Returns a new RDD by applying a function to all elements of the RDD.
`filter`	Returns a new RDD containing only the elements that satisfy a given condition(s).
`count`	Returns the number of total elements.
`distinct`	Returns a new RDD containing the unique elements.
`reduce`	Reduces the elements from the RDD using the specified commutative and associative binary operator.
`collect`	Returns an array that contains all of the elements of the RDD.
`take`	Takes the first specified number of elements.
`first`	Returns the first element of the RDD (take(1)).
`flatMap`	Returns a new RDD by first applying a function to all elements of the RDD, and then flattening the results.
`foreach`	Runs a function on each element of the dataset.
`union`	Return the union of two RDDs (combine RDDs).
`cartesian`	Returns the cartesian product of two RDDs.
`join`	It works with two RDDs (each consists of pairs). Both rdd should have the same key.
`persist`	Persists the RDD with the default storage level (memory only).
`unpersist`	Mark the RDD as non-persistent, and removes it from memory and disk.
`partitions`	Get the array of partitions of this RDD, taking into account whether the RDD is checkpointed or not.
`mapPartitions`	Returns a new RDD by applying a method to each partition of the (existing) RDD.
`sparkContext`	The SparkContext is used to create the RDD.
`saveAsTextFile`	Saves the RDD as a text file.
`reduceByKey`	Runs a reduce operation and group data by key.
`repartition`	Shuffles the data across the network.

Table 2.4: A List of some important RDD methods

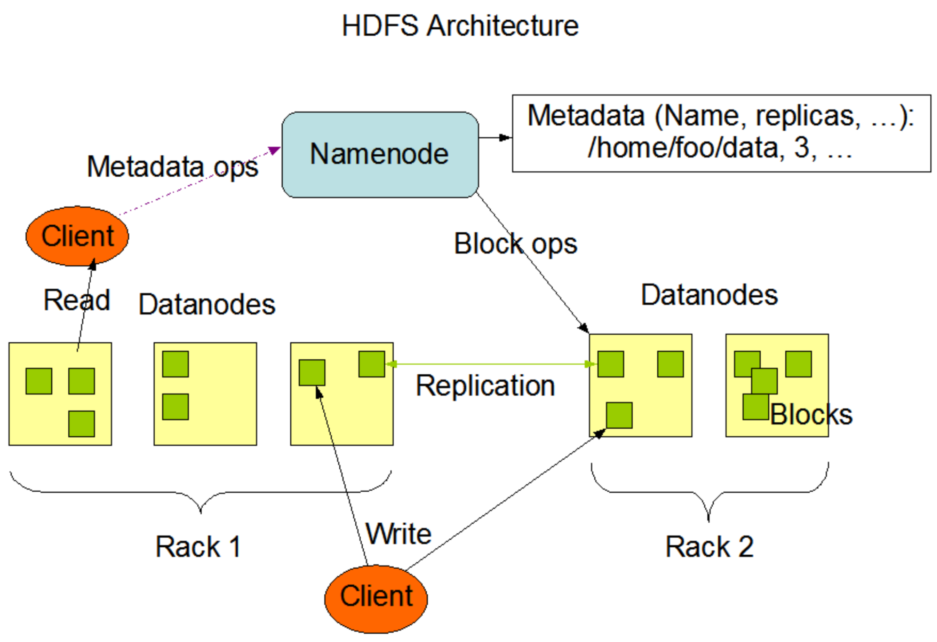


Figure 2.10: HDFS Architecture[30]

1. **Compatibility with Java**:- Scala is compatible with Java and its libraries. It can call Java methods, inherit Java interfaces and can do many things as we do in Java.
2. **Conciseness**:- Scala makes the code shorter. A code with 200 lines in Java can be written within twenty or thirty lines.
3. **Type inference**:- Scala is a typed inference language. We do not have to mention the type explicitly, and it automatically detects the type of variable.
4. **Object-oriented**:- Scala is purely an object-oriented language. Every value/-functions act like an object.

3 Related Work

This chapter deals with the past research work related to outlier detection in RDF data carried by the researchers. The research contributions and published work in this filed is limited to a subset of RDF data. There hasn't been a dedicated research work particularly on numerical anomaly detection of RDF data on a large dataset. In this chapter, we present the significant research contributions in this field.

3.1 Detecting Incorrect Numerical Data in DBpedia

[17] deals with the detection of incorrect numerical data in DBpedia. The process of finding numerical outliers performed in two steps: In the first step, the N-triples subject grouped according to their rdf: type, and in next step, the outliers detected by employing different outlier detection techniques. The clustering process used the FeGeLOD framework developed by [15] for generating the vectors. FeGeLOD collected the information for all subjects and creates a binary feature for each type. A threshold p is applied to filter out features that were either too generic, appearing in over p% of all cases, or too specific, appearing in less than $1-p\%$ of all cases. The actual clustering performed with the Estimation Maximization (EM) algorithm [4], using the implementation in WEKA [6]. The clustering of the subjects by their type was a bottleneck of this research paper as its runtime over 24 hours for the datasets containing only two properties- DBpedia-owl:populationTotal and DBpedia-owl:elevation.

This paper gives us the insight to develop an algorithm for our thesis. We have used the same strategy for clustering the subjects, i.e., clustering based on based on rdf:type. We have grouped the subjects by considering only DBpedia's rdf:type and used the Spark framework for making our algorithm distributed. We have tried to use the FEGeLOD method to for clustering the subjects, but this tool is no more updated and now developed further as Rapid Miner Linked Open Data Extension.

Also, the comparison among various outlier detection techniques has helped us to choose the best technique. This paper has also compared different outlier detection techniques, i.e., Inter Quartile Range (IQR), Kernel Density Estimators (KDE) [13] and dispersion estimators. However, combining IQR with KDE and grouping by single type produced the best results. Since IQR is performing well, we have used the IQR method in our approach to detect the numeric outliers. This paper only considered the limited kind of triples, i.e., triples with three properties, DBpedia-owl:populationTotal, DBpedia-owl:height, and DBpedia-owl:elevation. Whereas our method is scalable as well as generic and results in producing outliers from a large file, i.e., 16.6 Gigabytes dataset.

3.2 Detecting Errors in Numerical Linked Data using Cross-Checked Outlier Detection

[5] introduced two independent outlier detection approaches which give more reliable results. This work is much closer to the [17]. In the first step, it produced the significant subpopulation and then outlier detection applied on the subpopulation. In the second step, it used dataset interlinks, i.e., owl:sameas to verify whether the outlier detected in the first step was a natural outlier or not.

Dataset inspection plays an important role in the detection of outliers. The relevant information like the number of instances, the name of properties and how often each property was used with a numerical value at the object position determined. Some properties with a low number of numerical literals removed in this step hence reduced the amount of data to be processed in further steps. Next step was to find the most promising subpopulation so that the outlier detection technique could produce some meaningful results. This exploration organized in a lattice where the root node consists of property and a corresponding number of instances, and the rest of the lattice consists of instances which were not yet explored with some constraint. The lattice was pruned based on three criteria-

1. Prune the nodes with a lower number of instances.
2. Instance reduction ratio:- if the additional constraint leads to a reduction of less than 1% of the number of instances in the child node compared to the parent node, the child node pruned.
3. Kullback-Leibler divergence [10]- if the KL divergence is low, it means that the newly created child subpopulation is not changed from a parent. In this case, prune the newly created child node.

Once the lattice created, the next step was to find the outliers on all the unpruned nodes of the lattice. The results of the outlier score stored with a set of constraints which leads to the corresponding instance set [5]. The classification of outliers into natural or real accomplished with the help of the Linked data interlinking property. The resulting outlier instance value compared with different datasets. This procedure helped in better handling natural outliers and thus in reducing the false positive rate.

In summary, A lattice was created for each property and pruned according to the three criteria defined above. While using this approach in our work, we have come across a difficulty to handle a large number of nodes in the lattice. Since every subject associated with multiple rdf:type, that leads to a large number of nodes in the lattice and difficult to process.

3.3 Identifying Wrong Links between Datasets by Multi-dimensional Outlier Detection

[14] deals with finding wrong links between the datasets. Links between different datasets created for scalability purposes by a heuristic method which lead to occasional wrong links [14]. The Multi-dimensional outlier detection approach used to improve the quality of relations between datasets. It is a three-step process:-

1. Representation of each link as a feature vector. Two strategies used to create feature vectors. First one was using all direct types which means a binary feature created for each schema class, which set to true for a link if the linked resource has the class defined as its rdf:type. The second one was using all ingoing and outgoing properties. Two binary features created for each data and object property, which set to true if the linked resource was the subject resp. The object of a triple which used the property as its predicate [14].
2. Performed outlier detection on the set of vectors.
3. Assigned outlier score to each link and order them according to outlier score.

The three different databases namely DBpedia [11], Peel [16], and DBTropes [8] used in the experiment to detect wrong links. [14] compared total six multi-dimensional outlier detection approach namely k-NN Global Anomaly Score (GAS), Local Outlier Factor (LOF) [2], Local Outlier Probability (LoOP) [9], Cluster Based Local Outlier Factor (CBLOF) [7], Local Density Cluster Based Outlier Factor (LDCOF), and One-Class support vector machines. All of the techniques mentioned above-used cosine similarity measure to detect the similarity between the links. The evaluation showed good results, as well as this approach, is scalable, it processed link sets in a few seconds to a few minutes, depending on the configuration used. This work has a different domain i.e., finding wrong links between different datasets whereas, in our master thesis, we are focused on finding numerical outliers from the RDF dataset.

4 Methodology

4.1 Problem Description

Semantic Web has gained popularity over the years, and various companies as well as academia, are continually working on extracting structured information from the web resources, especially Wikipedia. As the RDF data collected by these organizations is a massive amount, it is challenging for a single device with sufficient memory and processing speed to process the data. There is a necessity of a distributed processing framework which can process as well as do some manipulation over large datasets for producing desired results. Our master thesis concerning "detecting numerical outliers on a large scale RDF datasets" uses Apache Spark framework to find numeric outliers from the dataset.

4.2 Design Goals

Our design goal is to determine the numerical outliers from large RDF dataset. The dataset is in N-Triples format and follows the subject, predicate and object notation. An example of RDF triples is shown in the listing 4.1:-

```
        dbr:germany    rdf:type    dbo:Country ,
                                   yago:District108552138 ,
                                   yago:WikicatFederalCountries ,
                                   dbo:PopulatedPlace ,
                                   dbo:Location ,
                                   dbo:Place .
        dbr:india      rdf:type    dbo:Country ,
                                   owl:Thing ,
                                   dbo:Location ,
                                   dbo:Place ,
                                   dbo:PopulatedPlace ,
                                   yago:YagoGeoEntity ,
                                   yago:YagoPermanentlyLocatedEntity ,
                                   yago:Region108630985 ,
                                          yago:WikicatCountries ,
                                          yago:WikicatFederalCountries ,
                                          yago:Country108544813 ,
                                          yago:Location100027167 ,
                                          yago:AdministrativeDistrict108491826 .
dbr:France       rdf:type       dbo:Country.
dbr:bonn         rdf:type       dbo:city.
dbr:delhi        rdf:type       dbo:city.
dbr:germany   dbo:populationTotal "82175700"^^xsd:integer.
dbr:india        dbo:populationTotal "1293057000"^^xsd:integer.
dbr:france       dbo:populationTotal  "66736000"^^xsd:integer.
dbr:bonn         dbo:populationTotal "311287"^^xsd:integer.
dbr:delhi        dbo:populationTotal  "18686902"^^xsd:integer.
```

Listing 4.1: Example of RDF triples

The initial twenty-two lines of the above listing 4.1 consist of the RDF schema of the triples, describing that subjects belong to either a country, city or any other class. Each subject may belong to multiple classes. For example, subject Germany has rdf:type location, place or PopulatedPlace. The rest of the listing related to the RDF data depicting numerical literals corresponding to the subjects. For example, Germany is a subject with a type of Country and has population 82175700.

The listing 4.1 is a fraction of the RDF dataset, and there may be millions or billions of triples in a file. Apache Spark helps to process the big RDF dataset by distributing it across the cluster.

4.3 Dataset Inspection

Understanding RDF data is a vital task because without having a comprehensive knowledge of the data, we can not generalize the algorithm. For instance, there are diverse types of rdf:type of a subject in RDF from the different organizations like DBpedia, Yago or Wikidata. We have selected only those rdf:type's that are published by the DBpedia. The selection of DBpedia over others is based on analyzing the patterns in the database. The consideration of all the rdf: type from different organizations may lead to a wrong similarity measure while cohorting[1] the subjects. The approach which we are going to adopt requires a dataset that satisfies the following conditions:-

[1] `http://www.dictionary.com/browse/cohort`

1. **Numerical Literals**:- The RDF data should consist of a sufficient amount of numerical literals as our approach is specific to numeric values. It helps in generalizing the algorithm.
2. **Type of Subject**:- The triples with the numerical literals are cohorted based on similar DBpedia rdf:type of the subject. It thus requires an RDF dataset with all the subjects having DBpedia rdf:type.
3. **Linked Hypernym Dataset (LHD)**:- The Linked Hypernyms Dataset (LHD) provides types in the DBpedia namespace. The LHD is generated from Dutch, English, and German Wikipedia articles. "The types are extracted from the first sentences of Wikipedia articles using Hearst pattern matching over part-of-speech annotated text and disambiguated to DBpedia concepts[2]". LHD provide a more specific as well as more precise type. For instance, "An Asteroid named "1840 Hus" is assigned type dbo:Asteroid in LHD, while DBpedia assigns it the imprecise type dbo:Planet"[2]. So, appending LHD with rdf:type in the dataset gives the accurate cohorts of the classes as described in the next section.

```
dbr: Germany  ns51: hypernym  dbr: Republic.
dbr: India    ns49: hypernym  dbr: Country.
dbr: France   ns53: hypernym  dbr: State.
dbr: Bonn     ns41:hypernym   dbr: City.
dbr: Delhi    ns3: hypernym   dbr: Territory.
```

Listing 4.2: An Example of Linked Hypernym Dataset

4.4 Jaccard Distance- Class Cohorting Measure

The subjects of RDF data share some common traits that are useful for cohorting them. We can not cohort those subjects which do not have any common characteristics among them. For example, India and Germany belong to the same class, i.e., country and have many common properties like population, areatotal, etc. The common characteristics of the subjects can be understood with the help of the rdf:type and LHD, and are beneficial for cohorting the subjects. The listing 4.3 represents the rdf:type and LHD of India. Although LHD of India, i.e., Country is also present in the DBpedia rdf:type, we try to keep hypernym in the pair RDD with a subject as key and value as a set of rdf:type and LHD.

```
dbr:India    rdf:type      dbo:PopulatedPlace ,
```

[2] https://www.sciencedirect.com/science/article/pii/S1570826814001048

```
                                        dbo:Place ,
                                        dbo:Location,
                                        dbo:Country.
      dbr:India   ns45:hypernym   dbr:Country.
```

Listing 4.3: rdf:type and LHD of subject India

The data structure “set” stores unique values and remove duplicates from the set. In the listing 4.4, it creates a pair of RDD from the above listing with a key as a subject and rdf:type/LHD as a value. It consists of unique values in the set, discards additional “Country” from the set. This approach is not utilizing the importance of hypernyms, as hypernyms with matching rdf:type is always discarded.

```
(India,Set{PopulatedPlace,Place,Location,Country})
```

Listing 4.4: Pair RDD with subject as key and value as rdf:type/hypernym

This issue is resolved by constructing the hypernym name unique so that it is not lost while creating a set of rdf:type and LHD. We concatenate the hypernym with a random string, e.g., “hyper”. So, RDD in the listing 4.4 has transformed to RDD in the listing 4.5.

```
(India,Set{PopulatedPlace,Place,Location,Country,hyperCountry})
```

Listing 4.5: Hypernym is appended with random string to resolve the conflict between rdf:type and LHD

The listing 4.5 has made sure that hypernym types are not missed while merging with DBpedia’s rdf:type. The pair RDD in the listing 4.5 is ready for cohorting the subjects. The similarity between the two subjects based on rdf:types/LHD can be calculated through Cosine distance, Jaccard distance or Manhattan distance. Since the intermediate data in our algorithm is in a set format, Jaccard distance is a better option.

The Jaccard coefficient[3] of two sets of strings A and B respectively can be defined as the cardinality of the intersection of sets divided by the cardinality of the union of the sets. The Jaccard distance is complementary to the Jaccard coefficient and can be calculated by subtracting one from the Jaccard coefficient. Jaccard distance is defined mathematically as follows:-

$$J(A, B) = 1 - (|A \cap B|/|A \cup B|)$$

[3] https://en.wikipedia.org/wiki/Jaccard_index

$J(A, B)$ outputs a real number range from zero to one. The value zero represents that the two sets A and B are similar, whereas one indicates that the two sets are dissimilar. The closer the value is to zero, the more similar the two sets are. E.g., the two sets A and B with values

$$A = \{city, PopulatedPlace, Settlement, Place\}$$
$$B = \{city, PopulatedPlace, Settlement, Location\}$$
$$|A \cap B| = 3 \text{ and } |A \cup B| = 5$$
$$J(A, B) = 1 - 3/5 = 0.4$$

In the above example, Jaccard distance between two sets of strings A and B results to 0.4. If the user has set the threshold value equals 0.45, then the sets A and B are considered as similar sets and clustered into one group. It depends how the user assigns a threshold value.

4.5 Dataset Cleaning

Even though the dataset serves all the conditions specified in the section 4.3, it still needs to be cleaned to make it suitable for the algorithm. The lesser the volume of data in the input, faster it is processed through Spark. For example, the triples with abstract and label information as shown in listing 4.6 are in a significant amount in DBpedia dataset, and their removals from the data make it concise. The cleaning steps in the dataset are:-

```
dbr:India  dbo:abstract   ”India,   officially   the Republic of India
                           (IAST: Bh\u0101rat Ga\u1E47ar\u0101jya),
                           is  ..” @en.
           rdfs:label     ”India”@en ,
                          ”India”@it   ,
                          ”Inde”@fr ,
                          ”Indie”@pl .
```

Listing 4.6: List of graph triples with abstract and label literals

1. **Pruning the unnecessary part of the URIs**:- The N-Triples format of the RDF data consists of triplets, i.e., subject, predicate, and object in the form of URIs. The elimination of redundant part of URIs has benefited us in reducing the size of data. For example, representation of India as URI in DBpedia is “< http://dbpedia.org resource/India>”. The “http://dbpedia.org- /resource/” part from the URIs have pruned and kept only “India” in the RDD.

2. **Appropriate Numeric Type**:- The algorithm has considered only those numeric literals which have literal of type xsd:integer, xsd:nonNegativeInteger and xsd:double. The consideration of other number formats like virtrdf:Geometry as shown in listing 4.7 are complicated to process as it comprises of alphanumeric characters.

```
dbr:Russia   geo:geometry          "POINT(90 60)"^^virtrdf:Geometry ,
             dbp:augRecordLowC     "\u221217.1"^^rdf:langString.
```

Listing 4.7: List of graph triples with Numerical Literal

3. **Removing irrelevant rdf:type**:- We need to compare rdf:type and LHD of every subject with every other subject and cohort them. As we have mentioned in the section 4.3, the removal of rdf:type of Wikidata and Yago, and keeping DBpedia rdf:type with LHD has helped in yielding good cohorts of classes.
4. **Removing super types**:- This step comes after step 3. Supertypes are those rdf:type of subjects that are more generic. This link[4] has helped us to visualize the supertypes as it consists of information in a tree structure. For example, rdf:type like Person, Place, Organization, etc. are supertypes because they are present in most of the rdf:type of the subjects. Sometimes their presence in the similarity measure may mislead the result. However, removing supertypes straightforwardly is not a good idea as some subjects in DBpedia contain only the supertype as their rdf type.

   For example, in listing 4.8, the computation of Jaccard similarity on this formulated RDD will output the following result:-

```
(India,Set(PopulatedPlace,Country,Place,Location))
(Bonn,Set(PopulatedPlace,Place,Location,City)))
```

Listing 4.8: RDD consists of supertypes

Jaccard similarity measure between India and Bonn is

$$|India \cap Bonn| = 3 \text{ and } |India \cup Bonn| = 5$$

$$J(India, Bonn) = 1 - 3/5 = 0.4$$

Since it is smaller than the threshold value (suppose the threshold value equals to 0.45), India and Bonn are cohorted, leading to the incorrect results. In this case, intersection on these two sets is performed and supertypes from the intersection are eliminated which lead to the following RDD:-

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

```
                    (India,Set(Country))
                    (Bonn,Set(City)))
```

Listing 4.9: RDD after removing supertypes

The Jaccard similarity measure between India and Bonn on RDD listed in 4.9 is $|India \cap Bonn| = 0$ and $|India \cup Bonn| = 2$

$$J(India, Bonn) = 1 - 0/2 = 1$$

The Jaccard similarity distance 1 indicates that India and Bonn are dissimilar, and not grouped together.

4.6 Techniques to find Outlier

1. **Technique 1**: The RDF data is cohorted according to the rdf:type of the subjects especially DBpedia type as well as Linked Hypernym type. The next step is to create cohorts of properties corresponding to each cohort of classes. The cohorts of properties are input to the Inter Quartile Range (IQR) method and IQR finds the outliers from the numerical data. In figure 4.1, city, country, and continents are cohorted according to their types, and each cohort e.g. city is further grouped according to different properties.
2. **Technique 2**:- For each property, the subjects are cohorted according to their rdf:type particularly DBpedia type as well as Linked Hypernym type. The cohorts of subjects are input to the IQR method to determine the numerical outliers. In figure 4.2, three distinct cohorts are generated, i.e., city, country, continent for a populationTotal property. Although both the techniques seem identical, their implementation in Spark makes them different. We have implemented both the techniques and will discuss their pros and cons in the next section.

4.7 Implementation obstacles and its Solution

We have already discussed the two techniques to find the outliers in RDF data. The creation of the cohorts of classes is the challenging part of the whole algorithm as it takes more resources. The subjects with similar rdf:type, as well as LHD grouped in one cohort and cohorts of similar classes, are created with Jaccard Distance. The Jaccard distance takes two finite sets as input and finds the similarity between them. In our case, the finite sets are the sets of rdf:types and LHD of the subjects. We have to create the input for Jaccard distance so that it can create cohorts of

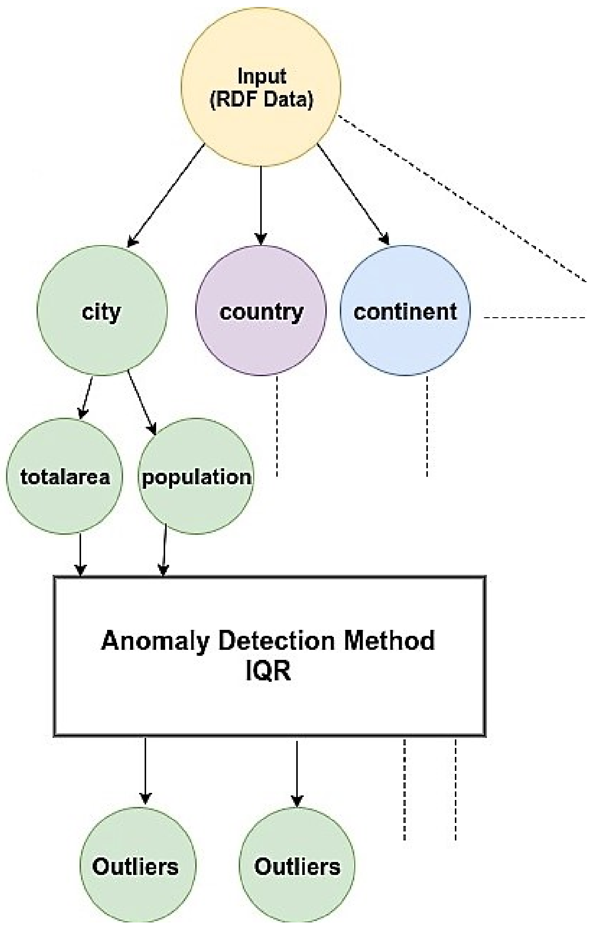


Figure 4.1: Anomaly Detection Model-1

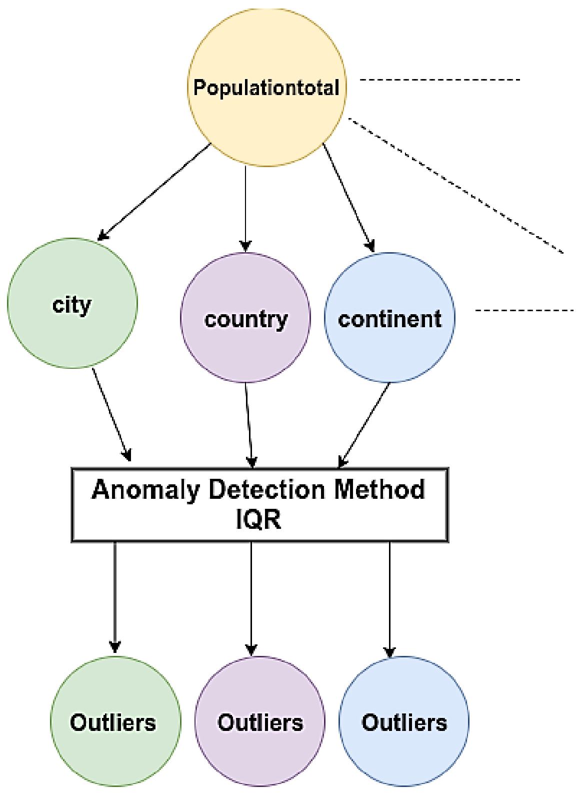


Figure 4.2: Anomaly Detection Model-2

classes efficiently. The next discussion is on the various obstacles faced during the implementation of cohorting of classes and its solution.

4.7.1 Technique 1 implementation obstacles

The cohorts of classes are implemented with the help of three different Spark methods. These methods are spark cartesian, mapPartitions, and DataFrame crossJoin. Since RDD is the main abstraction of Spark, we have tried first to create the cohorts of classes with RDDs.

Spark cartesian method

This is the first step to create input for Jaccard distance for making cohort of classes. In technique one, the RDD from listing 4.10 is to be transformed in the listing 4.11 so that it can serve the desired input to the Jaccard similarity function. Let us suppose that listing 4.10 to be an RDD "x" and listing 4.11 to be an RDD "y" after applying spark cartesian product function.

```
(Subject1,Set1(rdf:types,hypernym))
(Subject2,Set2(rdf:types,hypernym))
```

Listing 4.10: An RDD x with subject as key and value as set of rdf:type/hypernym

```
((Subject1,Set1(rdf:types,hypernym))(Subject2,Set2(rdf:types,hypernym)))
((Subject1,Set1(rdf:types,hypernym))(Subject1,Set1(rdf:types,hypernym)))
((Subject2,Set2(rdf:types,hypernym))(Subject2,Set2(rdf:types,hypernym)))
((Subject2,Set2(rdf:types,hypernym))(Subject1,Set1(rdf:types,hypernym)))
```

Listing 4.11: Cartesian Product of listing 4.10

The RDD in listing 4.11 is created by using spark inbuilt cartesian product. The cartesian transformation returns all possible pairs of (a, b) where a is in the source RDD and b is in the other RDD. The RDD a and b in our case is the same, and self-cartesian product on RDD x given in the listing 4.12.

```
val y = x.cartesian(x)
```

Listing 4.12: Spark Cartesian method

The cartesian product of an RDD with itself can be useful for tasks like user similarity. However, cartesian product is costly for large RDDs. As the datatset size

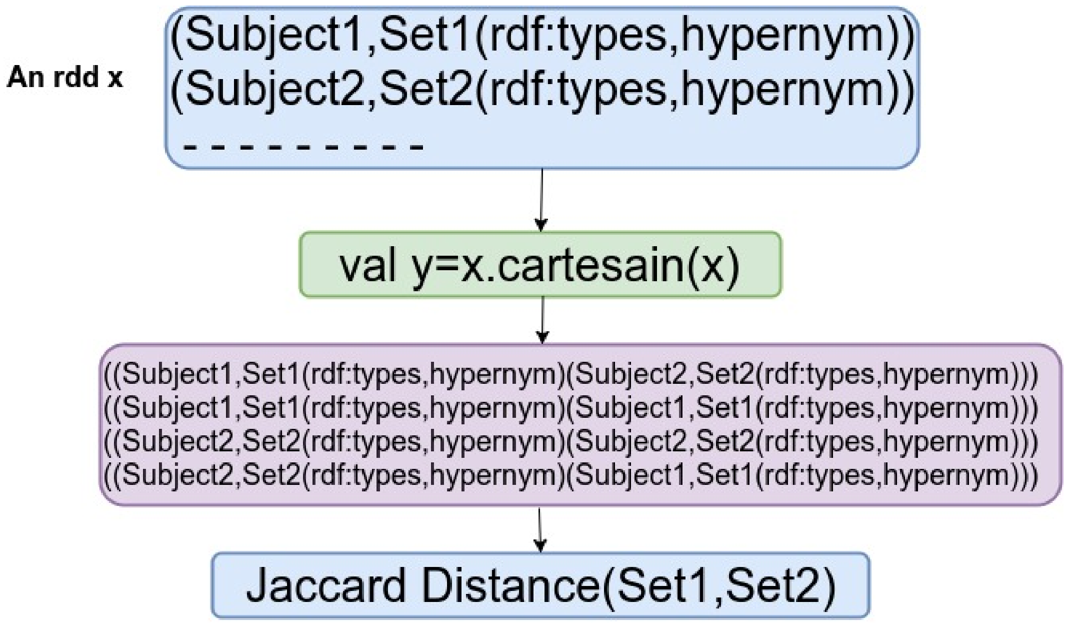

An rdd x
(Subject1,Set1(rdf:types,hypernym))
(Subject2,Set2(rdf:types,hypernym))
- - - - - - - - - -
val y=x.cartesain(x)
((Subject1,Set1(rdf:types,hypernym)(Subject2,Set2(rdf:types,hypernym)))
((Subject1,Set1(rdf:types,hypernym)(Subject1,Set1(rdf:types,hypernym)))
((Subject2,Set2(rdf:types,hypernym)(Subject2,Set2(rdf:types,hypernym)))
((Subject2,Set2(rdf:types,hypernym)(Subject1,Set1(rdf:types,hypernym)))
Jaccard Distance(Set1,Set2)

Figure 4.3: Spark cartesian method to find similarity between subjects

increases, it either makes the processing slow or fails to compute the cartesian product of the RDD. The figure 4.3 is the flow diagram that depicts how Spark cartesian method has created input for Jaccard distance for making cohorts of classes.

Spark mapPartitions method

Another way is to create the RDD "y" given in the listing 4.11 by mapPartitions utility function.

```
val broadcastVar = sparkSession.sparkContext.broadcast(x.collect())
val y = x.mapPartitions({ iter =>
              val k = broadcastVar.value
              for {
                x <- iter
                z <- k
               if x._1.toString() != z._1.toString()
               } yield (x, z)
           })
```

Listing 4.13: Cartesian Product with mapPartitions method

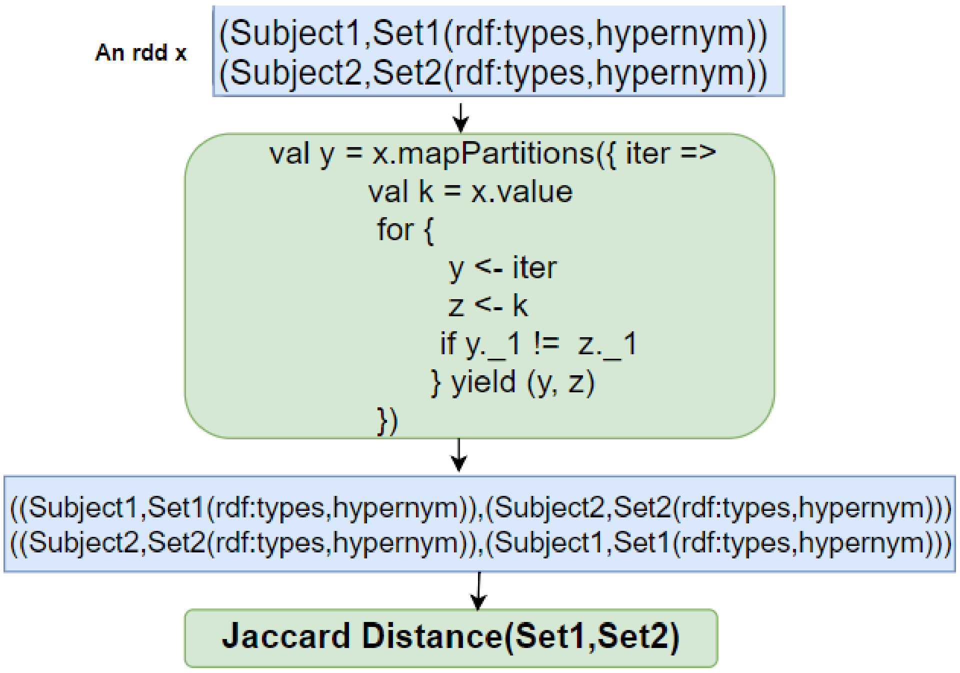

An rdd x
(Subject1,Set1(rdf:types,hypernym))
(Subject2,Set2(rdf:types,hypernym))
val y = x.mapPartitions({ iter =>
val k = x.value
for {
y <- iter
z <- k
if y._1 != z._1
} yield (y, z)
})
((Subject1,Set1(rdf:types,hypernym)),(Subject2,Set2(rdf:types,hypernym)))
((Subject2,Set2(rdf:types,hypernym)),(Subject1,Set1(rdf:types,hypernym)))
Jaccard Distance(Set1,Set2)

Figure 4.4: Spark mapPartitions method to find similarity between subjects

The advantages of using mapPartitions function is that it is called once for each partition. The duplicated elements of RDD "y" are removed while creating it (by employing a condition as shown in the listing). 4.13.

The listing 4.13 shows the implementation of creating the RDD "y" that will act as input to the Jaccard similarity function. Although the mapPartitions method has its advantages, still it does not work for large datasets. Because the RDD "x" is collected and then broadcasted to each worker before applying mapPartitons method. As the dataset size increases, the size of the RDD also increases and big RDDs can neither be collected nor broadcasted. The figure 4.4 is the flow diagram that depicts how Spark mapPartitions method is creating input for Jaccard distance for making cohorts of classes.

Spak DataFrame crossJoin method

The cartesian and mapPartitions methods have not worked well on large RDDs. The advantages of DataFrame over RDDs is that DataFrame does optimization using catalyst optimizer[5]. The DataFrame crossJoin is another function for creating input for Jaccard distance when the input is in the form of DataFrame. Let us suppose that the table 4.1 to be an DataFrame “x” and the table 4.2 to be an DataFrame “y” after applying crossJoin function.

id	Value
Subject1	Set1(rdf:types,hypernym)
Subject2	Set2(rdf:types,hypernym)

Table 4.1: Spark DataFrame example

Cross Join is one kind of join where each row of one dataset “a” is joined with another dataset “b”. Since both the dataset are same, i.e., “x”, it generates four rows as shown in table 4.2.

id1	Value1	id2	Value2
Subject1	Set1(rdf:types,hypernym)	Subject1	Set1(rdf:types,hypernym)
Subject1	Set1(rdf:types,hypernym)	Subject2	Set2(rdf:types,hypernym)
Subject2	Set2(rdf:types,hypernym)	Subject2	Set2(rdf:types,hypernym)
Subject2	Set2(rdf:types,hypernym)	Subject1	Set1(rdf:types,hypernym)

Table 4.2: Cross Join operation on spark DataFame in the table 4.1

The crossJoin function is also time-consuming like the Cartesian method and often should be avoided due to its $O(n^2)$ complexity where n is the size of the DataFrame. The listing 4.14 shows the implementation of crossJoin on DataFrame “x”. The figure 4.5 is the flow diagram that depicts how Spark DataFrame crossJoin method is creating input for Jaccard distance for making cohorts of classes.

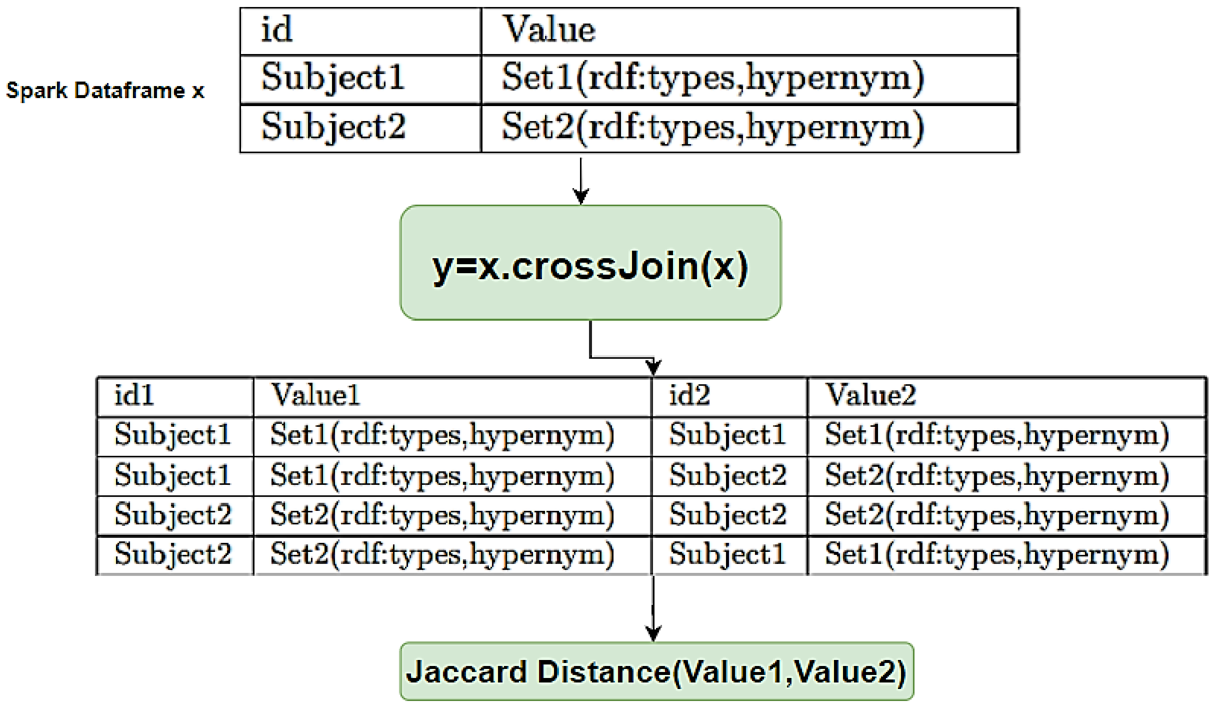
```
val y = x.crossJoin(x)
```

Listing 4.14: crossJoin on DataFrame ”x”

4.7.2 Technique 2 implementation obstacles

The reason for taking the properties into account in technique 2 is to take advantage of hash partition. The hash partitioning partitions the similar key in the same partition and lead to less shuffling while computing cartesian product.

[5]https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html



id	Value
Subject1	Set1(rdf:types,hypernym)
Subject2	Set2(rdf:types,hypernym)

id1	Value1	id2	Value2
Subject1	Set1(rdf:types,hypernym)	Subject1	Set1(rdf:types,hypernym)
Subject1	Set1(rdf:types,hypernym)	Subject2	Set2(rdf:types,hypernym)
Subject2	Set2(rdf:types,hypernym)	Subject2	Set2(rdf:types,hypernym)
Subject2	Set2(rdf:types,hypernym)	Subject1	Set1(rdf:types,hypernym)

Figure 4.5: Spark DataFrame crossJoin method to find similarity between subjects

```
(Property1,(Subject1,Set1(rdf:types,hypernym)))
(Property2,(Subject2,Set2(rdf:types,hypernym)))
```

Listing 4.15: A RDD x with a property as key and value as a tuple of subjects and set of rdf: type/hypernym

The RDD "x" in listing 4.15 consists of pair RDD with a property as key and value as a tuple of subjects and set of rdf:type/LHD. While partitioning the RDD "x", all the similar properties or keys are in the same partition and lead to less shuffling if we apply any action or transformation on it. Let us suppose listing 4.16 to be an RDD "y" after applying the cartesian product.

```
((Property1,(Subject1,Set1(rdf:types,hypernym))
(Proprty2,(Subject2,Set2(rdf:types,hypernym))))
```

Listing 4.16: Cartesian Product of RDD 4.15 (showing only one element of RDD y)

However, this technique has created another issue called data skew. Data skew-which means that few tasks will take too much time (up to 99% of the time) for completing whereas the vast majority of the tasks completed rapidly. In our case, the similar properties are in the same partition by hash partition technique. So, the properties that are more in numbers in some partitions will take more time if we apply transformation and action operations on them (as they are in one partition and lead to data skew). We have applied a hash partition method on the RDD before calculating cartesian product as shown in listing 4.17. In the next step, the cartesian method is applied to hash-partitioned RDD. The technique2 has suffered from the data skew problem.

```
val hashPartition=x.partitionedby(new Hashpartitioner(numofPartition))
val y=hashPartition.cartesian(hashPartition)
```

Listing 4.17: Cartesian Product of RDD "x" with HashPartition

In conclusion, both the techniques suffer from different issues. The technique one is implemented with three methods and is computationally expensive for large datasets. Also, we can not hash partition the RDD in Technique 1 as listed in listing 4.10, it leads to more shuffling (because this RDD has a unique key, i.e., subject). The computational issue of Technique 1 gives us insight into partitioning the RDD based on rdf properties as there are a considerable amount of similar keys. However, the Technique 2 suffers from data skew issue and is not considered for finding the outliers.

4.7.3 Solution by Using Locality Sensitive Hashing(LSH) and approxSimilarityJoin method

The above implementations for cohorting classes are unable to create input for Jaccard distance due to the sheer size of RDDs and data skew after cartesian product. The N^2 comparison in RDD “y” are ultimately time-consuming, volume-intensive, and hardware-reliant for large data sets.

Locality Sensitive Hashing (LSH)[6] is an important class of hashing techniques. It is commonly used in clustering, approximate nearest neighbor search and outlier detection with large datasets. LSH uses a family of functions (“LSH families”) to hash data points into buckets so that the data points which are close to each other are in the same buckets with high probability. Formally, An LSH family is defined as follows:-

In a metric space (m, dist), where m is a set and dist is a distance function on m, an LSH family is a family of functions h that satisfy the following properties:

$$\forall p, q \in m, \tag{4.1}$$

$$dist(p, q) \leq r1 \Rightarrow Pr(h(p) = h(q)) \geq p1 \tag{4.2}$$

$$dist(p, q) \geq r2 \Rightarrow Pr(h(p) = h(q)) \leq p2 \tag{4.3}$$

This LSH family is called (r1, r2, p1, p2)-sensitive.

The different LSH families are implemented in separate classes (e.g., MinHash), and APIs for feature transformation, approximate similarity join, and approximate nearest neighbor are provided in each class. In LSH, we need to define a false positive as a pair of distant input features (with $dist(p, q) \geq r2$) which are hashed into the same bucket, and also a false negative as a pair of nearby features (with $dist(p, q) \leq r1$) which are hashed into different buckets.

There are two LSH functions provided by spark namely- BucketedRandomProjectionLSH and minHashLSH [7] function. The Bucketed Random Projection is an LSH family for Euclidean distance whereas minHashLSH is an LSH family for Jaccard distance. Since our algorithm is based on Jaccard distance, we have used minHashLSH. The input for the minHashLSH function is binary vectors. It leads to the creation of the binary vectors[8] from the set of data (RDD “x”). Spark has two inbuilt functions for converting text into binary vectors- HashingTF and CountVectorizer[9]. The table 4.3 compares both the techniques. The results for both the

[6]`https://spark.apache.org/docs/2.2.0/ml-features.html#locality-sensitive-hashing`
[7]`https://spark.apache.org/docs/2.2.0/ml-features.html#minhash-for-jaccard-distance`
[8]`https://en.wikipedia.org/wiki/Feature_(machine_learning)`
[9]`https://spark.apache.org/docs/2.2.0/ml-features.html#tf-idf`

techniques are compared in the next chapter.

CountVectorizer	HashingTF
Scans data twice-one for building model and another for transformation	Scans the data only once.
It needs extra space equal to number of unique features	Does not require any additional storage
It works well when we want to convert back the numerical feature vector to text	No back conversion is possible

Table 4.3: Comparison between CountVectorizer and HashingTF

The minHashLSH function has a parameter named as setnumHashTables that are adjusted by the user. On increasing the number of hash table, the accuracy for cohorting the subjects will increase, but it also increases the communication cost and runtime. We have tried different values for the number of hash tables range from three to ten and our algorithm has worked well when it equals three.

In the next step, we have to fit the binary vector data through minHashLSH and create a model for it as shown in the listing 4.18. This model has a function named as Approximate Similarity Join for calculating the similarity between the subjects. It also supports self-joining of the same dataset. It takes two datasets as input and results in approximate pairs of rows in the datasets by using Jaccard distance and filter those results whose distance is smaller than a user-defined threshold value.

```
  val hashingTF = new HashingTF().setInputCol("values").
                    setOutputCol("features").setNumFeatures(numofFeature)

 val featurizedData = hashingTF.transform(x) //x is input DataFrame

 val mh = new MinHashLSH()
               .setNumHashTables(3)
               .setInputCol("features")
               .setOutputCol("hashes")

 val model = mh.fit(featurizedData)
 val  dffilter = model.approxSimilarityJoin(featurizedData, featurizedData,0.45)
 // dffilter  output two similar subjects whose Jaccard distance is  less  than
 //the threshold value i.e. 0.45
```

Listing 4.18: Creating cohorts of classes by using minHashLSH

The threshold value is adjustable by the user and ranges from zero to one. This section has solved the problem for cohorting of classes which is the main challenge of our algorithm.

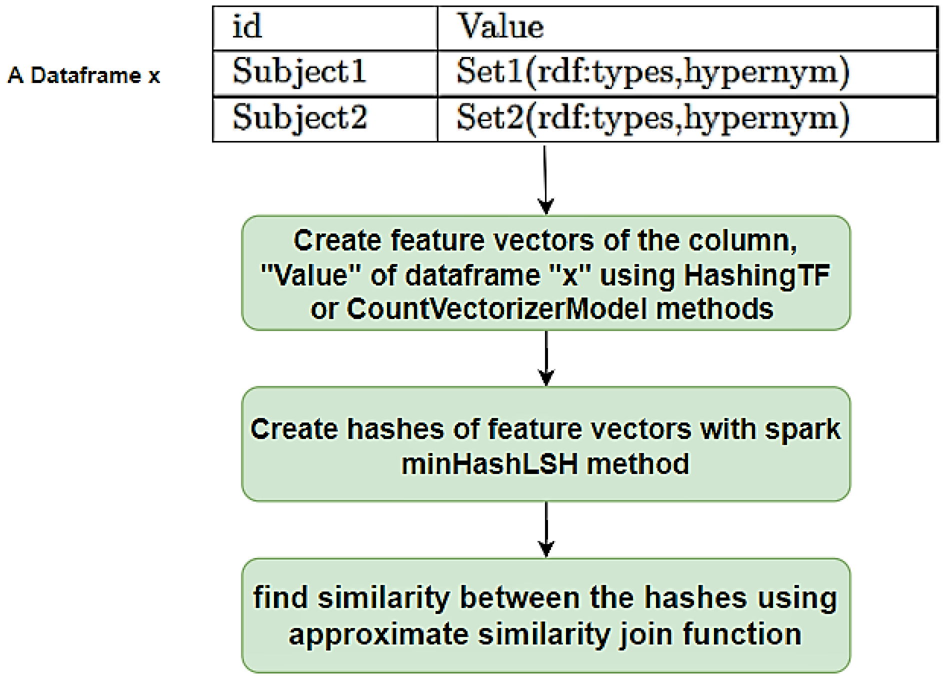

A Dataframe x
id
Value
Subject1
Set1(rdf:types,hypernym)
Subject2
Set2(rdf:types,hypernym)
Create feature vectors of the column, "Value" of dataframe "x" using HashingTF or CountVectorizerModel methods
Create hashes of feature vectors with spark minHashLSH method
find similarity between the hashes using approximate similarity join function

Figure 4.6: Spark minHashLSH method to create cohorts of similar classes

The figure 4.6 is the flow diagram that depicts how Spark minHashLSH method is creating input for Jaccard distance for making cohorts of classes. In the next section, we have explained our opted design to find the numerical outliers from the RDF dataset.

4.8 Our Design

The most of the computation in the algorithm has occurred while creating cohorts of classes and succeeded using the minHashLSH method. So, we have opted the Technique 1 (as shown in the figure 4.1) with three different methods as a valid approach to find the numeric outliers. The Technique 2 is ruled out due to data skew problem.

4.8.1 Storage Unit

Every system requires a storage system to save the input files and output file (if any). We have used HDFS to store the RDF file and Spark RDD to store the intermediate results. Since spark RDD does only in memory computation, it makes our process faster.

4.8.2 Numerical Outlier Model

The numerical outlier model with Technique 1 takes input as RDF N-Triples file containing numerical literals, subject with rdf: type and LHD, and it yields outliers from the dataset. The outliers in the output may be natural or real outliers. In the following section, we have explained the different parts of our design as described in the figure 4.1.

Input file

The first step is to feed the input in the form of RDF N-triples format. There is a need to prepare a dataset from DBpedia. The preparation of the dataset is a crucial and challenging task as the aggregation of all the three requirements mentioned in section 4.3 is hard to get in one dataset. For example, from DBpedia, we have merged various files to get the required dataset. Spark framework is used to partition the data into multiple parts so that data can be processed in a distributed way.

Creation of Cohorts

The next step is to process the data and create the cohort of classes out of it. The numerical outlier model filters the triples which only have literals of the type xsd:integer, xsd:double or xsd:nonNegativeInteger. This step assures the availability of the numerical literals in the dataset. The following step is to filter the rdf:type and LHD of the subject of numerical literals by considering only DBpedia type. These two steps build appropriate data for our experiment. The filtered data is input either in cartesian product/crossJoin or approxSimilarityJoin function for creating a pair of similar subjects. Since these functions return a pair of similar subjects, we have performed spark groupby function to produce a large cohort of subjects of similar class. The figure 4.1 depicting subjects are grouped into either country, city, and continent, etc.

Grouping Properties on each cohort

Every cohort contains many triples of the same type with different properties. For example, cluster country contains subjects like India, Germany, Russia with properties populationtotal, longitude, latitude, Totalarea and many more. The next task is to group the properties in each cohort (as properties longitude and totalArea are incomparable in the Country cohort). The filtered input is passed to the outlier detection algorithm.

Outlier detection

The IQR method has employed to find the outliers from the list of data that are grouped according to the properties. IQR results in the two numeric values "x" and "y" from the data as explained in section 2.6.2 in chapter 2. Any value less than "x" and greater than "y" from the list of data is considered an outlier. These outliers may be natural or not. For example, the population of India and China from a cohort named Country is a natural outlier.

5 Evaluation

We have designed and developed a numerical outlier detection prototype as illustrated in section 4.8. In this chapter, we will compare the different methods of implementation, ways of classifying the outliers and determining the cause of the existence of outliers in the output. These metrics are presented as:-

1. Comparing execution time of different approaches with varying dataset size.
2. Different ways of classification of outliers.
3. Error Analysis- Analyzing the reason for the existence of the outliers in the result

With this evaluation process, we can justify the research question regarding the detection of outliers in a massive dataset in a scalable manner. We have tested our implementation on Spark cluster[1] and its configuration details are explained in the next section.

5.1 Cluster configuration

To test out the performance of the outlier detection model that is based on Spark and Scala, we have deployed it on Spark cluster. We have used a Spark Standalone mode with Spark version 2.2.1 and Scala with version 2.11.11. A small private cluster of Smart Data Analytics (SDA) consists of 4 servers is used. These servers have a total of 256 cores, and each server has Xeon Intel CPUs at 2.3GHz, 256GB of RAM and 400GB of disks space, running Ubuntu 16.04.3 LTS (Xenial) and connected in a Gigabit Ethernet2[2] network. Each Spark executor is assigned a memory of 250GB.

5.2 Datasets

There are different sources of collecting the datasets like Yago and Wikidata, but we have considered DBpedia datasets in our thesis. The three different sized datasets

[1]`https://en.wikipedia.org/wiki/Computer_cluster`
[2]`https://en.wikipedia.org/wiki/Gigabit_Ethernet`

are prepared from DBpedia to explore the scalability of the code. These datasets comprise numerical literals, rdf:type property of subjects, and LHD. These three parameters meet the requirements that are required to detect the outliers from the dataset. The datasets are prepared from the different version of DBpedia, e.g., the latest version[3] of DBpedia does not contain the LHD knowledge.

DBpedia Datasets		
Dataset Categorization	Dataset Size	# of Triples
small-DBPedia	110MB	512190
medium-DBPedia	3.7GB	24670594
DBPedia	16.6GB	117544372

Table 5.1: Description of DBpedia Dataset

The table 5.1 consists of three columns- dataset categorization, file size and the number of RDF N-triples. We have categorized the dataset into three sections- small-DBPedia (size in MB), medium-DBPedia (size between 3GB to 10GB) and DBPedia (size between 11-20GB). This categorization is simply for our convenience so that we can distinguish the datasets easily. The third column gives information about the total number of triples in each file.

5.3 Configuration Parameters

The configuration parameters are the essential parameters for the algorithm. The adjustment of the configuration parameters helps to get the desired results. We have adjusted the two configuration parameters- Jaccard similarity threshold value and the list size of a cohort of class with a specific property for the calculation of IQR. These two parameters are explained in detail as follows:-

1. **Jaccard Similarity Threshold**:- As discussed in chapter 4, the Jaccard Similarity threshold value is assigned between zero to one. The Jaccard distance outputs a real number, with a range from zero to one. If this value is less than the given threshold value, the subjects of RDF triples are considered to be similar, and cohorts of classes are created. In our task, we have assigned a threshold value of 0.45 in two functions- approximate similarity join function as well as cross join function.
2. **Low number of instances**:- If the list size of a cohort of class with a specific property is small, IQR operation is not performed on that list. Because a small

[3] `http://wiki.dbpedia.org/downloads-2016-10`

size list of numbers does not result in a good outlier value. For example, D is a list of numbers and we want to find the outliers from the given list.

$$D = \{1, 50, 70, 500000\} \tag{5.1}$$

$$Q1 = 25.5, \quad Q3 = 250035 \tag{5.2}$$

$$IQR = Q3 - Q1 = 250035 - 25.5 = 250009.5 \tag{5.3}$$

$$x = Q1 - 1.5 * IQR = 25.5 - 1.5 * 250009.5 = -374988.75 \tag{5.4}$$

$$y = Q3 + 1.5 * IQR = 250035 + 1.5 * 250009.5 = 625049.25 \tag{5.5}$$

If any value from the list D is smaller than x and higher than y, it is considered as an outlier. However, none of the value in the list is either smaller than x or greater than y, leads to the conclusion that there is no outlier in the list. However, if we analyze the list D, we can see that the value 500000 is considerably larger than the rest of the values of the list and should be an outlier. With this example, we have observed that small sized list does not output the outlier although it consists of an outlier. There is no clear boundary that how much size the dataset should have. However, by examining the data and cohort size, we can define the size of the list for IQR. In our approach, we have assigned the value for the size of the list equals 10.

5.4 Evaluation Results

The table 5.2 describes some interesting properties of the DBPedia i.e, 16.6GB dataset. 22375991 numeric literals were selected for anomaly detection after the filtering process. These triples belong to 1567 distinct properties. From these properties, only 408 were found to have outliers and the total number of outliers found in DBPedia are 24015 that is approximately 0.1 % of the total numeric literals present in the data. We present our results based on different techniques discussed in the previous chapter. The results from Spark client mode is taken into account, and the metrics like runtime of our algorithm, classification of outliers and outliers analysis is the prime focus of this section.

5.4.1 Results based on Spark client mode

The execution time is a crucial parameter to monitor the performance and scalability of an algorithm. This section deals with the comparison of the execution time of outlier detection algorithm. The comparison is evaluated by varying dataset size

Statistics for DBPedia large file(16.6GB)	
Statistics	Value
Distinct Properties	2863
Triples (including duplicates)	117544372
Numeric literals after filtering process(including duplicates)	22375991
Filtered distinct numerical properties	1567
The number of properties with outliers	408
Total number of outliers	24015
– Runtime TV series has the total number of outliers	482
– Built-year property of buildings has the total number of outliers	86
– PostalCode of area has the total number of outliers	176

Table 5.2: Statistics for DBPedia file(16.6GB)

as well as by employing the different techniques of implementation. The dataset is categorized into small-DBPedia, medium-DBPedia and DBPedia as described in table 5.1. We have excluded the time taken by the reader class, i.e., NtripleReader[4] class for reading the data. All the experiments are performed on a Spark client mode with the configuration described in table 5.3.

Parameter Name	Value
Driver Memory(GB)	50
Executor Memory(GB)	200
Total number of cores	192

Table 5.3: Spark configuration parameters in cluster mode

In figure 5.1, the runtime for three different datasets with three different methods are compared. The execution time for the small-DBPedia dataset is approximately equivalent for all the methods because the small dataset does not require a high number of resources like memory and cores. For the medium-DBPedia, the DataFrame crossJoin function takes less time compared to other approaches. The other two approaches take more time due to approximate similarity join function that calculates the approximate similarity for finding the similar items in our algorithm and also the process of computing the vector from strings and creation of vocabulary for the vectorizing is noticeable for the medium sized data. Due to this reason, the execution time of HahingTF and CountVectorizerModel model is more as compared to crossJoin on medium datasets. For the large size i.e., DBPedia,

[4]`https://github.com/SANSA-Stack/SANSA-RDF/blob/1916bb1b52d0e8e4f03243540e4079a5980fa50e/sansa-rdf-spark/src/main/scala/net/sansa_stack/rdf/spark/io/NTripleReader.scala`

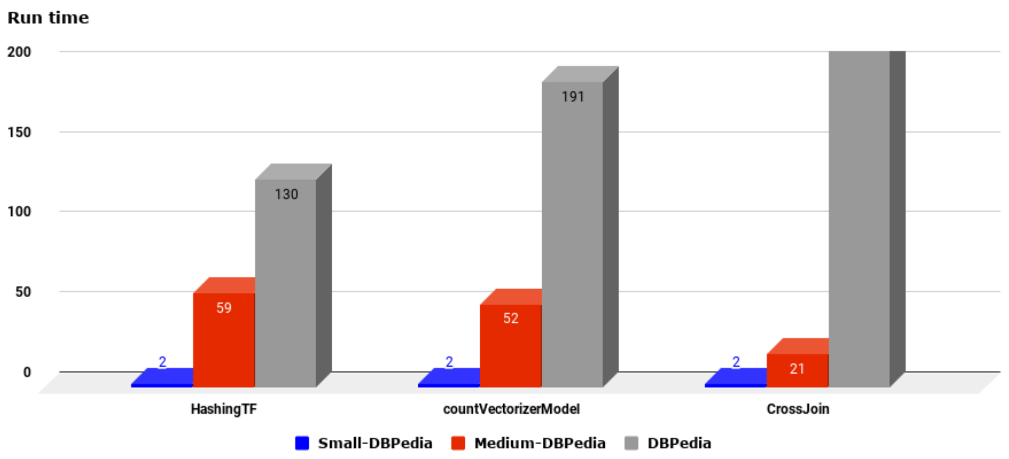


Figure 5.1: Execution time for three different approaches for three different datasets

the minHashLSH with the HahingTF approach for cohorting the subjects performs better than other methods. The DataFrame crossJoin approach fails on the large dataset due to its $O(n^2)$ complexity. As the data size grows, intermediate data after crossJoin increases exponentially and it is difficult to handle such a huge amount of data hence leads to failure. The countvertoizerModel approach takes more time than the HashinTF on the large dataset as it scans the data twice described in table 4.3. By examining the results shown in figure 5.1, it can be inferred that minHashLSH with HashingTF approach is better than all other approaches for big datasets. The table 5.2 shows the statistics about outliers in DBPedia i.e., 16.6GB. The scalability of the opted approach i.e, minHashLSH with HashingTF has shown in the figure 5.2.

5.5 Analyzing the Outliers

The outliers that are generated from the DBPedia, i.e., 16.6GB file are numerous, and the next task is to label them as natural or real outliers. We have adopted two techniques for the classification of outliers. The detail explanation of these techniques are described as follows:-

5.5.1 Manual inspection of outliers

The first method for classifying the outliers into natural or real outliers is by employing human annotators. We have assigned the output file (file consists of outliers)

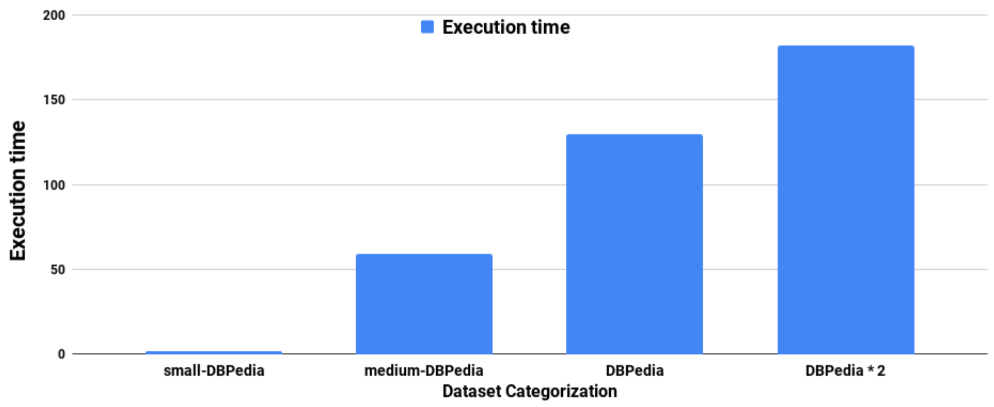


Figure 5.2: Execution time of different datasets for minHashLSH with HashinTF

to a group of people for labeling the outliers. The annotators match the outliers values to the corresponding Wikipedia pages and label the outliers accordingly. If the outlier value matches with the Wikipedia page, it is considered as a natural outlier, otherwise a real outlier. The reason for the occurrence of outliers may be because of the issues in DBpedia extraction tool or vandalism[5] in Wikipedia pages.

While going through the list of outliers present in the output file, some outliers are recognized easily. For example, the properties like date, builddate usually have a range from 0001 to 9999 at the literal position. However, we have found some unusual patterns in the output as shown in the listing 5.1 and classified them as real outliers.

```
dbr:GS&WR_Class_201  dbo:builddate   188718951901^^xsd:int
dbr:GE_UM12C         dbo:builddate    19631966^^xsd:int
```

Listing 5.1: Example of real outliers in DBpedia

Since the outliers are numerous, the manual inspection of outliers for classification makes it more challenging.

5.5.2 Visualization of outliers through scatter plot graph

The second method for classifying the outliers is through a visualization tool. There are online and offline tools for generating the graph. We are using python matplotlib[6] library for creating the outlier graph that gives us an outline for classification

[5]https://en.wikipedia.org/wiki/Vandalism_on_Wikipedia
[6]https://matplotlib.org/

of outliers as natural or real outliers. We have randomly selected few properties from the list of outliers and created a CSV file of each property containing numerical values with the help of Spark and Scala. The python script is used for creating a scatter plot graph. The distinction between outliers and normal values is accomplished through color coding. The red color corresponds to the outliers whereas green color represents the normal data value. The x-axis of the scatter plot graph consists of data values whereas the y-axis shows the count or frequency corresponding to the data value.

Runtime property of TV series

The runtime properties of TV series describes the duration of TV series in seconds. For example, the listing 5.2 depicts that the TV series "Dying Alive" has runtime 6888.0 seconds.

```
dbr:Dying_Alive          dbo:runtime   6888.0^^xsd:double
dbr:Undersea_Kingdom     dbo:runtime   13560.0^^xsd:double
```

Listing 5.2: runtime property of TV series in DBpepdia

We have calculated IQR for the detection of outliers on the numeric value of runtime property and plot it with the help of python script. While analyzing the graph 5.3, the runtime values between 15000 and 20000 seconds are considered as a natural outlier because any TV series may have runtime around 48 hours. For example, the TV series named as "Deng Xiaoping at History's Crossroads" has runtime around 48 hours. When cross verified with Wikipedia, this value matches with the Wikipedia value and acts as a natural outlier. The reason of occurrence of this TV series as an outlier because of the IQR maximum range i.e. 8874 seconds. The lower and maximum range of the runtime property gives the idea that most of the TV series has runtime in this range only. Any value greater than the upper range of IQR is considered as an outlier.

It is difficult to analyze the outliers present near zero in the graph 5.3, so we have zoomed in the scatter plot graph to get a more clear picture of the outliers. In graph 5.3, it can be seen that there are many TV series which have the negative runtime. Since time is a positive quantity, so any value below zero can be considered as real outliers. For example, the TV series "The X Factor Digital Experience" has runtime -9000 seconds in DBPedia and is a real outlier.

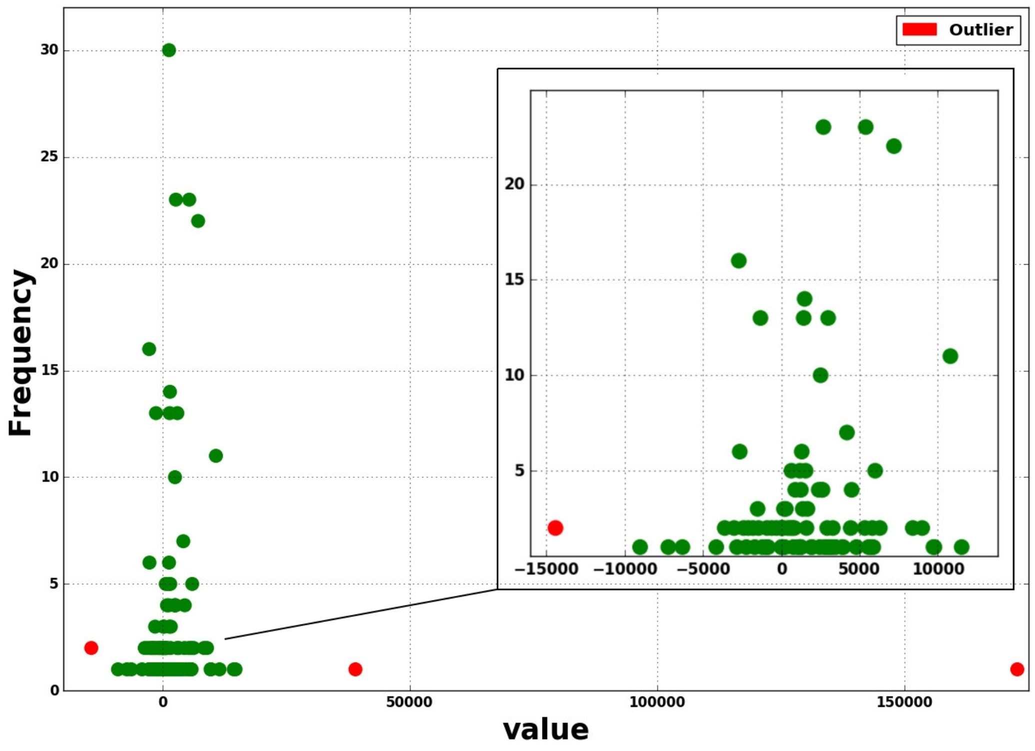


Figure 5.3: Scatter plot graph showing outliers in runtime (in seconds) property of TV series

Built property of buildings

Built property depicts that when some building is built and it has value in years. The listing 5.3 shows that Foolad Shahr Stadium was built in 1988.

```
dbr:Foolad_Shahr_Stadium    dbo:built    1998^^xsd:int
  dbr:Ivaylo_Stadium        dbo:built    1958^^xsd:int
```

Listing 5.3: Built property in DBepdia

The figure 5.4 shows the scatter plot graph of built property. The x-axis represents the value of built property in years whereas the y-axis represents the frequency corresponding to each year. Since built property signifies the past events, any value greater than the current year is considered as a real outlier. But the value in the x-axis shows that years have value around 0.9 x $1e^{19}$ and these values act as real outliers. For example, Metropolitan Life Insurance Company Hall of Records building was built in the year 9223372036854775807 as per outliers list and it is impossible to build something in the future. There are many other values lies near zero in the scatter plot graph and we have to explore these values to get a better insight about outliers.

The graph 5.4 also shows zoom-in view of built property. The lower range of IQR equals 1026.0 wheres upper ranges equals 2618.0. Any value below and above these threshold values acts as an outlier. Since the years can never have negative values so the value below zero in the graph is a real outlier. There is a huge concentration of outliers near -2000 in the graph and are considered as real outliers.

Width property of cars

The width property of cars has range either in xsd:double or xsd:integer. The listing 5.4 shows the automobile named as Toyota Avalon[7] has width 1997 in DBpedia.

```
dbr:Jaguar_S−Type     dbo:width    "2007"^^xsd:integer
dbr:Toyota_Avalon     dbo:width    "1997"^^xsd:integer
```

Listing 5.4: RDF data of width property of cars present in DBepdia

We have plotted the outlier detection graph for the width property of automobiles as shown in the figure 5.5. The value around 2000 in the graph act as real outliers, because DBpedia extraction tool has extracted years instead of the actual value.

[7] https://en.wikipedia.org/wiki/Toyota_Avalon

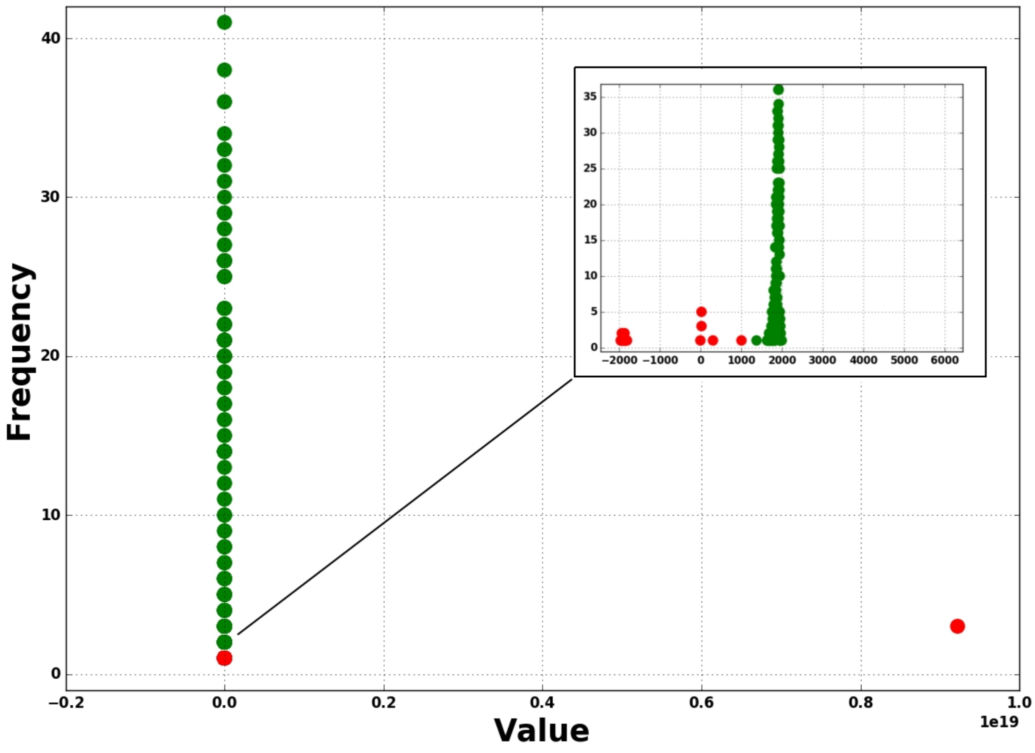


Figure 5.4: Scatter plot graph showing outliers in built property (in years) of buildings

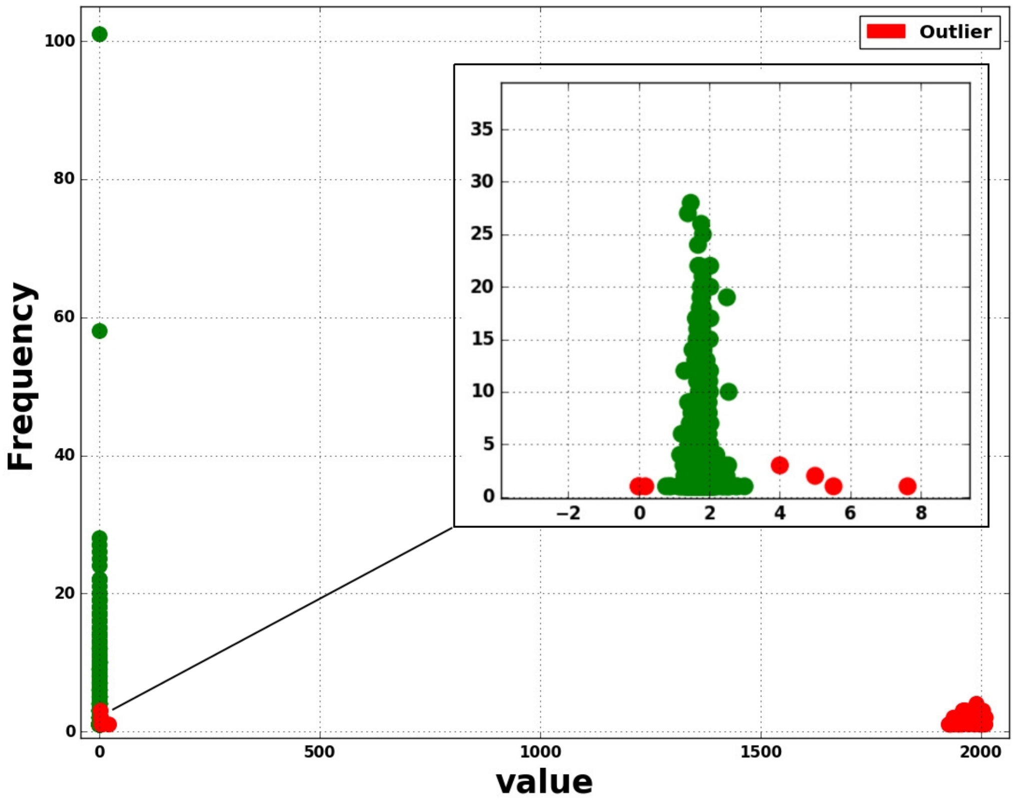


Figure 5.5: Scatter plot graph showing outliers in width property of cars

The unit of measurement plays an essential role in any physical quantity. The comparison is performed in a similar unit of measurement. However, in DBpedia, there is an intermixing of incompatible units of measurement that leads to inaccurate data. The figure 5.5 shows that the Oldsmobile 88 cars has width 1992. When it has cross verified with Wikipedia pages, it is found that DBpedia extracted years of manufacturing instead of actual width value. Since the outliers near zero are hard to visualize, the graph 5.5 also shows the zoom in view of the scatter plot graph. It shows that most of the cars have a width range from 0.627 to 3.011. Any value out of this range act as outliers. The validity of outliers needs to be cross verified with corresponding Wikipedia pages of cars.

5.6 Error Analysis

We need to analyze the source of occurrence of the outliers in the RDF data which caused them to appear in the final results. There may be the following reasons for the occurrence of the outliers in the result:

1. Due to incorrect information in Wikipedia.
2. Due to DBpedia extraction tool

Incorrect information in Wikipedia

“The reliability of Wikipedia (predominantly of the English-language edition) has been frequently questioned and often assessed [8]”. Wikipedia pages are maintained manually, and the input is neither restricted nor validated. This human intervention for managing Wikipedia become the source of the incorrect information. The incorrect information present at Wikipedia results to outliers in the RDF data. The correctness of these outliers is difficult to assess as we have to confirm with various valid resources. It also requires a wide range of domain knowledge for a particular subject of interest.

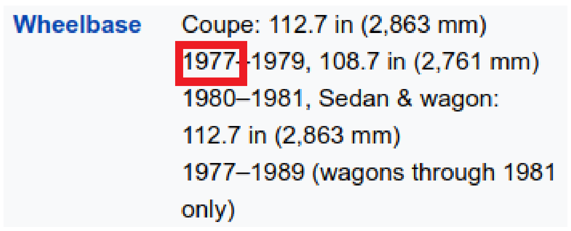
DBpedia extraction tool

The infobox section of Wikipedia presents a summary of some unifying aspect that the articles share and sometimes to improve navigation to other interrelated articles. It makes easier for the user to find important information about that page quickly as well as compare it with that of other articles. The infobox information present in the form of name and value, for example, if a Wikipedia page is related to India, the infobox will consist of the population as name and number as a value. The DBpedia extraction tool may extract the incorrect value from the infobox and lead to outliers in the final result. The extraction tool of DBpedia either append some extra number in the data or remove some part of it. We have collected various examples where we can analyze the shortcomings of the DBpedia extraction tool.

1. **Extracting value present at first place from Wikipedia Infobox**

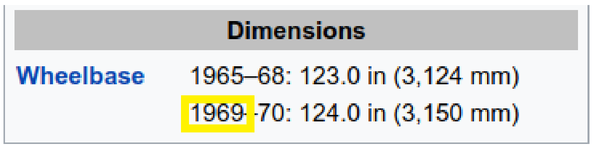
   While going through outliers, we have observed that the DBpedia extraction tool is extracting the numeric value present at first place from the infobox. For example, DBpedia have extracted the information from Wikipedia about two cars, namely Dodge Diplomat[9] and Oldsmobile 88[10] as follows:

[8] `https://en.wikipedia.org/wiki/Reliability_of_Wikipedia`
[9] `https://en.wikipedia.org/wiki/Dodge_Diplomat`
[10] `https://en.wikipedia.org/wiki/Oldsmobile_88`



Wheelbase	Coupe: 112.7 in (2,863 mm) 1977–1979, 108.7 in (2,761 mm) 1980–1981, Sedan & wagon: 112.7 in (2,863 mm) 1977–1989 (wagons through 1981 only)

Figure 5.6: Infobox of Dodge Diplomat car [9]



Dimensions	
Wheelbase	1965–68: 123.0 in (3,124 mm) 1969–70: 124.0 in (3,150 mm)

Figure 5.7: Infobox of Oldsmobile 88 car[10]

```
dbr:Dodge_Diplomat    dbo:wheelbase    1977^^xsd:int
dbr:Oldsmobile_88     dbo:wheelbase     1969^^xsd:int
```

Listing 5.5: Wheelbase property of cars

The listing 5.5 is present in the 16.6GB dataset and output as outliers in our final results. When we compare these values to their respective Wikipedia pages, we have analyzed that DBpedia extracted the first value from the infobox irrespective of looking at its actual value. For example, in figure 5.7, DBpedia has extracted 1969 present at starting position and declining the actual value 124 inches. In another example of Dodge Diplomat car, same observation is noticed, i.e., extracting the year 1977 instead of actual value 112.7 inches.

2. **Concatenation of numbers**

   Sometimes DBpedia extraction tool removes the hyphens, comma or any other separator between the values and merges them into one number. The listing 5.6 is present in our DBpedia dataset and acts as outliers in the final output. The property "dates" is indicating about the happening of a particular event, and it can be observed that this value is out of range (the value of year has values in the range between 0001 and 9999). The second property in the listing 5.6 is a postal code, and maximum length of the postal code in worldwide is ten digits [11].

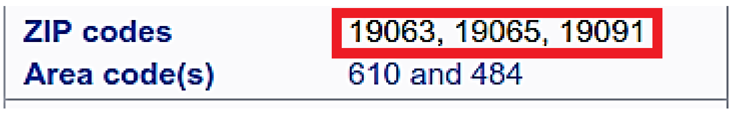
[11]https://en.wikipedia.org/wiki/Postal_code

```
dbr:517th_Air_Defense_Group    dbr:dates 19451953^^xsd:int
dbr:Media,_Pennsylvania        dbr:postalCode 190631906519091^^xsd:int
dbr:General_Motors_Building_(Manhattan) dbr:location 7675^^xsd:int
```

Listing 5.6: date, postal code and location as a property example

However, in the listing 5.6, the postalcode value is more than ten digits and acts as a real outlier. Next, the location property of General motors building has value equals 7675, and it outputs as an outlier in our result. We can not predict by looking at its value because this value seems to be normal. We have to cross verify by looking at its Wikipedia page and then classify it as a real or natural outlier.

We can clearly examine from the figures 5.8 and 5.9 that DBpedia extraction tool removes the separator "comma" between the values and merge them into one big number. For example, the zip codes of Media Pennsylvania[12] are 19063, 19065, 19091 as shown in infobox in the figure 5.8 and merged to 190631906519091. Also, the date property with value 1945-1953 of 517th Air Defense Group[13] is depicted in the figure 5.9 are concatenated to 19451953. In figure 5.10, DBpedia has concatenated location of General Motors building[14], i.e., "767" with other value "5th" and result in 7675.



ZIP codes	19063, 19065, 19091
Area code(s)	610 and 484

Figure 5.8: Infobox of Media, Pennsylvania[12]

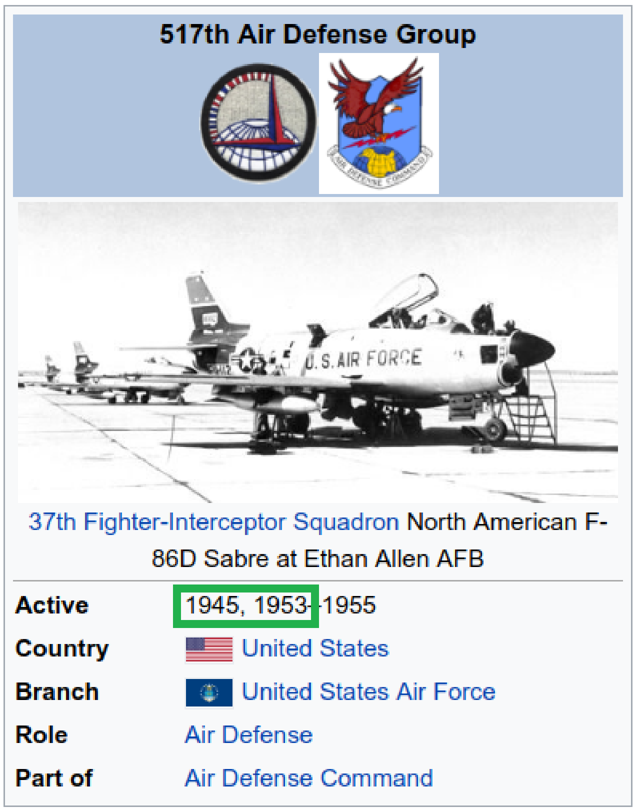
3. **Problems Interpreting and Converting Units**

   There are some properties in the DBpedia which use a different unit of measurement. For example, the property runtime has value either in seconds, minutes or hours.

```
dbr:Arthur_(TV_series) dbr:runtime −1560^^xsd:int
dbr:Aqua_Teen_Hunger_Force dbr:runtime −720^^xsd:int
```

Listing 5.7: dates and postal code property example

[12]https://en.wikipedia.org/wiki/Media,_Pennsylvania
[13]https://en.wikipedia.org/wiki/517th_Air_Defense_Group
[14]https://en.wikipedia.org/wiki/General_Motors_Building_(Manhattan)

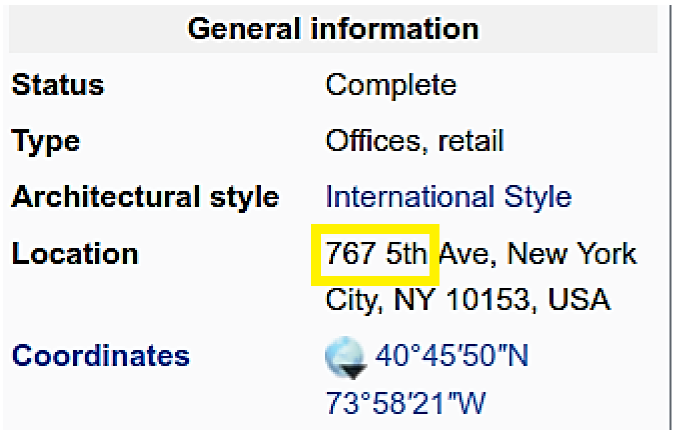


517th Air Defense Group

37th Fighter-Interceptor Squadron North American F-86D Sabre at Ethan Allen AFB

Active	1945, 1953–1955
Country	United States
Branch	United States Air Force
Role	Air Defense
Part of	Air Defense Command

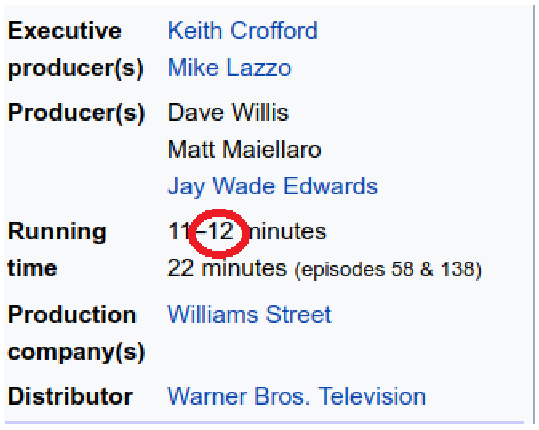
Figure 5.9: Infobox of 517th Air Defense Group[13]



General information

Status	Complete
Type	Offices, retail
Architectural style	International Style
Location	767 5th Ave, New York City, NY 10153, USA
Coordinates	40°45′50″N 73°58′21″W

Figure 5.10: Infobox about the location of General Motors Building[14]



Executive producer(s)	Keith Crofford Mike Lazzo
Producer(s)	Dave Willis Matt Maiellaro Jay Wade Edwards
Running time	11–12 minutes 22 minutes (episodes 58 & 138)
Production company(s)	Williams Street
Distributor	Warner Bros. Television

Figure 5.11: Infobox of TV show named as Aqua Teen Hunger Force [15]

DBpedia converts these values in seconds and stores in its database. However, while converting these value, it does some incorrect calculation and leads to outliers. For example, the runtime for two TV shows is shown in listing 5.7 as negative in our 16.6GB dataset.

If we look at the infobox of these two TV shows as given in the figures 5.11 and 5.12, the runtime for both the shows is in the range of minutes, i.e., 11-12 and 24-26 minutes. DBpedia extraction tool extracts values for runtime property in minutes i.e., -12 minutes from the infobox of Arthur TV series and multiply it with 60 (-12*60= -720 seconds) and save it to DBpedia data dump. DBpedia has also extracted the hyphen which is a separator between the values and produces the runtime in negative for both the TV shows. The same calculation is computed on infobox of Aqua Teen Hunger force TV show and result to output -1560. The values -720 and -1560 acts as real outliers as time is a positive quantity.

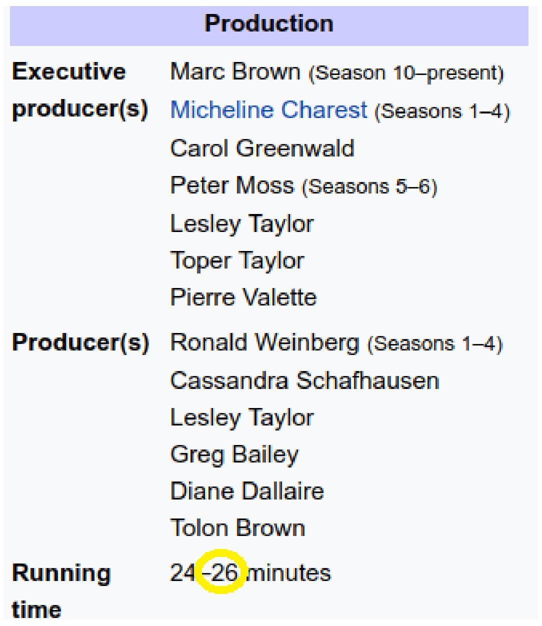
4. **Infobox properties without starting year**

   Some events on Wikipedia has no starting point, and the value for those events has marked as empty. This kind of entry in infobox produces an incorrect value when extracted by DBpedia extraction tool.

```
dbr:Olaf_Berner         dbr:years    −1978^^xsd:int
dbr:Albert_Mitchell     dbr:years    −1921^^xsd:int
```

Listing 5.8: DBpedia dataset with year property

[15] https://en.wikipedia.org/wiki/Aqua_Teen_Hunger_Force
[16] https://en.wikipedia.org/wiki/Arthur_(TV_series)



Production	
Executive producer(s)	Marc Brown (Season 10–present) Micheline Charest (Seasons 1–4) Carol Greenwald Peter Moss (Seasons 5–6) Lesley Taylor Toper Taylor Pierre Valette
Producer(s)	Ronald Weinberg (Seasons 1–4) Cassandra Schafhausen Lesley Taylor Greg Bailey Diane Dallaire Tolon Brown
Running time	24-26 minutes

Figure 5.12: Infobox of Arthur TV series[16]

The listing 5.8 is a snapshot of our 16.6GB DBPedia dataset and results in an outlier in the final result. Since the property years can never have negative value, the listing, 5.8 shows the property "years" have negative value.

The cause of this error is due to the unavailability of starting year of the event in Wikipedia infobox. For example, in figure 5.13 and 5.14, the starting year of the events are missing. Only event ending year is present with value -1921 and -1978. While extracting the value from the infobox, it also extracts the hyphen with the value and dump in DBpedia database.

5. **Removing characters from alphanumeric value**

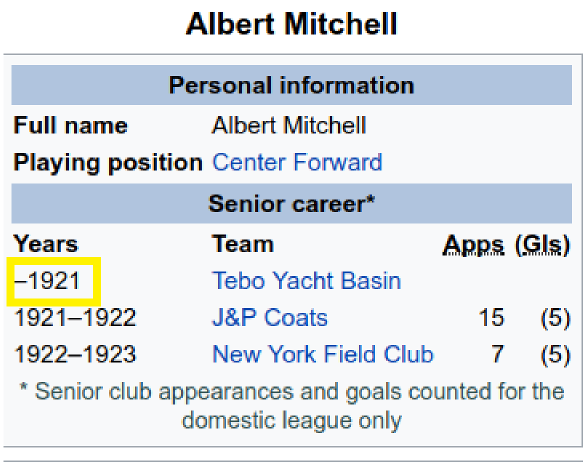
   There are some cases when the DBpedia extraction tool removes the characters that are followed by a numeric value.

```
dbr:Idoxifene                  dbr:unii  456^^xsd:int.
dbr:Octatropine_methylbromide dbr:iupacName −88^^xsd:int.
```

Listing 5.9: Property unii and iupacName with their numeric value

The listing 5.9 are the outliers detected from the DBPedia dataset. However,

[17] https://en.wikipedia.org/wiki/Albert_Mitchell
[18] https://en.wikipedia.org/wiki/Olaf_Berner



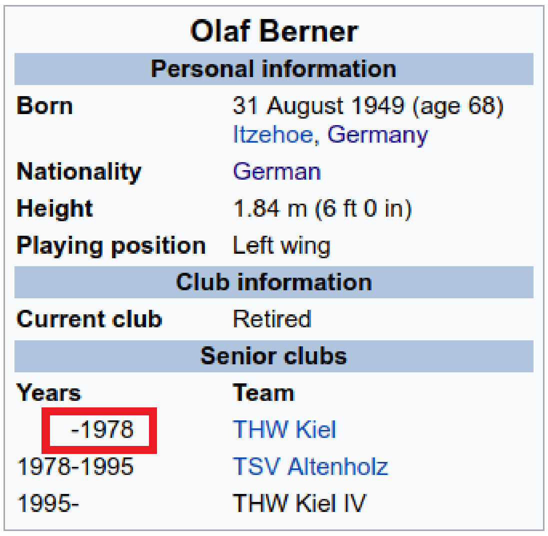
Albert Mitchell

Personal information		
Full name	Albert Mitchell	
Playing position	Center Forward	

Senior career*			
Years	**Team**	**Apps**	**(Gls)**
–1921	Tebo Yacht Basin		
1921–1922	J&P Coats	15	(5)
1922–1923	New York Field Club	7	(5)

* Senior club appearances and goals counted for the domestic league only

Figure 5.13: Infobox about Albert Mitchell[17]



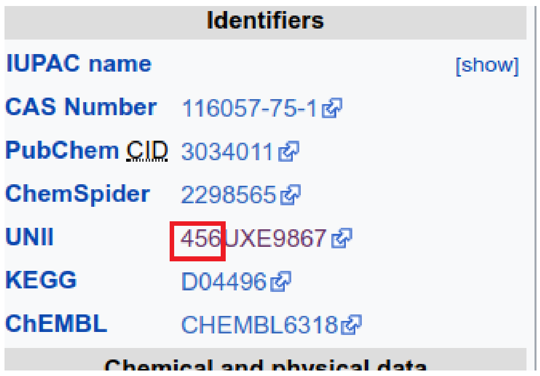
Olaf Berner

Personal information	
Born	31 August 1949 (age 68) Itzehoe, Germany
Nationality	German
Height	1.84 m (6 ft 0 in)
Playing position	Left wing

Club information	
Current club	Retired

Senior clubs	
Years	**Team**
-1978	THW Kiel
1978-1995	TSV Altenholz
1995-	THW Kiel IV

Figure 5.14: Infobox about Olaf Berner[18]

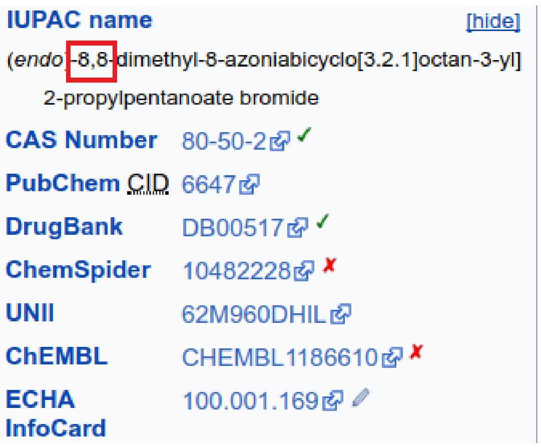


Identifiers	
IUPAC name	[show]
CAS Number	116057-75-1
PubChem CID	3034011
ChemSpider	2298565
UNII	456UXE9867
KEGG	D04496
ChEMBL	CHEMBL6318

Figure 5.15: Infobox about idoxifene[19]

when we analyze their respective Wikipedia pages, it is found that the values of these outliers are different from 16.6GB DBPedia database.

The figure 5.15 depicts that idoxifene with property unii has value 456UXE9867, whereas it is 456 only in the database. Also, the figure 5.16 have IUPAC name is alphanumeric, but DBpedia has extracted it as -88 and stored in the database.



IUPAC name [hide]

(*endo*)-8,8-dimethyl-8-azoniabicyclo[3.2.1]octan-3-yl] 2-propylpentanoate bromide

CAS Number	80-50-2 ✓
PubChem CID	6647
DrugBank	DB00517 ✓
ChemSpider	10482228 ✗
UNII	62M960DHIL
ChEMBL	CHEMBL1186610 ✗
ECHA InfoCard	100.001.169

Figure 5.16: Infobox about Octatropine methylbromide[20]

[19]https://en.wikipedia.org/wiki/Idoxifene
[20]https://en.wikipedia.org/wiki/Octatropine_methylbromide

6 Conclusion

In our work, we have successfully detected the numerical outliers in a scalable manner in Knowledge Graphs, especially DBpedia. We have implemented our approach in three different ways and concluded that minhashLSH with HashingTF outperforms the other techniques while cohorting the classes on large RDF datasets. Our method finds both types of outliers- natural as well as real outliers. We have classified the outliers through manual inspection by human annotators or by plotting them in the scatter plot graph. The numerical outliers detected in the data are useful for improving the quality of RDF Data. The outliers can be corrected in DBpedia source and will be beneficial for other researchers who are specifically working on numerical data.

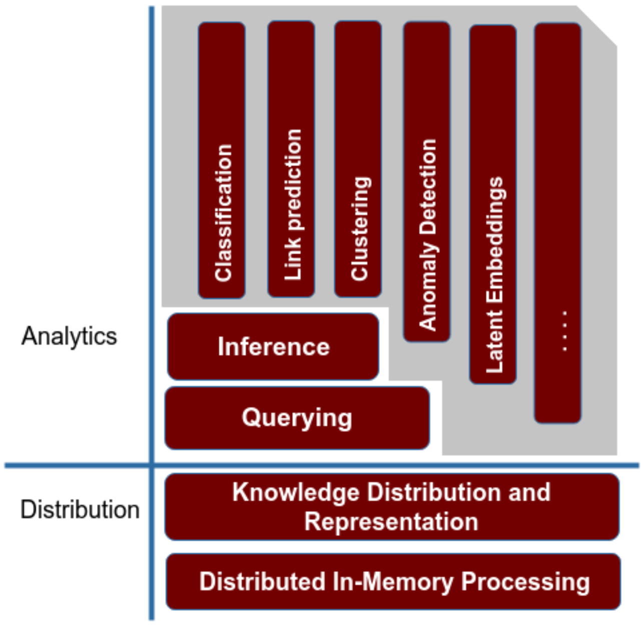


Figure 6.1: SANSA Stack[1]

[1]http://sansa-stack.net/faq/#what-does-SANSA-stand-for

Our thesis work is an addition for Semantic Analytics Stack (SANSA), a distributed computing frameworks (specifically Spark and Flink) with the semantic technology stack[1] as shown in the figure 6.1. A layer named as "Anomaly Detection" in SANSA Stack will help to improve the quality of RDF data by removing the outliers.

In future work, there are possibilities to extend our thesis work. We have only considered numerical literals which are the type of integer, non-negative integer and double. It will be exciting to see to continue this work for another xsd datatypes like xsd:date, xsd:byte, etc. So far, we have only considered one particular data source, i.e., DBpedia. Our thesis work can be stretched to other data source of RDF data like Yago, Wikidata, etc. This research work is limited to numerical outliers only. It can be extended to find the outliers at subject/predicate position by using the schema knowledge. The outlier detection algorithm can be re-implemented in Flink[2] and the comparison of the performance of the algorithm on both the frameworks will give a better insight into our work. As Flink manages many things automatically like data partitioning and caching whereas in Spark, we have to manually set these parameters.

[2]`https://flink.apache.org/`

7 Bibliography

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[2] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000.

[3] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[4] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[5] Daniel Fleischhacker, Heiko Paulheim, Volha Bryl, Johanna Völker, and Christian Bizer. Detecting errors in numerical linked data using cross-checked outlier detection. In *International Semantic Web Conference*, pages 357–372. Springer, 2014.

[6] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[7] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, 2003.

[8] Malte Kiesel and Gunnar Aastrand Grimnes. Dbtropes—a linked data wrapper approach incorporating community feedback. In *Proceedings of EKAW*. Citeseer, 2010.

[9] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Loop: local outlier probabilities. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1649–1652. ACM, 2009.

[10] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[11] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef,

Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[12] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.

[13] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[14] Heiko Paulheim. Identifying wrong links between datasets by multi-dimensional outlier detection. In *WoDOOM*, pages 27–38, 2014.

[15] Heiko Paulheim and Johannes Fümkranz. Unsupervised generation of data mining features from linked open data. In *Proceedings of the 2nd international conference on web intelligence, mining and semantics*, page 31. ACM, 2012.

[16] Yves Raimond, Christopher Sutton, and Mark B Sandler. Automatic interlinking of music datasets on the semantic web. *LDOW*, 369, 2008.

[17] Dominik Wienand and Heiko Paulheim. Detecting incorrect numerical data in dbpedia. In *European Semantic Web Conference*, pages 504–518. Springer, 2014.