Distributed Data Parsing and Vandalism Detection on Large-scale Knowledge Graphs Using Apache Spark and Hadoop Ecosystem

Nayef Fayez Roqaya

Matriculation Number: 2901456
August 30, 2018

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master in Computer Science (Smart Data Analytics)

Supervisors

Prof. Dr. Jens Lehmann
Dr. Hajira Jabeen - Ph.D. student. Gezim Sejdiu

Examiners

Prof. Dr. Jens Lehmann - Dr. Damien Graux

Institut für Informatik III

Rheinische Friedrich-Wilhelms-Universität Bonn

Declaration of Authorship

I, Nayef Fayez Roqaya, declare that this thesis, titled **"Distributed Data Parsing and Vandalism Detection on Large-scale Knowledge Graphs Using Apache Spark and Hadoop Ecosystem"**, and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my own work. I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Acknowledgements

First of all, I would like to thank my three supervisors, **Prof. Dr. Jens Lehmann**, **Dr. Hajira Jabeen** and **PhD student. Gezim Sejdiu** for their guidance throughout this work. Professor Lehmann is a professor of Computer Science at the University of Bonn, leader of the Smart Data Analytic Group and the lead scientist at Fraunhofer Institute. Doctor Jabeen is a Senior Researcher at University of Bonn. Gezim Sejdiu is a PhD student at the University of Bonn.

I would also like to thank **Dr. Damien Graux** a Senior Researcher in Enterprise Information System Group at Fraunhofer Institute for contributing with professor Lehmann evaluate this thesis .

Prof. Dr-Ing. Christian Bauckhage, a professor of Computer Science at the University of Bonn and lead scientist for machine learning at Fraunhofer IAIS, provided me advice during my academic track in addition to his courses at the Bonn-Aachen International Center for Information Technology (B-IT) that contributed in improving my knowledge and skills in the Data Science.

I would like to further express my thanks to **Stefan Heindorf** a researcher at Paderborn University for his time and contribution in explaining ambiguous points related to Wikidata and previous vandalism detection approaches.

My parents Fayez Roqaya and Hind Wahbi, encouraged my natural interests in science, computer systems and mathematics early in my life, as they encouraged and educated me in ways both straightforward and devious. Of course, I would like to thank my wife, Letf Alnirabieh for her patience, as she supported me in so many ways while I worked towards my degree.

Contents

List of Figures

Abstract

Data representation plays a vital role in data processes in context of the data science community. Accordingly, it is important to note that big data is a collection of massive volumes of data which consist of structured data, semi-structured data, or unstructured data. On the one hand, standard Extensible Markup Language (XML) data representation is used for re-presenting a wide range of knowledge bases.

One of the most famous knowledge bases that depends on a standard XML data representation is a Wikidata knowledge base. On the other hand, Resource Description Framework (RDF) data format is commonly used in the data community, including formats such as Turtle, N-Triple, TRIX(RDF Triple in XML) and RDF/XML. For instance, RDF/XML format is used in representing many knowledge bases such as the RDF version of Wikidata. Hence, data parsing and deriving structured data from semi-structured data is a challenge that must be dealt with before data can be processed to achieve any goal.

Wikidata is a new, free, large-scale and open knowledge base created by the Wiki-Media Foundation. Like other knowledge bases, Wikidata's content can be created and edited by anyone, which makes it extremely vulnerable to vandalism . Because of this, Wikidata frequently gets vandalized, which exposes all its users to the risk of spreading vandalized and falsified information.

Keywords: Vandalism, Parser, large-scale, Wiki-Data, Apache Spark, Hadoop-Ecosystem, Apache Jena, RDF4j, Random-forest, Decision-Tree, Receiver operating characteristic, Dense Vector, Bonn-Berry, Fault tolerance

Chapter 1

Introduction

1.1 Introduction

Dealing with big data encourages us to think about how we can store and process the large-scale data. This point requires us to understand how the big data are spread in the distributed data storage and processed by the distributed computing environment when compared with traditional systems that work in sequential order. The parallel implementation is more time efficient than its sequential counterpart and scaled up to process massive amounts of the data that can run on a single machine within multiple cores [**1**].

Many data serialization formats were developed for representing the information on the web. Both the standard Extensible Markup Language (XML) and Resource Description Framework (RDF) are some of the most important data models used to create data serialization formats and taking place in the " Data representation technology". The Data serialization formats are methods of sending and receiving data in semantic web technology, but these formats also have a critical problem of having large-scale datasets that can not fit in the memory of the machine. Therefore, the Distributed Computing and Data Storage environments should be considered for parsing and processing a big RDF and standard semi-structured data, such as standard XML datasets (e.g. Wikidata).

Through the growing number of edits on the knowledge bases such as Wikipedia, Stack-Exchange, Freebase, and Wikidata, the number of the ill-intentioned and erroneous edits have increased [**2**]. For instance, as a result of anyone can edit anything, Wikipedia and all of other projects of the Wiki-

media foundation prove that the crowd can be trusted to build and maintain reliable knowledge bases. Though this approach has been the successful for knowledge bases that utilize and depend on this model, these knowledge bases are also plagued by vandalism. Because of this problem, detecting such vandalizing and/or damaging edits is essential in the further development of knowledge bases.

Similar to the large-scale data parsing, applying vandalism detection on large-scale datasets is a challenge for an efficient vandalism detection. Hence, using Apache Spark as a parallel data processing framework and Hadoop ecosystem as a distributed large-scale data storage helps in exploiting the idea of the parallel processing on multiple nodes to process the big data. Additionally, we do experiments with the machine learning approaches to train a classifier efficiently that achieves a satisfactory performance in detecting vandalism [**3**]. This topic can be best explained using two steps: Distributed Data Parsing and Distributed Vandalism Detection.

1.2 Motivation

There are three factors that motivate the research undertaken in this study. First, there is a dearth of research on the use of a distributed computing environment and a distributed data storage for parsing large-scale RDF and standard XML datasets. This parsing technique in computing the large-scale datasets improves the performance and supports the scalability and the efficiency in parsing systems. Accordingly, wide range of lessons learned that should be considered in developing new parsers for different data representations.

Second, a wide range of sequential vandalism detection approaches that deal with Wikidata were implemented [**2, 4, 5, 6, 7**], but there is not a distributed vandalism detection approach to achieving this goal by an efficient and a scalable way based on a distributed computing and data storage environments. As a result of using Apache Spark and Hadoop ecosystem for dealing with large-scale data-sets in this type of distributed data parsing and vandalism detection, the horizon for new critical strategies to improve the accuracy and scalability is opened.

Third, re-engineering and implementing a distributed vandalism detection module in order to improve the accuracy in detecting the vandalism in Wikidata give us a chance to participant in the WSDM Cup 2019.

1.3 Objectives

The main objectives of the case study in this thesis are identifying and re-engineering two core modules: The first is a distributed data parsing module for parsing the data from both RDF data representing (RDF/XML, TRIX) and standard XML data representing. The second is a distributed vandalism detection module for Wikidata item pages, which are represented by a standard XML. Wikidata has already become the largest structured crowdsourced knowledge base on the web, and when it is taken into consideration that anyone can edit this knowledge base, it has a high probability to contain a vandalized content. Because of this, there is a possibility, we will find a vandalism in Wikidata.

In both cases, we are focusing on measuring the accuracy and scalability after implementing and re-engineering of these modules in a Distributed Computing Environment using Apache Spark and a Distributed Data Storage using Hadoop ecosystem to be able to deal with a large-scale data efficiently. This thesis considers the current traditional data parsing module and the traditional vandalism detection module before following a new strategy in an attempt to face challenges of implementing these modules via a Distributed Computing and Data Storage environments.

1.4 Contributions

This case study offers a solution to find an efficient approach for a distributed data processing which is called **"Bonn-Berry"**. The Bonn-Berry approach combines a distributed data parsing module and vandalism detection modules in one approach. Hence, the main contributions of this thesis are:

(1) Implementing a scalable and accurate distributed data parser. In this context, we distinguish between two different types of data parsers. The first is a standard XML data parser and the second is an RDF data parser for RDF/XML and TRIX. Hence, our system includes the following sub-Modules:

(1.1) Distributed RDF/XML parser.
(1.2) Distributed TRIX parser.
(1.3) Distributed Standard XML parser.

(2) Implementing a scalable and accurate distributed vandalism detection

module based on data parsing and features extraction and measuring the accuracy and scalability, when inputting large-scale data(e.g Wikidata).

1.5 Thesis Structure

In the following we explain the background to this thesis. In the light of this thesis, traditional data parsing and vandalism detection technologies are argued to be inadequate. The apparent lack of research on the use of distributed data computation and data storage environments in the data parsing and detection of vandalism in large-scale knowledge bases are the inspiration for this research case study. This thesis focuses on a distributed data processing and its contribution summaries under these heading:

(1) Distributed RDF data parsing module which parses the RDF/XML and TRIX RDF data.
(2) Distributed standard XML data parsing module which parses the Wikidata item pages.
(3) Distributed vandalism detection on Wikidata which is represented by the standard XML data representation.

Chapter 1 provides an introduction to main ideas are contained in this thesis and illustrates the motivations that have encouraged us to work on this case study. Additionally, this chapter clarifies the main contributions in the research case study.

Chapter 2 mainly focuses on state of the art review. This chapter reviews the literature on the importance of the sequential data parsing in extracting structured data from semi-structured data as well as examining sequential vandalism detection approaches that were implemented by other research teams to detect the vandalism in Wikidata.

Chapter 3 discusses the data parsing and vandalism detection technologies from an abstract point of view. Additionally, this chapter introduces the main component in the SANSA-Stack. Furthermore, it illustrates some details about the technical strategies that contribute to solving problems in this case study, such as using Apache Spark, Hadoop ecosystem and Apache Jena. Alot of concepts which improve our knowledge to be able to solve the problem in current case study are reviewed in details.

Chapter 4 shows in great details the methodologies that are used as pro-

posed solutions in this case study. In the beginning, we display the first module from the Bonn-Berry approach which is a distributed data parser. In this context, chapter 4 outlines the distributed data parsing topology, compares between distributed data parsing and sequential data parsing and clarifies the core idea behind the logic partitioning and physical partitioning. Further, it reviews the distributed RDF parser for RDF/XML and TRIX as well as the distributed standard XML parser. Then the chapter presents the second module from the researchers' newly developed the **Bonn-Berry** approach which concerns distributed vandalism detection. In this context, the distributed vandalism detection topology, comparison between distributed and sequential vandalism detection are reviewed. Moreover, the distributed vandalism detection module on Wikidata is examined. Furthermore, the context features, the content features and the machine learning classifiers are explained.

Chapter 5 reviews the **Bonn-Berry** Work-Flow on Wikidata. It explains the implementing techniques are used for implementing the Bonn-Berry approach. There are three main steps are explained in this chapter in the context of implementing:

1. Big data engineering module.
2. Computational big data module.
3. Classifiers module.

Chapter 6 outlines the research methods which are used in evaluating results of the Bonn-Berry approach as a distributed data process. Accordingly, this chapter illustrates using of the binary classification evaluator which is used for measuring the accuracy of the Bonn-Berry approach and display the limitations of this approach. Moreover, the chapter shows the role of horizontal scalability in the distributed systems.

Chapter 7 provides a summary and future work of the thesis.

Chapter 2

Related Work

This section gives a comprehensive overview of the literature on the data parsing and vandalism detection.

2.0.1 Sequential Data Parsing Approach

Some approaches rely on the **RDF4j** parser and **Apache Jena** parser which have been proposed as standard sequential RDF parsers based on its technique in parsing the data [8, 9]. In this context, a couple of points has to be taken into account. The first is both Apache Jena and RDF4J parsers take a place in a "Sequential parsing Approach" and read the RDF file as one object block by RDF object (The data will not be distributed but it will be uploaded in the memory as one block). By iterating over lines of the object that represents the input file, the parser extracts the triples (S-P-O). Specifying the RDF format type for parsing by the correct RDF Object is discovered automatically by the RDF model that depends on extension of the input file. The second is related to Apahe Jena and RDF4J parsers that deal with RDF file format and cannot parse the standard XML files.

In contrast, the new distributed parsing module deals with RDF file formats, including RDF/XML, TRIX and standard XML file dynamically. Throughout this case study, we focus on a distributed data parsing module that relies on streaming the standard XML and RDF files by using Apache Spark and Hadoop ecosystem as efficient technologies for distributed storage and computation. This module is similar to the preliminary approaches as discussed before, but extends it in two respects:

First, we parse the input file by using Apache Spark and Hadoop ecosystem in a pre-processing step which makes the parsing approach distributed

and capable of achieving a high level of scalability and correctness, comparing with the sequential parsing approaches which process the input as one object which is uploaded in the memory as one block. In the case of large scale input file , the object has a massive size and there is not enough memory for saving the object. Here, we learn how the distributed programming technique contributes in avoiding this problem.

Second, all of the previous parsing approaches [**2, 10, 11, 12**] do not parse the standard XML files. It focused just on the RDF files or standard XML individually, comparing with the new distributed approach which deals with a couple of these cases dynamically[**3, 8**].

2.0.2 Sequential Vandalism Detection Approach

In the previous studies, some approaches for vandalism detection have been proposed, such as **Buffalo-Berry** approach [**4**], **Conker-Berry** approach [**5**], **Logan-Berry** approach [**6**] and **Honey-Berry** approach [**7**]. All of the previous approaches relied on the area under the receiver operating characteristic **ROC** curve to measure the accuracy of the vandalism detection approach. We can measure the accuracy depending on the area under the **ROC** curve where the area of 0.5 represents a worthless test, compared with an area of 1, which represents the perfect test.

Accordingly, to the best of previous knowledge, the state of the art baseline testing for vandalism detection task was achieved by Heindorf and his research team[**2**] who is the vandalism detection corpus constructor. They relied on the area under the receiver operating characteristic **ROC** curve value for measuring the accuracy of the machine learning approach in order to achieve a precise vandalism detection module. In this context, they achieved the value (**ROC** $= 0.991$) [**2**]. The Buffalo-Berry approach and the Conker-Berry approach tried to improve the results. The first achieved the value (**ROC** $= 0.937$) and the second achieved the values (**ROC** $= 0.937 - 0.960$) [**4, 5**]. Comparing with the Logan-Berry and Honey-Berry approaches, they produced the values (**ROC** $= 0.920 - 0.986$) and (**ROC** $= 0.9441 - 0.959$) respectively [**6, 7**].

Recent vandalism detection approaches are correctly categorized as central vandalism approaches which do not rely on the distributed data computation and data storage environments (in contrast, the new module is a distributed vandalism module). Essentially, all of the recent approaches rely on the features extraction and re-engineering in addition to choose the suitable

classifier as a critical point for achieving the best accuracy. It is necessary here to clarify correctly what is meant by each approach and comprehend the properties of each one in more detail.

Buffalo-Berry Approach. The approach is used in this investigation relied on the re-engineering features from the previous approach [**2**] which depended on a wide range of features, such as content and context features. The mode of processing depended on 47 features which were implemented by Heindorf [**2**] as a corpus constructor and analyzer[**2**]. In the following, explaining about the most critical new features were implemented in this approach such as symbol ratio, which represented the proportion of the following characters in the string: (&, %, $, #, _, ~, ^, \). Then the researchers computed the most extended sequence of the same character feature, the similarity of the comment to the corresponding language label and number of editions were done by a given registered User [**4**].

In addition to extracting the item-id and user-id from the XML itme page, they checked if the revision was a minor one. In this context, in the exploratory data analysis, the researchers found that revisions in sessions which consisted the creation of an entity were never rolled back. They could not be sure if there were a technical impossibility to rollback a set of revisions that involved a new item. Also, different classifiers were implemented to get the best accuracy. Accordingly, improving the features and changing in some existing features in addition to setting the parameters of the classifiers played a vital role in achieving a better value for an area under the receiver operating characteristic **ROC** curve. The candidate algorithms [**13, 14**] for implementing the classifiers in this approach were **Logistic Regression** and **XGBoost** [**4**].

Conker-Berry Approach. In this approach, different strategies were followed. The researchers did not rely on the feature engineering which were processed from any prior work. For this challenge, they proposed to use three categories of features: Page-based features, such as page title. User-based features, such as user-name. Comment-based features, such as comment revision. Extracting the features based on JSON was not taken into account in this approach due to complexity in the extraction process compared with page-based features which were simple and easy to extract [**5**].

The research team in this approach worked on extracting the “path” features from the IP of the anonymous user. For instance, if we have an IP 192.168.10.1, we would generate the following text features: ”192”, ”192.168”,

"192.168.10" and "192.168.10.1". In the following, explanation about types of the comment tags which are used to extract the most important features in this case study.

(1) Structured comment. All information will be inside /* */.
(2) Unstructured comments. The information will be outside /* */.
(3) Links. All the wiki-links are inside the comment.

The candidate algorithms [**13, 14**] for implementing the classifiers in this approach were **Logistic Regression** and **XGBoost** [**5**].

Logan-Berry Approach. This approach is similar to the Buffalo-Berry approach. The features were built on top of the features were described in Heindorf [**2**]. Additionally, they added spam features that introduced a spam counting feature, which counted some spam and occurrence in training datasets. Spam features were the most important features that added to this approach. Categorical features like user-name and group had a significant data range, in addition to introducing the spam counting feature [**6**].

In this approach, as a result of the training dataset is too big for the system to handle, the researchers applied data sampling on the data. Due to the large size of the data set, they considered the learning algorithms which can scale well. The candidate algorithms [**13, 14, 15**] for implementing the classifiers in this approach were **Logistic Regression**, **XGBoost** and **Extremely Randomized Trees** [**6**].

Honey-Berry Approach. More recent techniques related to features have been tested by the researchers in this approach to achieve better results. Accordingly, this approach relied on a subset of features adopted in Heindorf [**2**]. By re-engineering existing features, experimenting with other features and extracting other new features, this approach was built [**7**].

In this context, "IsSpecContriUser" feature which checks if the user is a special contributor or not and "HasHashTag" feature which reflects existing the properties in the comment tag or not, are the newly extracted features in this approach. Both of these features contributed significantly to improving the value of ROC. As a result of size of the data that can not be handled by the machine, the researchers separated the training data into positive and negative examples then used all positive and negative examples only in the data set of Jan and Feb 2016 for validation. The fast and straightforward sampling method in this approach based on two points:

1. Random sampling strategy with a sampling fraction from negative samples.

2. Removing the duplicate contents of all data.

The candidate algorithms [**14, 15, 16**] for implementing the classifiers in this approach were **Multi-layer Perceptron**, **XGBoost** and **Extremely Randomized Trees** [**7**].

Since the previous approaches [**2, 4, 5, 6, 7**] follow the sequential programming technique, it can not deal with large-scale knowledge bases without taking into account the resources such as memory and cores number, where the approach is running. As a result of this problem, the distributed systems play a vital role in improving the scalability and achieving a satisfactory accuracy, and have a primary contribution for dealing with a large-scale data.In this context, the **"Bonn-Berry"** approach will be presented in details in the methodologies chapter as a proposed solution.

Chapter 3

Preliminaries

3.1 SANSA-Stack

The SANSA-Stack is an open source *data flow processing engine* for performing distributed computation over large-scale RDF datasets. It provides data distribution, communication, and fault tolerance for manipulating massive RDF graphs and applying machine learning algorithms on the data at scale[**17, 18**]. **Figure 3.1** illustrates the SANSA-Stack architecture. The current case study is a part of anomaly detection module in the SANSA-Stack.

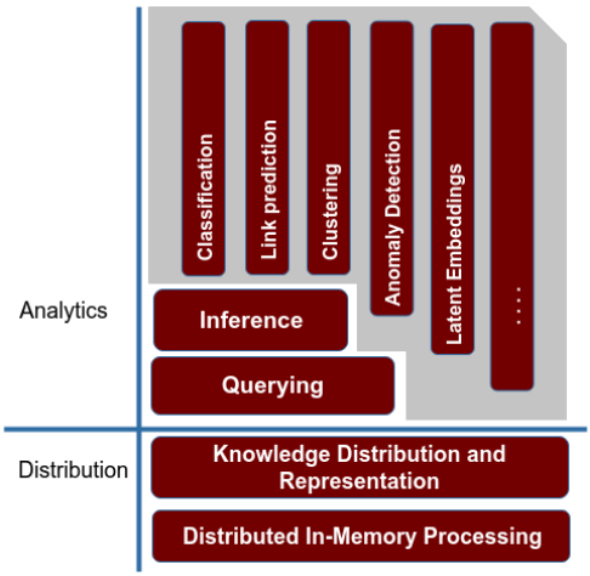

Figure 3.1: SANSA-Stack Architecture

3.2 Data Representations

There is a wide range of data representations for representing the knowledge bases, such as Turtle, Json, RDF/XML, Standard XML and TRIX(RDF triple in XML). In the current case study, standard XML, RDF/XML and TRIX are the primary data representations for datasets. In the following, a closer look at the structure of these types of data representations.

(1) Resource Description Framework/Extensible Markup Language (RDF/XML) Data Representation. RDF/XML is a syntax that was defined by the W3C. In this context, RDF/XML defines an XML syntax for RDF which is called RDF/XML in terms of name spaces in XML.

Figure 3.2 reviews the main tags which are contributory tags in building the RDF/XML dataset. The tags **(rdf: Description)(/rdf:Description)** are generally seen as a core tag strongly related to representing the RDF/XML data. Each colour in the figure represents an independent block of data that illustrates a fact or a group of facts.

(2) RDF Triples in XML (TRIX) Data Representation. Figure 3.3

```
<rdf:Description rdf:about="http://bio2rdf.org/taxon:41025">
        <rdf:type rdf:resource="http://rdf.geospecies.org/ont/geospecies#Bio2RDFtaxon"/>
        <skos:closeMatch rdf:resource="http://lod.geospecies.org/families/tNYux"/>
</rdf:Description>

<rdf:Description rdf:about="http://www.uniprot.org/taxonomy/41025">
        <rdf:type rdf:resource="http://rdf.geospecies.org/ont/geospecies#UniprotTaxon"/>
        <skos:closeMatch rdf:resource="http://lod.geospecies.org/families/tNYux"/>
</rdf:Description>

<rdf:Description rdf:about="http://dbpedia.org/resource/Corydalidae">
        <rdf:type rdf:resource="http://rdf.geospecies.org/ont/geospecies#DBpediaResource"/>
        <skos:closeMatch rdf:resource="http://lod.geospecies.org/families/mHM6h"/>
</rdf:Description>

<rdf:Description rdf:about="http://lod.geospecies.org/ses/CLcjS.rdf">
        <dcterms:title>About: Species Chauliodes pectinicornis, Family Corydalidae</dcterms:title>
        <dcterms:publisher rdf:resource="http://rdf.geospecies.org/ont/geospecies#GeoSpecies_Knowledge_Base_Project"/>
        <dcterms:creator rdf:resource="http://rdf.geospecies.org/ont/people.owl#Peter_J_DeVries"/>
        <dcterms:description>GeoSpecies Knowledge Base RDF: Species Chauliodes pectinicornis</dcterms:description>
        <dcterms:identifier>http://lod.geospecies.org/ses/CLcjS.rdf</dcterms:identifier>
        <dcterms:language>en</dcterms:language>
        <dcterms:isPartOf rdf:resource="http://lod.geospecies.org/ontology/void#this"/>
        <dcterms:modified>2010-07-15T12:42:09-0500</dcterms:modified>
        <cc:license rdf:resource="http://creativecommons.org/licenses/by-sa/3.0/us/"/>
        <cc:attributionURL rdf:resource="http://lod.geospecies.org"/>
        <cc:attributionName>Peter J. DeVries</cc:attributionName>
        <cc:morePermissions rdf:resource="http://about.geospecies.org/projects/"/>
        <foaf:topic rdf:resource="http://lod.geospecies.org/ses/CLcjS.rdf"/>
        <foaf:topic rdf:resource="http://lod.geospecies.org/ses/CLcjS.html"/>
        <foaf:primaryTopic rdf:resource="http://lod.geospecies.org/ses/CLcjS"/>
</rdf:Description>
```

Figure 3.2: RDF/XML Data representation

illustrates the core tags which are contributory tags to build TRIX data-set.

The tags **(Triple)(/Triple)** are generally seen as primary tags strongly related to representing TRIX data [**19**].

```
<?xml version='1.0' encoding='UTF-8'?>
<TriX xmlns='http://www.w3.org/2004/03/trix/trix-1/'>
        <graph>
                <uri>http://data.enel.com/LMF/context/charging-stations</uri>
                <triple>
                        <uri>http://data.enel.com/LMF/resource/charging-station/ENEL_00000078-address</uri>
                        <uri>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</uri>
                        <uri>http://www.w3.org/2006/vcard/ns#Work</uri>
                </triple>
                <triple>
                        <uri>http://data.enel.com/LMF/resource/charging-station/ENEL_00000078-address</uri>
                        <uri>http://www.w3.org/2006/vcard/ns#street-address</uri>
                        <typedLiteral datatype='http://www.w3.org/2001/XMLSchema#string'>Via A. Battelli</typedLiteral>
                </triple>
                <triple>
                        <uri>http://data.enel.com/LMF/resource/charging-station/ENEL_00000078-address</uri>
                        <uri>http://www.w3.org/2006/vcard/ns#postal-code</uri>
                        <typedLiteral datatype='http://www.w3.org/2001/XMLSchema#string'>56127</typedLiteral>
                </triple>
                <triple>
                        <uri>http://data.enel.com/LMF/resource/charging-station/ENEL_00000078-address</uri>
                        <uri>http://www.w3.org/2006/vcard/ns#locality</uri>
                        <typedLiteral datatype='http://www.w3.org/2001/XMLSchema#string'>PISA</typedLiteral>
                </triple>
```

Figure 3.3: TRIX Data representation

(3) Standard Extensible Markup Language (XML) Data Representation. There are much knowledge-bases rely on the XML representation for representing its pages, such as Wikipedia and Wikidata. As a primary example in the current case study, we use Wikidata standard XML item pages. Each XML page relies on **(page)(/page)** and **(revision)(/revision)** tags for representing the content of the page. **Figure 3.4** shows the structure of the standard XML page with wide range of different tags which will be explained in more detail in section **4.2.2**.

3.3 Data Parsing and Vandalism Detection Technology

The new approach is presented in this case study relies on concept of the distributed data processing. We illustrate an overview of the most important notions in this work. A complete overview is given by identifying two modules: **Data Parsing** module and **Vandalism Detection** module. Data parsing is a syntax or syntactic analysis which is the process of analyzing string symbols in natural language, computer languages or data structures with taking into account specific rules and grammars. Such parsing is sometimes used for different types of data; we have various commonly used parsers

Figure 3.4: Standard XML Data representation

for the different file formats.

Due to the fact that anyone may create and edit the Wikidata item page, it is frequently subject to vandalization. Hence, the basic idea behind vandalism detection is to be able to ascertain if anyone has caused vandalism in item pages of Wikidata. In this context, when using vandalism detection for finding contrast and anomaly in the data, a particular attention must be paid to the revisions that followed to the same item where each item page has one revision or more. Additionally, in context of the data parsing and vandalism detection modules, dealing with large-scale data sets represents a challenge against high scalability.

In all cases, the primary step is the data parsing. For this purpose, there are two basic approaches currently being adopted in our research. The first is a standard XML parser and the second is RDF parser. Standard XML parser is used for extracting a structured data from a semi-structured data (e.g. Wikidata) comparing with the second parser which is used to parse RDF/XML and TRIX files [**19**]. Since the primary approach should be applied to Wikidata item pages, we consider a standard XML parser as a core parser in the parsing module which will be an input for the second module "Vandalism Detection". After the feature extraction and using the **Mean**

and **Median** for engineering the data, the missing numerical values are filled by using the Mean and Median calculated values. Then "NA" value is used for filling the missing string values and most frequent case values are used for filling the missing Boolean values.

Receiver operating characteristic (ROC) plots **True Positive** rate vs. **False Positive** rate [**20**]. To present the area under the receiver operating characteristic **ROC** curve which measures the accuracy, five classifiers can be applied: Random-Forest classifier, Decision-Tree classifier, Logistic-Regression classifier, Gradient-Boosted-Tree classifier and Multi-layer Perceptron classifier. By comparing the results, we can conclude the best classifiers for the vandalism detection module. The use of the Apache Spark and Hadoop ecosystem as a distributed computing and data storage environments play a leading role in implementing any data processing pipeline.

3.4 Data Parsing Topology

From an abstract point of view, parsing knowledge bases means reading the data as an input, analysis of contents of the dataset, consideration the rules of language and generating a useful and more credible data. In other words, data parsing aims to extract a structured data from a semi-structured data. Despite of its commonly used, parser definition is used in different disciplines to mean different things. Almost of the data parsing processes share a number of the key steps as the **Figure 3.5** shows.

Data Reader. It is the first step in the data parsing process where the

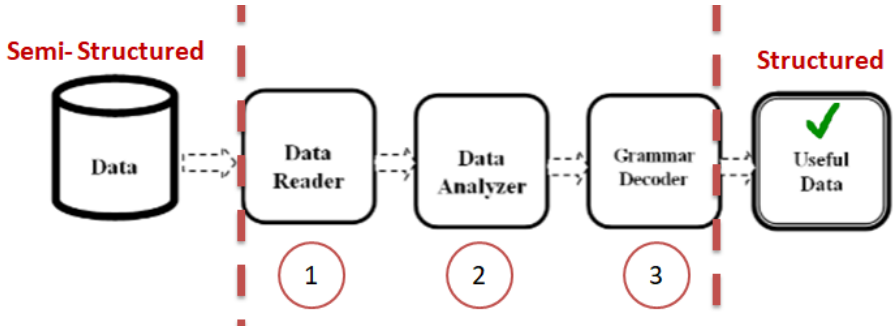

Figure 3.5: Data Parsing Flow

parser can read the data with taking into account the type of the data if it is a semi-structured or unstructured data.

Data Analyzer. It is the second step in sequence of the data parsing where the parser understands the structure of data and try to draw a structured map of the content of this data.
Grammar Decoder. Based on the type of the data, it converts the data from its current case to the helpful structured data case, such as (CSV records, triples) with taking into account the rules of the language which is used as a data representation language.

3.4.1 Distributed Data Parsing Approach vs Sequential Data Parsing Approach

Data Parsing has one concept and one meaning, but the strategies have been used for implementing this module are different. In this context, we can compare between structure of the distributed and Sequential data parsing approaches. The **Figure 3.6** gives a closer look at this comparison.

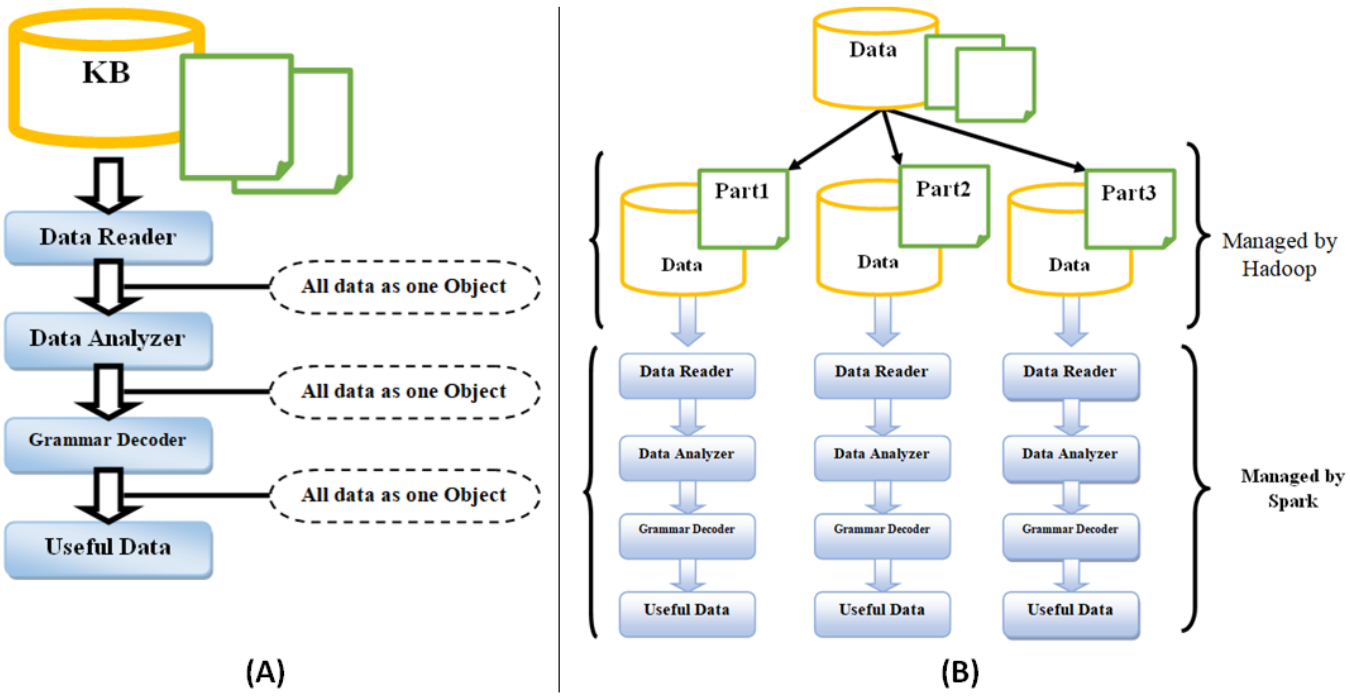

Figure 3.6: (A) Sequential Data Parsing vs (B) Distributed Data Parsing

Sequential Data parsing. In this parsing technique, the parser reads the data from the input , but there is no ability to analyse and apply the grammar decoder without collecting the data in one object. This step is essential

to keep the dependencies between the lines of the data, especially if each line in the model depends on the previous line for keeping the logic. **Figure 3.6** part(A) shows the sequential data parsing steps.

Distributed Data parsing. In the distributed data parsing, the data is partitioned into many parts and the parsing process is applied in parallel for all parts at the same time. The data input is divided into the blocks of HDFS as a distributed data storage then the data parts are partitioned again by range partitioning technique in Apache Spark. This technique is helpful because it reduces the runtime and improves the scalability. **Figure 3.6** part(B) illustrates how the data input is divided into three parts then each part is processed in parallel with other parts by Apache Spark. In this context, the redundancy which aims to increase the reliability, is achieved in the storage phase and computation phase.

3.5 Vandalism Detection Topology

Figure 3.7 gives an abstract overview of the vandalism detection, where we illustrate significant contributory steps to develop vandalism detection approach.

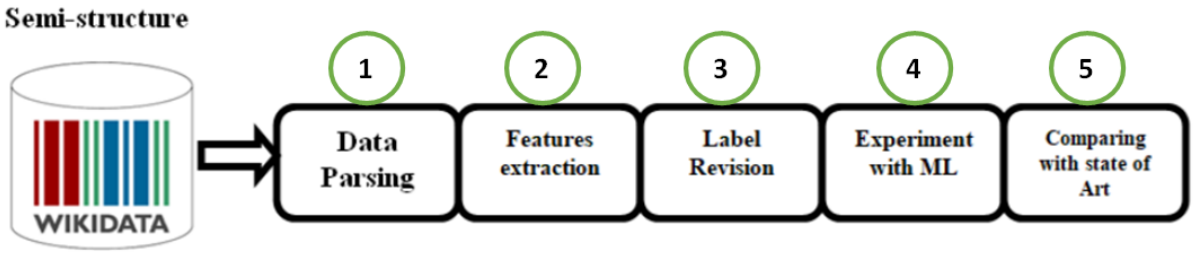

Figure 3.7: Vandalism Detection Flow

• **Data Parsing**. It converts the semi-structured data to structured and useful data. In the current case study, we use the standard XML data parser.
• **Features Extraction**. From the structured data, we extract the content and context features which will be explained in details in the sections (**4.3.2**) and (**4.3.3**).
• **Label Data**. After extracting the features, we label the revision by a list of the truth which has previous knowledge about the revision if it is a vandalized revision or not.
• **Training**. We train many classifiers and choose the classifier which achieves

the best results by checking the area under the receiver operating characteristic **ROC** curve value that measures accuracy of the vandalism detection module.

After applying all of the previous steps, we comparing the results with state of the art which represents the results of the previous approaches are implemented.

3.5.1 Distributed Vandalism Detection Approach vs Sequential Vandalism Detection Approach

Sequential programming approach has changed somehow from its original definition, especially after appearance new distributed data computation and data storage environments. Here, we compare between a distributed vandalism detection module and sequential vandalism detection module.

The core differences between a distributed vandalism detection and a cen-

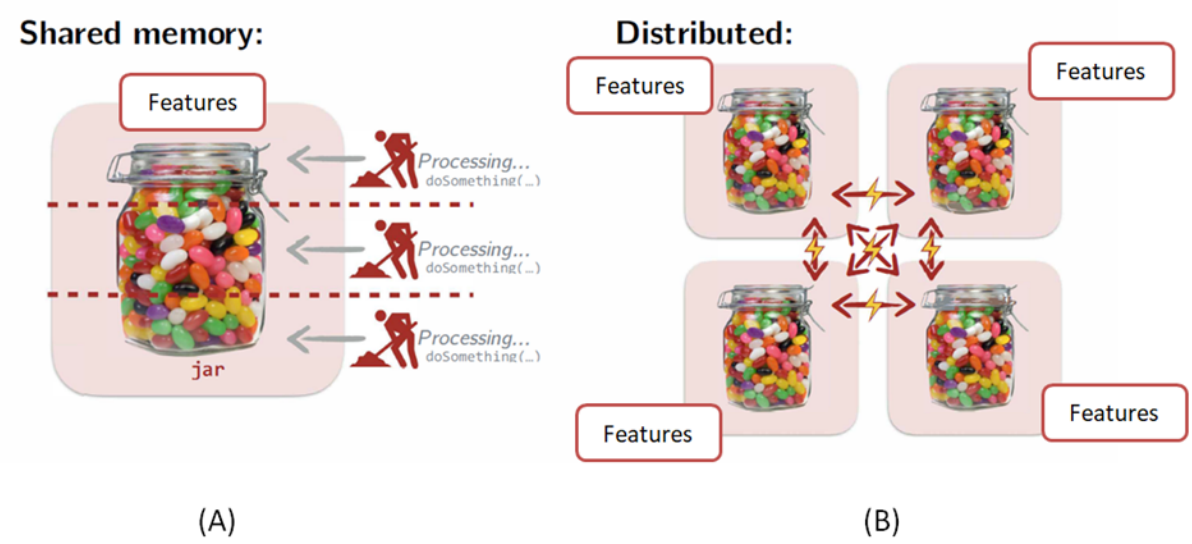

Figure 3.8: Sequential Vs Distributed Features Extraction in Vandalism Detection Module

tralized (sequential) vandalism detection are illustrated by the **Figure 3.8** [**21**]. After parsing the Wikidata item pages by a distributed data parser, we get a vector of features for each revision in the dataset. For instance, if we have one million revisions, that means we have one million feature vectors. On the one hand, in a sequential vandalism detection, all vectors of the

features are uploaded in memory of the machine then it will be processed sequentially step by step as we illustrate in **Figure 3.8** part (A). On the other hand, in the distributed vandalism detection as a partial example, the features are partitioned into distributed parts and processed in parallel as the **Figure 3.8** part(B) shows. Essentially, based on part(A), we learn how the shared memory technique is used in the computation phase, comparing with part(B), the distributed memory technique from Spark is used in the computation phase. Apache Spark by range partitioning which will be explained in detail in **3.7.5** optimizes the performance of the distributed vandalism detector.

3.6 Logic Partitioning and Physical Partitioning

In this context, we display a need to be explicit about precisely what is meant by the Physical partitioning and Logic partitioning.

Physical partitioning. By using Hadoop distributed file system (HDFS) as a distributed data storage, physically all the data will be distributed in the blocks of HDFS.

Logic partitioning. Reading the data of the revision from XML file relies on a streaming of the data which located between two tags: **(revision) fact (/revision)**. If we lose the logic partitioning which is represented by the sequential lines of the data between the tags (revision) fact (/revision), we lose the meaning and break the fact was represented by the revision.

The same for RDF/XML data which represents the fact by blocks between the tags **(rdf: Description) fact (/rdf:Description)** and TRIX file which represents the facts by the blocks between the tags **(triple) fact (/triple)**. In this context, the dataset is saved in the node of HDFS, where the name node keeps track of where the data block is stored for a particular file. Back up node keeps a copy of name node data. Accordingly, the data is distributed and saved in HDFS, then this data is partitioned by Apache Spark in computation phase which processes the data in parallel.

3.7 Apache Spark and Hadoop ecosystem Technologies

Apache Spark as an open-source cluster-computing framework and Hadoop ecosystem as an open source software framework for storage and large-scale processing of the data are the primary tools for implementing the distributed data parser and vandalism detector modules. The data structures in Apache Spark and Hadoop ecosystem, such as Hadoop RDD and Spark RDD are a robust critical point in a distributed programming approach, especially with a large-scale data [**1**].

3.7.1 Hadoop Ecosystem

The Hadoop Distributed File System (HDFS) is a distributed data storage used by Hadoop application. HDFS relies on two concepts: Name node and data node. **Name node** is the data manager which responsible for HDFS files. In this context, The files are split into blocks to be replicated and stored in **data nodes**. HDFS takes care of fault tolerance. In this case, any failure in any data node will not make any interrupting in the service [**22**].

Hadoop ecosystem includes many services to address the particular needs, but in current case study, we just depend on HDFS (Hadoop Distributed file System) for saving the data sources in distributed data storage system. **Figure 3.9** reviews the Hadoop ecosystem zoo [**23**].

One of the most critical advantages of Hadoop ecosystem is supporting **fault tolerance technology** which is a capability of a system to deliver uninterrupted service, although one or more of its components is failing. Based on data system part in **Figure 3.10**, we learn how if the data nodes (1,4) have a failure, the other data nodes (2,3) will contribute to achieve uninterrupted service. In this context, Hadoop distributed file system (HDFS) is used to scale a single cluster to many nodes. For availability as components fail, distributed file system **replicates** these smaller pieces into additional servers. The **redundancy** can be increased or decreased on a per-file basis or for a whole environment.

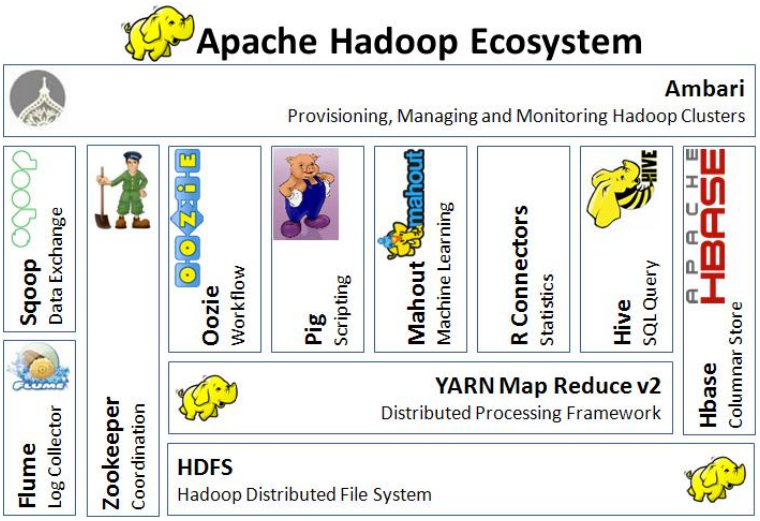

Figure 3.9: Apache Hadoop Ecosystem Zoo

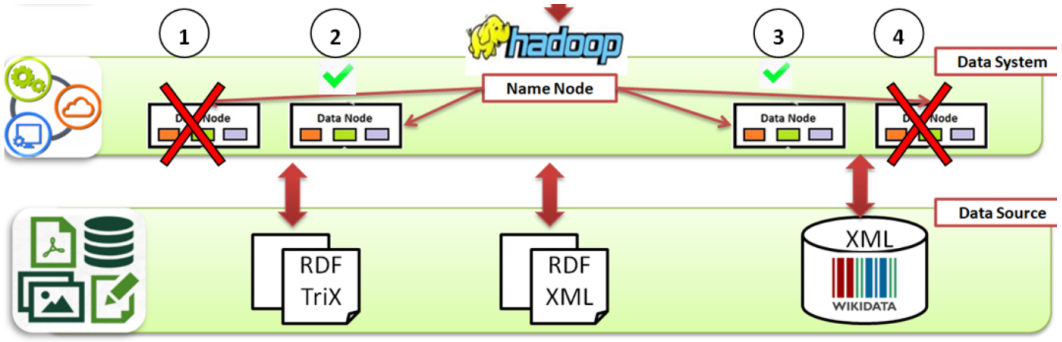

Figure 3.10: Apache Hadoop Ecosystem and Fault tolerance technology

3.7.2 Apache Spark APIs

Figure 3.11 illustrates the Spark core's API and Spark. There are many programming languages can be used in Spark such as Scala, Java, Python and R. Additionally, wide range of libraries such as Spark SQL, Spark streaming, Machine learning and Graphx can be used for implementing the pipeline machine learning [**3**].

We can understand a tale of the three Apache Spark APIs based on **Figure 3.12**,. It can be organized into Resilient Distributed Data-set (RDD), Data-Frame and Dataset. In the current section, we comprehend the areas where significant differences have been found, including these APIs [**3**].

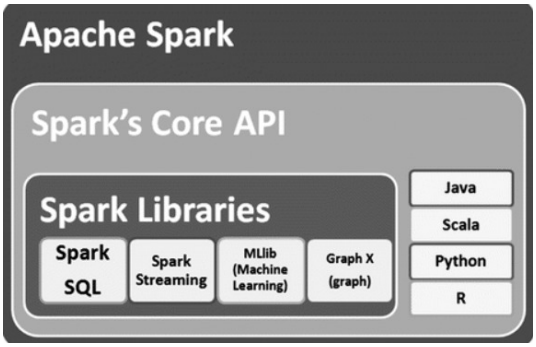

Figure 3.11: Spark Core's API and Spark Libraries

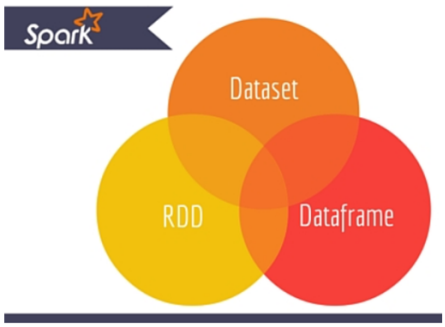

Figure 3.12: Apache Spark APIs

Resilient Distributed Datasets (RDDs), Data-Frame and Dataset differ not only in programming attribute but also in which they are distributed in memory of the machine. An Resilient Distributed Datasets(RDD) is an immutably distributed collection of elements of our data, read-only and partitioned across nodes in our cluster that can be operated in parallel. This can be better understood by breaking down the name RDD into three points:

1. Resilient for recomputing missing or damaged partitions due to node failures (i.e.fault tolerance).

2. Distributed due to the data residing on multiple different nodes.

3. Dataset represents records of the data we work with.

Figure 3.13 reviews a general idea of RDD and how it works. Without modification of RDD, new RDD is created by transformation which applies some functions on an RDD [**24**]. **Figure 3.14** shows how, via transformation, Spark does not execute the transformation immediately, but creates a lineage. A lineage tracks all transformations which have to be applied on

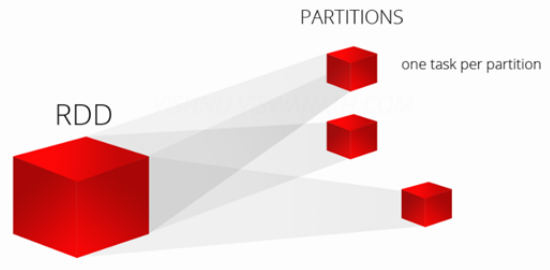

Figure 3.13: Apache Spark APIs - RDD

that RDD and creates a new RDD. In addition to these data structures, Apache Spark provides us with a distributed key-value data structure, which is called Pair RDD, and allows us to act on each key in parallel or regroup data across the network [**24**].

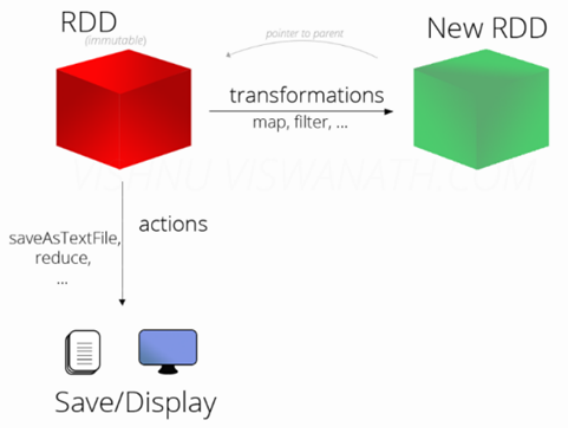

Figure 3.14: Apache Spark APIs - RDD Transformation

The differences between the RDD and Pair RDD are reviewed by **Figure 3.15**. Apparently, the role of the key in pair RDD is a vital role in fast indexing and fast retrieval as Pair RDD deals with ordered key data. This helps to support the performance of any operation which can be applied to this RDD [**1**].

Accordingly, there are some essential limitations for Spark RDDs [**1**]:

1. Handling the structured data, where RDDs do not infer any schema of the ingested data.

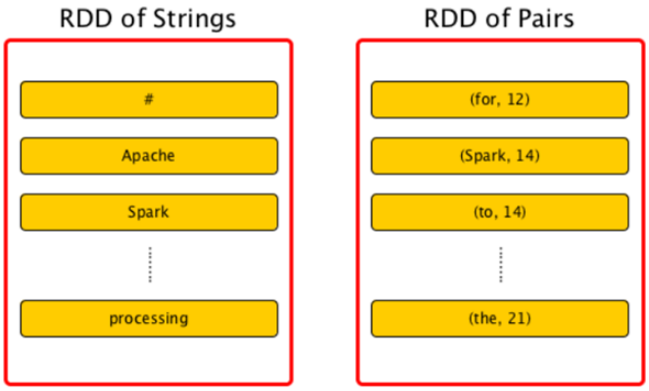

Figure 3.15: Apache Spark APIs - RDD Vs Pair RDD

2. Runtime type safety where there is not runtime type safety in spark RDD. In other words, runtime type safety means it does not allow us to check for errors at the same time we are compiling the program.

3. There is no input optimization engine.

Data-Frame is an immutable distributed collection of data that is organized into named columns, like a table in a relational database. **Figure 3.16** shows how we can create a data frame in three ways [**25**]:

1. Using various data formats. For instance, loading the data from CSV.

2. Specifying the schema programmatically.

3. Loading the data from current add.

comparing with data set, this is a collection of strongly-typed JVM objects.

Figure 3.16: Apache Spark APIs - Data Frame

Apache Spark ML. Spark ML is a package which aims to provide a set of high-level APIs which help programmer to create machine learning pipelines in the context of a distributed data computation [**3**].

Apache Spark SQL. The Spark SQL is a Spark module for structured data processing. Accordingly, the interfaces are provided by Spark SQL provide Spark with more information about the structure of both the computation being performed and the data. Internally, Spark SQL uses extra information to perform special optimizations [**26**].

3.7.3 Word2Vec Technique

Word2Vec is a helpful technique from Apache Spark which is used in machine learning and natural language processing. In this context, it creates a vector of words in the text corpus. The first step is constructing a vocabulary from the corpus; then it learns the vector representation of a word in the vocabulary. Accordingly, the vector representation can be investigated as a feature for machine learning and natural language process [**27**].

3.7.4 Dense Vector and Vector Assembler Techniques

Vector Assembler is a helpful technique for transforming and combining a list of column or vectors into a single vector. It is an important technique for training machine learning models [**1**].
Dense Vector is a class in a Spark module that is used to store double data types in order to be a compatible input to Spark ML methods [**26**].

3.7.5 Hash Partitioning vs Range Partitioning in Apache Spark

There are two types of partitions that Spark provides, and for our approach we make of use two of them: **Hash Partitioning and Range Partitioning**. Choosing the appropriate type relies on the action we want to perform on the data. Hence, there are vital factors affecting partitioning options:

1. Available resources.
2. Transformations used to derive RDD.
3. External data sources.

Hash partitioning spreads the data in various partitions depending on the key, in comparison to **Range partitioning** which relies on existing keys that follow a particular ordering to be efficient technique. Regarding configuring the Spark partitioning, it is available for pair RDD, and Spark cannot specify the worker node where the key will go, but it guarantees that a set of the keys will be together on some nodes [**28**]. In the current case study, we rely on the Range Partitioning technique because of having a unique Id for each revision and these revisions can be ordered based on the Id.

In the following figures (**3.17**) and (**3.18**), we compare between the processing technique before and after using range partitioning. In the first case, we have a vector of features. Without use Range partitioning, we can see how some partitions do not have any data for processing, and other partitions have extra data. This issue will make a load on some partitions while the other partitions will not have any data. Comparing to the second case where the Range partitioning technique is used and all partitions contain a data.

This technique is investigated in order to achieve a balance in processing the data. Based on the experiments in this case study, using Range partitioning improves the performance approximately 58%, but the key point is choosing the appropriate number of partitions in Spark context with taking into account size of the data and properties of the resources.

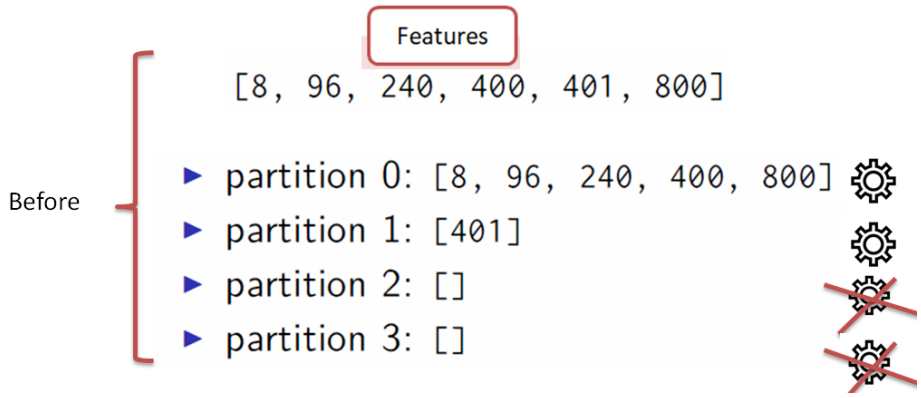

Figure 3.17: The Partitions Before Using Range Partitioning

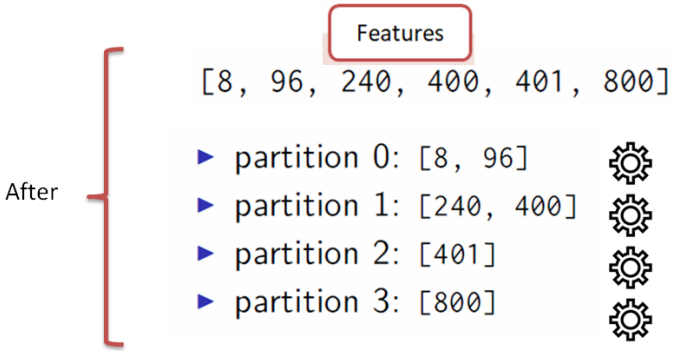

Figure 3.18: The Partitions After Using Range Partitioning

3.7.6 Cluster Architecture

The cluster architecture can fall under the following combination concepts. Spark applications run on a cluster as independent sets of the process. The coordination task is managed by Spark context object in the Driver (main program). For running on a cluster, Spark context connects to many types of cluster managers that allocate the resources across the applications [**29**].

After connecting, immediately the Spark gets executors on the nodes in the slaves(cluster). In this case, the process runs the computations and saves the data for our application. Hence, it sends our application code to executors. Spark context sends the tasks to executors for running, and each application gets its executor process that stays up for the period of whole application and runs the task in multi-threads. Accordingly, **Figure 3.19** shows the cluster overview. Each worker node has an executor. Moreover, **Figure 3.20** outlines the cluster node architecture [**29**]. In this context, the driver submits the job to the master and the master sends the task to the slaves which return the results to the master.

Apache Spark enables to use a distributed task dispatching, scheduling in addition to primary input and output functionalists. On the one hand, Spark uses a fundamental data structure known as Resilient Distributed datasets(RDD). RDD is a collection of data partitioning across the machine. On the other hand, RDD is a data built up through transformations which is lazy in general. Also, we can run an operation at any time by calling an action in RDD. In this context, driver program loads the code to cluster. When the code executes after every operation, it will be time and memory consuming because of each time data goes to cluster for evaluation [**29**]. Lazy evaluation in Apache Spark gives some benefits which fall in the following

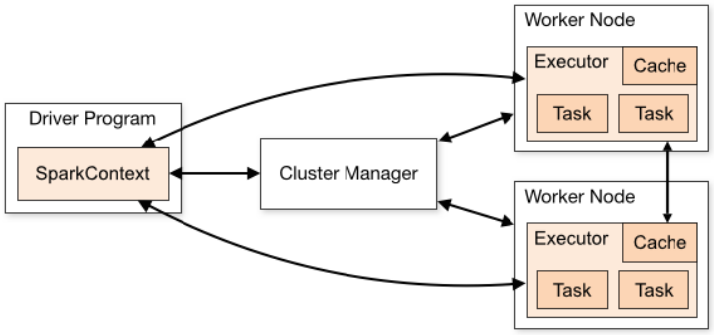

Figure 3.19: Cluster Overview

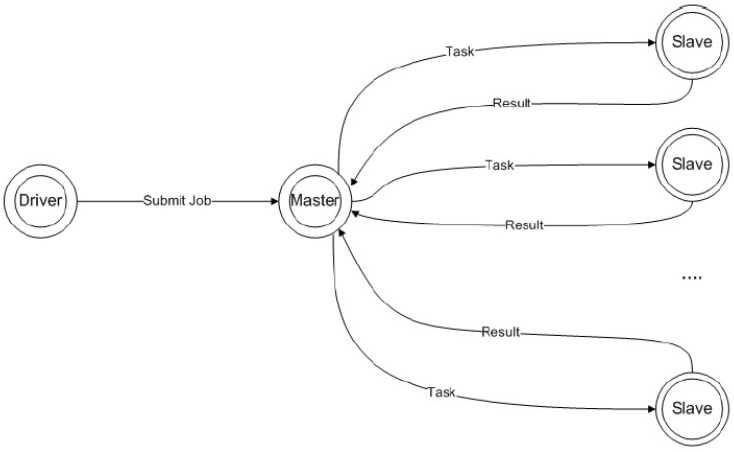

Figure 3.20: Cluster Node Architecture

points:

1. Users can organize their Apache Spark program into smaller operations as a result of the lazy evaluation.

2. Saving calculation overhead can be managed based on Spark Lazy Evaluation which plays a vital role in achieving this property.

3. Apache Sparks lazy evaluation can overcome time and space complexity.

3.8 Binary Classification Evaluator

Binary Classification Evaluator is an evaluator of cross-validated models from binary classifications, such as Random Forest classifier, Decision Tree classifier, Gboost Tree classifier, Logistic Regression classifier and Multilayer Perceptron classifier. The best model can be found by maximizing model of evaluation metric which is the area under the receiver operating characteristic **ROC** curve.

Figure 3.21 illustrates plotting the false positive rate against the true positive rate to create the the area under the receiver operating characteristic **ROC** curve. In other words, we can measure the accuracy by the area under the receiver operating characteristic (ROC) curve . Hence, **Figure 3.21** displays how the area of 0.5 represents worthless test comparing with an area of 1 representing the perfect test. For instance, in current case study, we use the area under the receiver operating characteristic **ROC** curve value for measuring the accuracy of the vandalism detection module which is a part of the Bonn-Berry approach.

If the area under the receiver operating characteristic **ROC** curve has a high value, the vandalism detector is more accurate. The area under the receiver operating characteristic **ROC** curve value is a build in parameter in binary classification evaluator so, we just need to call the **"areaUnderROC"** metric to produce the ROC value.

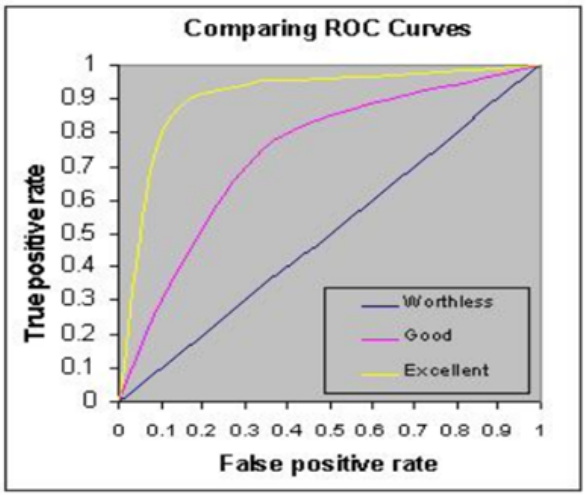

Figure 3.21: area under ROC "Receiver Operating Characteristic"

3.9 Machine Learning Classifiers

It is commonplace to distinguish different types of classifiers that will be illustrated in the following parts:

Random Forest classifier. The term "Random forest" is used to refer to a group of the tree predictors such that each tree relies on the values of a random vector sampled separately. There are many reasons why we use this type of classifier, such as there is no need to prune trees. The accuracy is generated automatically. Additionally, the over-fitting is not a problem, and it is easy to set the parameters.

In other words, in a forest composed of multiple decision trees, the algorithm classification results are voted by these decision trees. **Figure 3.22** shows how the forest composed of multi-decision tree [**30**].

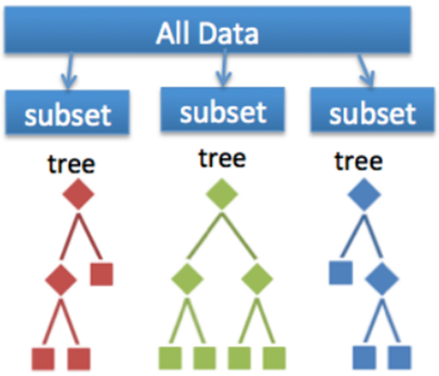

Figure 3.22: Decision trees in Random Forest Classifier

Decision Tree classifier. In this context, we illustrate a simple and widely used classification technique. It is a Decision Tree classifier which can be applied to solve many classification problems. Decision Tree classifier poses a series of carefully crafted questions related to the attributes of the records in data. It has an answer then follow-up question which is asked until a conclusion about the class label of the record is reached. In a tree structure, the decision tree classifiers organized a series of test questions and conditions.

The root nodes and internal nodes contain property and attribute test conditions to classify records that have various characteristics. Finally, the terminal node is assigned a class label (True or false, Yes or No) [**10, 31**]. **Figure 3.23** reviews the basic structure of a decision tree for the ideal tree

case [**32**].

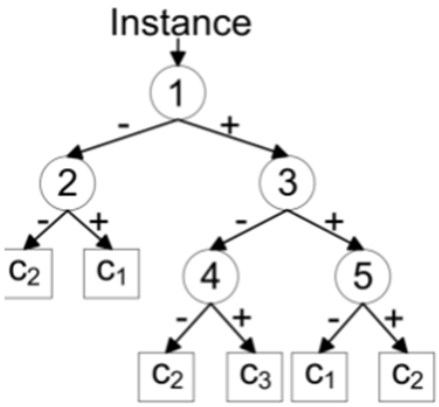

Figure 3.23: Decision trees

Logistic Regression classifier. In the case of dependent variable having just two values, the Logistic Regression can be used. On the one hand, if we have two values, such as 1 **and** 0 **or Yes and No**, we can deal with Logistic Regression. On the other hand, when the dependent variable has three or more unique values, such as "Married", "Single" and "Divorce", then the Multinomial Logistic Regression is usually reserved for the case [**33**]. Based on **Figure 3.24** part(B), we can comprehend how the Logistic Regression classifies the data in the case study, and figure **Figure 3.24** part(A) shows a 3D representation of Logistic classifier [**34**].

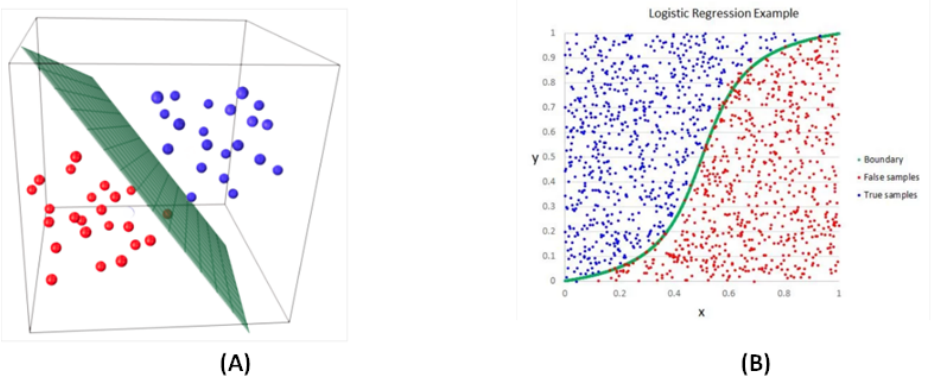

Figure 3.24: Logistic Regression

Gradient-boosted Tree classifier. This classifier does resembling for data many times to generate results which form a weighted average of resampling data. The core goal of using the tree boosting is to create series of a decision tree that builds a predictive model. Accordingly, a tree in the series is fit to residual of the prediction from the previous trees in this series.

In this context, the residual is defined in the field of derivative of loss function [**35**]. Gradient boosting model is an ensemble of many trees where the model is created in un-parallel. **Figure 3.25** shows how the classifier does ensembling the trees in a un-parallel way in our current case study [**34**].

Figure 3.25: Gradient-boosted tree

Multilayer Perceptron classifier. Based on the artificial neural network, we have a Multilayer Perceptron classifier. In this strategy of classification, each layer is connected to the next layer in the network. The input data is represented by the input nodes and all other nodes maps the inputs to the output by a combination of inputs with the nodes/weights and Bias and apply the function. A number of the inputs represents number of the features and number of the outputs represents the classes. **Figure 3.26** reviews MLP [**36, 37**] which we use in the case study.

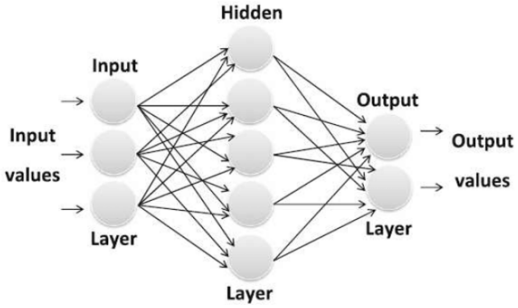

Figure 3.26: Multilayer perceptron

3.10 Over-fitting Management.

Over-fitting is a modelling error that occurs when a function is too closely fit to a limited set of data points. In this context, the model takes the form of making an overly complicated model to explain property in the data under study. To understand the over-fitting, an example of over-fitting from the real life is explained in the follwoing to be able to comprehend this concept in depth. If we want to achieve a prediction about the students from the University of Bonn that will land a job interview based on their curriculum vitae, we train a model from a data-set of 10.000 CV and their outcomes. Then we try the model out on the original data-set. It predicts outcomes with 99% accuracy. The bad news starts when we run the model on new data set of CV. We get only 50% accuracy.

3.11 Apache Jena and RDF4J Parsing Technologies

There are many data serialization formats for representing information on the web, and Resource Description Framework (RDF) is one of the most critical data models for achieving this goal.

RDF4J Parser. RDF4J is an open source framework from Java for processing RDF data. Hence, this includes parsing, storing, inferencing and querying of data. It offers an easy-to-use API which can be connected to all leading RDF storage solutions. RDF4J supports all mainstream RDF file formats, including RDF/XML, Turtle, N-Triples, N-Quads, JSON-LD and TRIX [**11**].
Apache Jena Parser. Apache Jena is an open source and free Java framework for building Semantic Web and Linked Data applications. Jena parser processes widely used RDF file formats, including Turtle, RDF/XML, N-Triples, JSON-LD. RDF/JSON, N-Quads, TRIX, RDF Binary [**12**]

Chapter 4

Methodology

4.1 Bonn-Berry Approach

In this context, we propose an approach we called it Bonn-Berry approach(BBA). It is a **Machine Learning**-based approach for data parsing and vandalism detection on Wikidata. The Bonn-Berry approach includes two modules: Distributed Data Parsing module and Distributed Vandalism Detection module. **Figure 4.1** shows how the BBA is a part of SANSA stack and located in the **anomaly detection** module in SANSA's architecture [**17**].

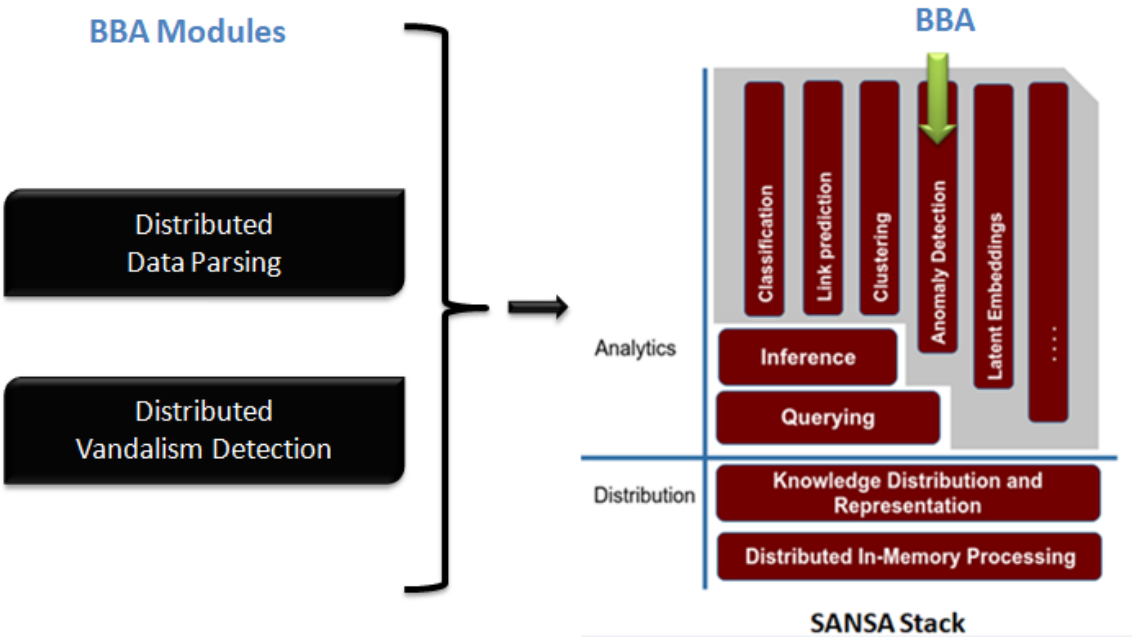

Figure 4.1: The Bonn-Berry Approach as a part of SANSA stack

4.1.1 Bonn-Berry Architecture

The Bonn-Berry architecture has built upon three essential parts which can be shown in **Figure 4.2**.

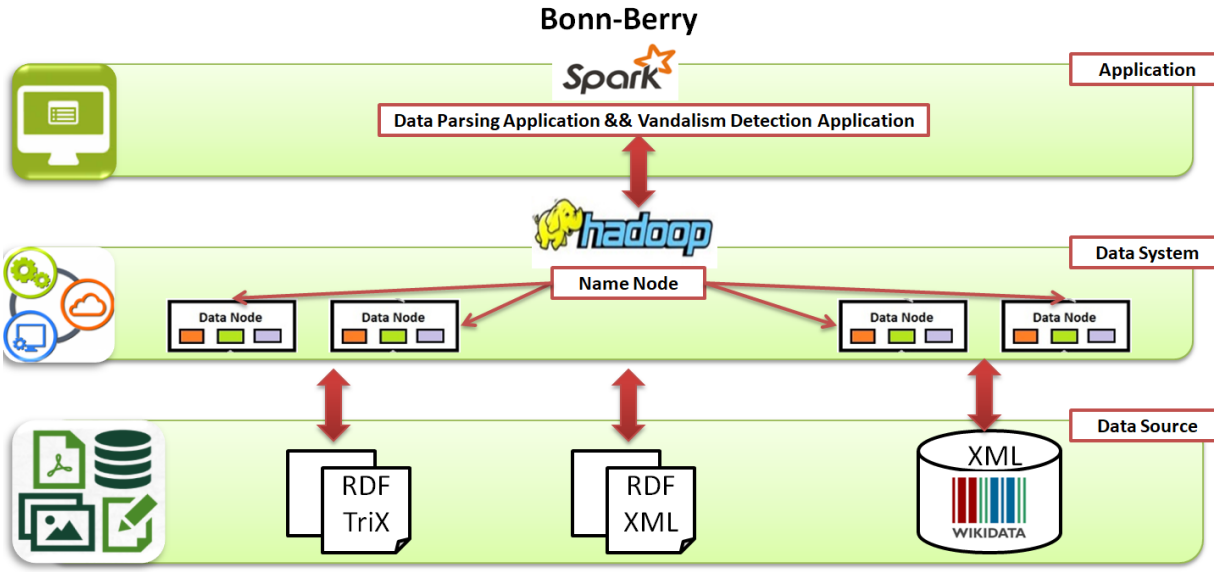

Figure 4.2: Bonn-Berry Architecture

(1) Data sources. In this part, there are the experimental datasets (Standard XML dataset: e.g. Wikidata, RDF/XML dataset: e.g. Geo–species, TRIX dataset: e.g. Charging–the station).
(2) Data system. Apache Hadoop ecosystem, where the data is saved in HDFS as distributed blocks of data.
(3) Applications. In this part, there are the Apache Spark applications which represent the implementation of the main algorithms in the Bonn-Berry approach.

In this context, the data sources are stored in the data nodes of HDFS, then the Apache Spark applications which are used to implement the data parsing and vandalism detection modules reads the data from Hadoop for processing. For optimizing the performance more and more, Apache Spark application uses the range partitioning technique which we explained in depth in the Section **3.7.5**. Hence the distributed data which is read from HDFS divided again in computation phase into partitions based on the ID of each record. Each record represents one revision, and each revision has a unique id.

Range partitioning technique uses the unique id in diving the record into partitions. This step makes the retrieval and processing of the data more efficient. Essentially, the most important key point in the **Bonn-Berry** approach is use of Apache Spark and Hadoop for distributed and fault-tolerant vandalism detection. In this case, the **redundancy** which aims to increase the reliability is achieved in the computation phase and the storage phase.

4.2 Distributed Data Parsing

4.2.1 Distributed RDF Parser Module

Distributed Data Parsing module is the first module from the Bonn-Berry approach. In this module, there are two RDF parsers: Distributed Resource Description Framework/Extensible Markup Language (RDF/XML) parser and distributed RDF Triples in XML (TRIX) parser.

(1) Distributed RDF/XML parser

There are many knowledge bases have much different representations, and RDF/XML is one of these representations. As a primary example in current case study, we use **GeoSpecies Knowledge Base** which includes information and facts about biological orders, families, species as well as species occurrence records and related data [**38**]. This parser has reveal several key points that are responsible for representing the knowledge bases by RDF/XML data representation.

In this context, **Figure 4.3** shows an explanation about the topology of the distributed RDF/XML parser. Accordingly, extracting the data from RDF/XML file uses **(rdf:Description)(/rdf:Description)** tags as **streaming parameters**. The dataset is divided into the blocks of HDFS. Then applying the steps which are illustrated by the topology in **Figure 4.3** and implemented by Apache Spark. The output of RDF/XML parser will be a group of triples **(Subject, Predict, Object)**.

(2) Distributed TRIX parser

As a primary example in current case study, **charging-station Knowledge Base** was used. This dataset includes geographical addresses for charging station, such as the address, city code and city name [**39**]. This parser has reveal several key points that are responsible for representing knowledge bases

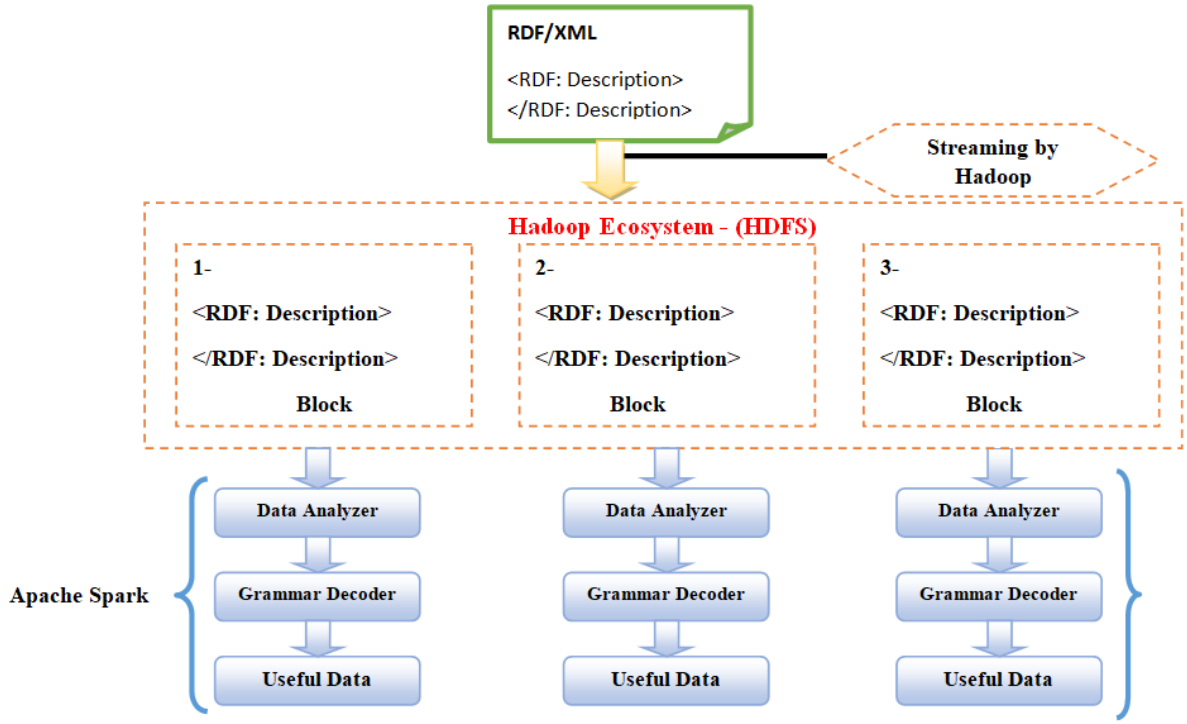

Figure 4.3: Distributed RDF/XML Parser Topology

by TRIX representation. **Figure 4.4** reviews the distributed TRIX parser.

Accordingly, extracting the data from TRIX file uses (Triple)(/Triple) tags as **streaming parameters**. The dataset is divided into the blocks of HDFS. After applying the steps which are illustrated by the topology in **Figure 4.4** and implemented by Apache Spark, the output of TRIX parser will be a group of triples **(S-P-O)**.

4.2.2 Distributed Standard XML Parser Module

Based on structure of the standard XML, widely varying tags have used. These tags are investigated to represent the Wikidata item page. Each revision has the following tags:

- Item ID tag **(title)(/title)**. It represents the item page to which the revision belongs. Each page has a unique title.
- Parent ID tag **(parentid)(/ parent id)**. It represents the previous

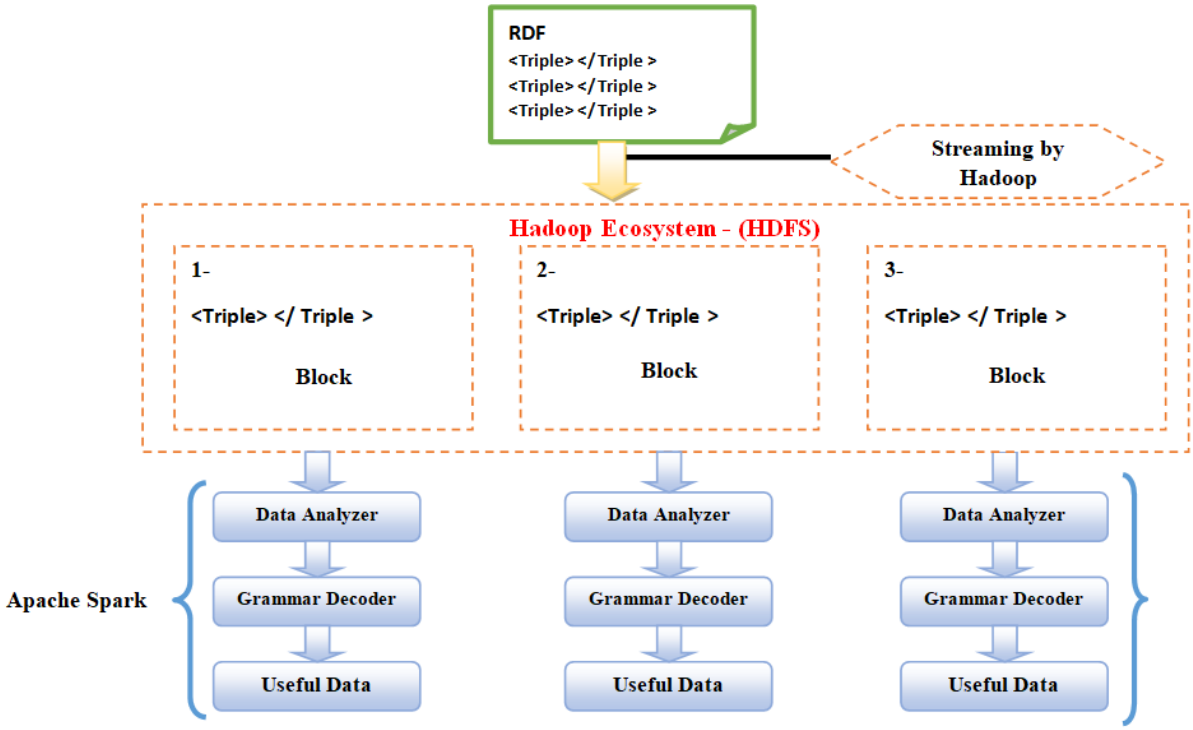

Figure 4.4: Distributed TRIX Parser Topology

version from current revision. If the revision is the first revision for the item page, that means the revision will not have a parent revision and the tags **(parentid)(/ parentid)** will not appear in representing of the item page.

- Timestamp tag **(timestamp)(/timestamp)**. It specifies details, such as time of revision
- Contributor tag **(Contributor)(/ Contributor)**. This tag has more than one possibility. It may include the contributor name and contributor Id in case the contributor is a registered user, or it contains just IP address tag for an unregistered user.
- Comment tag **(Comment)(/ Comment)**. In all cases, the comment is automatically generated to summarize the changes made as follows:

comment ::== (action) (subaction)? (param)+ (tail)

- Model and format tags provide general information about the item page and application.

- Json tag contains much information related to the revision, such as label, descriptions, aliases, claims, and site link.

Each item page has at least one revision. In this case, there is a particular issue that should be taken into account in a distributing standard XML parser. **Logic partitioning** between the tags is a sensitive point in the distributed parser because the data distributes inside the blocks of HDFS, compared to a sequential system, the parser reads the data line by line as order as its original sequences. Many XML representations were used to represent the knowledge bases. In this section, we clarify how we can extract the structured data from semi-structured data. The case study in this context is a Wikidata XML item page. **Figure 4.5** shows an example about the topology of the distributed standard XML parser. We have an item page which includes three revisions as an example. Each revision is distributed in the blocks of HDFS. Extracting and distributing the revisions inside the blocks of HDFS in Hadoop depends on Hadoop streaming process which takes the tags **(revision)** **(/revision)** as **streaming parameters** and extracts the data which located between this couple of tags.

After distributing each revision which has been represented by one record

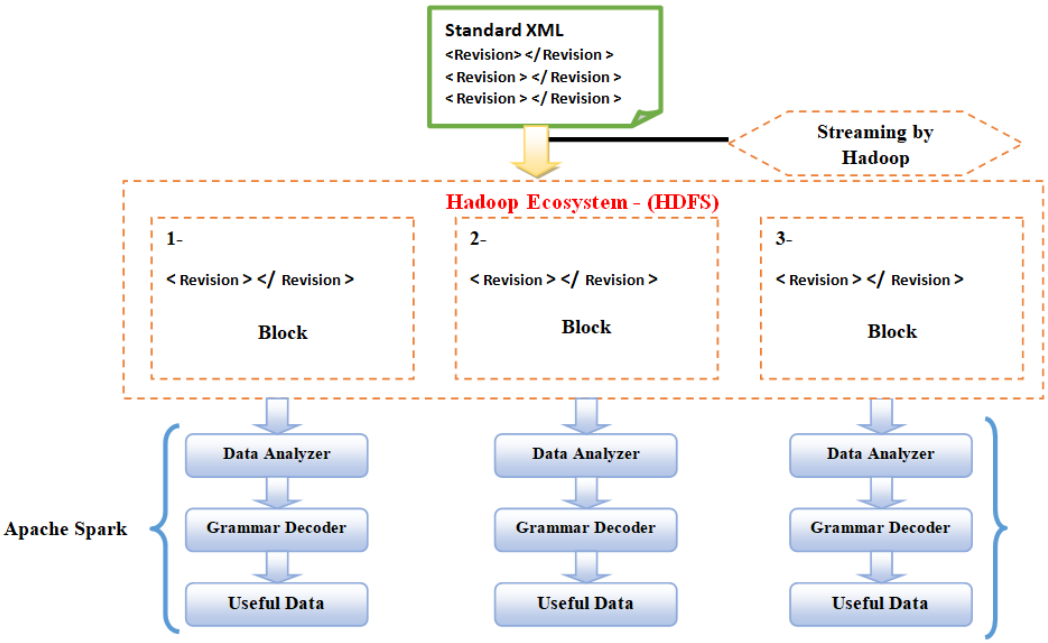

Figure 4.5: Distributed Standard XML Parser Topology

with all its tags, Apache Spark partitions the data. Then the data analyzer starts processing the data. In the next step, the grammar decoder maps the

data revision to be a helpful data. This case study found that a distributed data storage and a distributed computation environments have twice the impact on better scalability and correctness which clarify the role of the distributed data processing in dealing with large-scale data.

After applying the steps which are illustrated in the topology, output of the standard XML parser gives for each revision 16 values. Accordingly, these values will be used for feature extraction in the vandalism detection module. Table(4.1) gives a closer look at these values.

XML standard Parser	**XML standard Parser**
1. Reversion ID	9. Item ID
2. Parent ID	10. Time Stamp
3. Contributor IP	11. Contributor Name
4. Contributor ID	12. Json text
5. Label	13. Description
6. Aliases	14. Claim
7. Site-links	15. Model
8. Format	16. SHA

Table 4.1: Output's vector of standard XML Parser

4.3 Distributed Vandalism Detection

A Spark writes data parsing application, hence Spark divides the data and carries out the data range partitioning to improve the distribution concept and optimizes the performance. Accordingly, after parsing the data by the distributed process, the vandalism detection approach which is a part of Spark application starts in parallel to extract the feature from the structured data and experiment with machine learning model for evaluating the approach.

4.3.1 Features extraction concept

An example of Wikidata item page and its three revisions as an example can be reviews by **Figure 4.6**. Each item page has an **item head** and **item body**. There is a wide range of the terms distributed on the item page for representing the data, such as label, description and item title. Based on

Figure 4.6, the **label** is "Jens Lehmann", the **description** is " full professor university of Bonn" and the **item title** is "Q54833759".

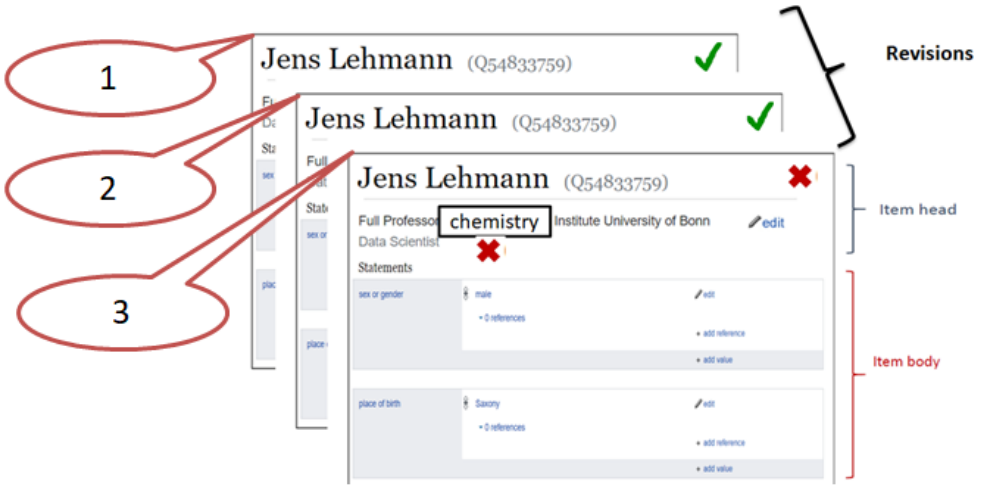

Figure 4.6: Wikidata item page with its revisions

Registered and unregistered users of Wikidata have been represented as a significant source of contribution in vandalized item page in the Wikidata knowledge base. Accordingly, the registered users and unregistered users can modify and add new revision, and this policy increases the possibility to have vandalism in the knowledge bases. For this reason, the needs for vandalism detection modules for Wikidata has increased. **Tracking the historical changes** on Wikidata item pages plays a crucial role in building new features before training the data. Hence, we extract helpful features that reflect the changes and modifications on the data. Almost of the new features based on the hinting the changes between the first revision in the item page and its children.

Administrators of this type of open source knowledge bases do not have enough time to check the vandalized revisions in large scale knowledge base periodically. Thus, one of the core goal in current methodology is designing an efficient vandalism detection module for the Wikidata knowledge base to predict the vandalized and unvandalized revisions. The new vandalism approach hints at vandalized revisions through a wide range of precision – recalls points to enable different use case. This approach is based on 71 re-engineered previous features and 25 new engineered features. In this context, we have two core categories of features: **Content and Context features**. These features can be classified based on its data type under the following categories:

1. **Double-Features**. This type of features is extracted by calculating the ratio (e.g. Ratio of upper case character).

2. **Integer-Features**. This type of features is extracted by calculating the frequency(e.g. Number of the revisions belong the item page).

3. **Boolean-Features**. This type of features is extracted by checking the existing special text in the revision(e.g. Existing Bad Word or not).

4. **String-Features**. This type of features is extracted by checking the existing special parts in the revision (e.g. Action and Sub-action in the revision).

4.3.2 Content features

Based on Heindorf et al.[**2**], we employ, re-engineer and add many features in the new module. In total, the content features have been classified as belonging to Characters features, Word features, Sentences features and Statement features. Accordingly, we employ 25 Character features, 17 Word features, 4 Sentence features and 3 Statement features as content features.

(1) Character level features. At the character level, there are many cases which can contribute to extract the features that will be used later with experiment machine learning model. Unexpected character sequences, such as a repetitions of character, executive capitalization, exceptional character and missing spaces, form the primary goals for calculating the ratio. Additionally, calculating the ratio for the natural languages such as Arabic, Indian, Bengali, Brahmi, Chinese and Persian can make sense in some cases. For instance, when the user try to modify an English item page by one of these languages, there is a high possibility to be a Vandalized user. [**2**]. We re-use the character features from Heindorf et al and Buffalo-Berry [**2, 4**] and re-engineer some of them, and drop others which don't make sense to detection performance.
New Character features. The new features which we add to the new module fall under 6 headings: Hexadecimal ratio, control ratio, blank ratio, printable ratio, visible ratio and alpha ratio.

(2) Word level features. At the word level, to quantify words usage, we re-use the words features from Heindorf et al.[**2**]. Then we re-engineer Band word ratio, female first name, male first name, containing the lousy word, including ban word, proportion number of the links, number of language word and Qids in the revision. Many new features are depened on the

similarity to hint the changes between the revisions such as Jaccard Distance and Jaccard Index [**40**]. Jaccard Index makes a comparison among elements for two sets (e.g. two string values) to see which parts are shared and which are distinct. Compared with Jaccard Distance which is the complement of the Jaccard index and can be calculated by subtracting the Jaccard Index from 100%.
New Word features. In this features level, we rely on new features which fall under two headings:

1. Similarity the current revision with its parent revision. This depends on comparing the current revision and parent revision, such as comparing number of the shared words between the comment tail of current revision and comment tail of the parent revision.

2. Similarity the current revision with its parent revision without taking into account the stop words by comparing number of the shared word without stop word between the words in the comment tail of current revision and comment tail of the parent revision.

(3) Sentence level features. At the sentence level, comment tail is a core to quantify sentence usage. Based on Heindorf et al.[**2**], we use four features. Comment tail length, similarity comment contains a site link and label, similarity comment contains a label and site link.
New Sentence features. In this features level, we add a new feature which represents the similarity between the comment tail of the current revision and the comment tail of the parent revision.

(4) Statement level features. At the statement level, we re-engineer the features based on Heindorf et al. [**2**]. In this context, we do not add any new features. Tables **(4.3)**,**(4.2)**,**(4.4)** shows all content features are used in the case study and shows the previous and new features which are added.

Content features Part(1)	
Sentences Features	**Statement Features**
1.Comment tail length	1.Property of Revision
2.Similarity-Comment-Contain-Site-Link with-label	2.Data value of revision
3.Similarity-Comment-Contains-Label with-Site link	3.Item value of revision
4.Similarity-current-comment-tail-with Parent**(New)**	

Table 4.2: Content features (Sentences and Statement Level)

Content features Part(2)		
Character Features	**Character Features**	**Character Features**
1.Upper-Case-Ratio	9.Punc-Ratio	17.Tami-Character-Ratio
2.Lowe-Case-Ratio	10.Long-CharacterSeq-Ratio	18.Telugu-Character-Ratio
3.Alphanumeric-Ratio	11.Arabic-Character-Ratio	19.Symbol-Character-Ratio
4.ASCII-Ratio	12.Bengali-Character-Ratio	20.Alpha-Character-Ratio**(New)**
5.Bracket-Ratio	13.Brahmi-character-Ratio	21.Visible-character-Ratio**(New)**
6.Digital-Ratio	14.Cyrillic-Character-Ratio	22.Printable-Character-Ratio**(New)**
7.Latin-Ratio	15.Han Character-Ratio	23.Blank-Character-Ratio**(New)**
8.White-Space-Ratio	16.Malaysia-Character-Ratio	24.Control-Character-Ratio**(New)**
		25.HexaDecimal-character-Ratio**(New)**

Table 4.3: Content Features (Character level)

4.3.3 Context Features

Based on Heindorf et al.[**2**], we employ, re-engineer and add many features to the new module. Different features have been proposed to be classified as context features. In context features, we have User level features, Item level features and Revision level features. As a result of this classification, in total we employ 20 User features, 4 Item features and 24 Revision features.

(1) User features. At the user level, we use the same features in Heindorf et al and Buffalo-Berry [**2, 4**]. We distinguish between two types of users. **Registered** users who have a User Name and User Id comparing with **unregistered** users which have just an IP address(IP4, IPv6). From the meta-data file which is provided by the vandalism detection corpus administrator [**41**], we obtain Geo-information about the users in Wikidata

Content features Part(3)	
Word Features	**Word Features**
1.Language Word Ratio	9.Female-First name
2.Contain language Ratio	10.Male-First name
3.Lower case Word Ratio	11.Is-Contain-Bad word
4.Uppercase Word Ratio	12.Is-Contain-Ban Word
5.Longest word	13.CurrentRevision-ParentRevesion-shared-words**(New)**
6.Is Contain URL	14.Current Revision-Parent Revesion-Shared-Words-Without-Stop-Words**(New)**
7.Bad Word Ratio	15.Proportion-Qid
8.Ban word Ratio	16.Proportion-Language-Words
	17.Proportion-Links

Table 4.4: Content Features (Word level)

knowledge base such as user-city, user-country, user-city-code, user-country-code and user-continent.
New User features. In this context, we add IP long value for unregistered user as a new feature where we convert the IP address to a numerical long value, in addition to "HasBirthdate" and "HasDeathdate" features.

(2) Item features. At the item level, to quantify items usage, we re-use the item features from Heindorf et al and Buffalo-Berry [**2, 4**].

(3) Revision features. At the revision level, to quantify Revision usage, we re-use the revision features from [**2, 4, 5, 6, 7**]. Based on meta-data file [**41**] which has been provided by the vandalism detection corpus administrator, we get the revision session and revision tags.
New Revision features. In this feature level, we re-engineer a lot of new features such as the number of (label, description, aliases, claims, site link, statements, references, qualifier, qualifier-order, badge) for each revision as **new features** at revision level, comparing to the previous approaches, they used these features at the level of item. Experimentally, using these features in this revision level is more efficient in improving the accuracy comparing with using the features in Item level. More new features can be genertaed by studying the historical changes between the current revision and parent revision such as the parent's sub action and parent's action. Then we use

item time stamps, JsonTextsize, action of the partner revision, sub-action of the revision, parent ID and revision time-stamp. Tables **(4.5), (4.6)**, **(4.7)** outline all context features that are used in the case study and show the previous and new features which are added.

Context features Part(1)
Item Features
1.Number-Revision Of Item Has
2.Number-Unique-User-Item
3.Frequency-Item
4.Item-Title

Table 4.5: Context Features (Item level)

Context features Part(2)		
User Features	**User Features**	**User Features**
1.Is-Privileged	8.User-ID	16.User-Country-name
2.Is-Bot-User	9.Has Birth Date**(New)**	17.User-Frequency
3.Is-Botuser-Withoutbotflag	10.Has-Death-Date**(New)**	18.No-Unique Items-User Contribute
4.Is-Property	11.User-County-code	19.No-Unique-Items User Edit
5.Is-Translator	12.User-Content-code	20.Unique-Items-User
6.Is-Register	13.User-Time-zone	
7.IP-Long-Value**(New)**	14.User-Region-code	

Table 4.6: Context Features (User level)

4.3.4 Classifiers and Training Dataset

Classification is a vital strategy in the field of data mining. Hence, we have a set of records which are called the training data-set. Each record has several

Context features Part(3)		
Revision Features	**Revision Features**	**Revision Features**
1.Revision-Session	9.Revision-Sub-Action	17.Number-label-values-InRevision**(New)**
2.Revision-Tags	10.Number-Badges Values-In Revision**(New)**	18.Number-Description-values-In Revision **(New)**
3.Language-Revision	11.Revision-Parent-ID	19.Number-Aliases-Values-In Revision**(New)**
4.Revision-Language Local	12.Parent-Sub-Action **(New)**	20.Number-Claims-Values-In Revision**(New)**
5.Is Latin-Language	13.Content Type.	21.Number-Site-Link-values-In Revision**(New)**
6.Json-Length	14.Bytes-Increasing Between-Current Revision and-Parent Revision**(New)**	22.Number-Qualified-Order Values Revision **(New)**
7.Revision-Action	15.Time-Since-Parent-and-Current Revision**(New)**	23.Number-References-Values-Qualifier **(New)**
8.Parent-Revision-Action**(New)**	16.Number-Label Values-In Revision	24.Number-Statement Values-In Revision**(New)**

Table 4.7: Context Features (Revision level)

attributes. Accordingly, one of the categorical attributes which are called the class label. It points the class to which each record belongs. The primary goal of classification is to use the training data set to build a model of the class label which can be used to classify the new data that its class labels are unknown [**42**].

Detecting over-fitting is an important step, but it does not solve the main problem. Fortunately, we have several options to try solving this issue, such as Cross-validation, Train with more data and Remove features. We can choose the most suitable strategy depending on our data and results. In current case study, we choose two options. The first is training with more data and the second is to remove the feature. Based on the final Receiver Operating Characteristic (ROC) results, Random-Forest classifier is the best. We clarify this point in details in evaluation chapter in context of accuracy. In current case study, the **over-fitting** is appeared as a result of using some types of features such as sentence and statement features. In the

evaluation chapter, we can see the **Figure 6.6** which shows how the ROC value decreased when we use the sentence features and statement features comparing with other features which contribute in improve this value to be better.

Chapter 5

Implementation

We can use different programming languages in Spark, such as Java, Scala, Python and R. In implementing the current case study, we use scala IDE and almost of implementing functions rely on Spark ML.

5.1 Bonn-Berry Work-Flow on Wikidata

The work-flow diagram for implementing the Bonn-Berry approach on Wikidata as the primary dataset is reviewed in **Figure 5.1**. We do not take into account the RDF/XML, and TRIX datasets in the vandalism detection module for two reasons: First, vandalism detection on the RDF datasets is not required as a contribution in this case study. Second, there are no previous vandalism detection modules that deal with this type of RDF file. Hence we can not evaluate the vandalism detection in RDF case, hence the part of current case study which related to vandalism detection is designed just for Wikidata (standard XML).

The first step in implementing the Bonn-Berry approach is a reading the dataset from HDFS. In the next step, we stream the revisions depending on the streaming parameters which represent the tags of the revisions and save them in Hadoop RDD. To be able to apply some action by Apache Spark, we save the Hadoop-RDD in Spark-RDD. In this step, the data parsing process extracts the data to recognize the dataset as a helpful structured data which is saved in a data-frame. In the next step, the data frame is the input of the vandalism detection module which extracts the features. For each revision, we have vectors of features which are reorganized in a data frame for applying the classifiers. After applying the classifiers and binary classification evaluator, we get ROC value for measuring the ability of the

module to achieve its tasks precisely.

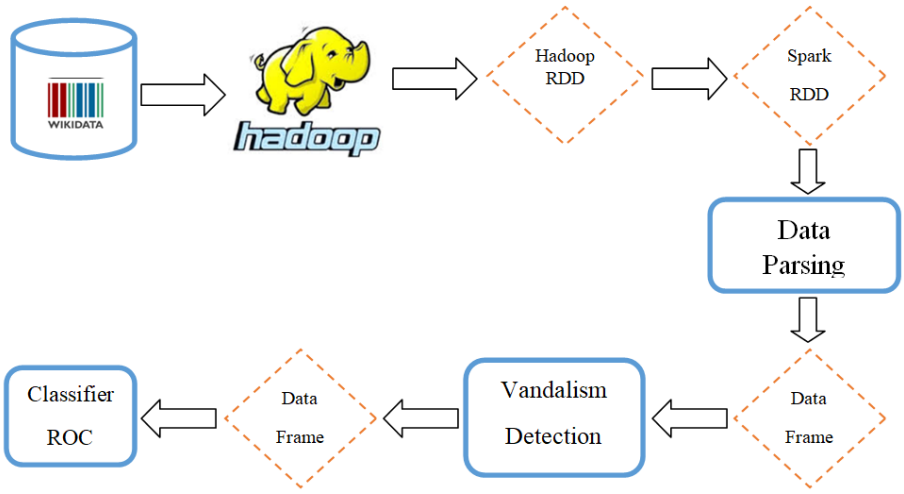

Figure 5.1: Bonn-Berry Work-Flow on Wikidata

5.2 Big Data Engineering Module

(A) Exploring the Data. The goal of exploring the data is understanding nature of the data. In other words, exploring the data analyses the data set to detect its characters and properties. Accordingly, we explore the Wikidata item page to understand tags of the page that represent the data in the item page.

(B) Data Parsing. Based on exploring the data, the data will be parsed by using a parser that extracts the structured and helpful data from a semi-structured data in current case study.

(C) Preparing and cleaning the Parsed Data. After parsing the data and getting the structured data from semi-structured data, we prepare the data in a way which can be suitable for vandalism detection module. For example, we eliminate from the data unhelpful symbols and characters.

(D) Packaging and Integrate th Data. Packaging the data in this case study includes classification of the data tags. For vandalism detection module in the Bonn-Berry approach, the original input is Wikidata which is groups of item pages. These Item pages are represented by the standard XML representation and parsed by standard XML parser. The tags of Wikidata item

pages are categorized under two main types:

1. Standard tags which are generated automatically by Wikidata application.

2. Json tag which reflects the changes that are applied on the item page by the users.

5.3 Computational Big Data Module

(A) Computing of Features Extraction process. Based on features which were explained in the sections **(4.3.2, 4.3.3)** , we find the values of features for each revision. Then we collect values of each level feature in a spreading vector for each revision. The values in the vectors have different data types (Integer, Double, Boolean and String). In this context, we give some example of features extraction. For instance, one of character feature is an uppercase character. Based on the following sentences we calculate the ratio value for the character feature.

$$RatioUppercaseCharacter = \frac{Numberofuppercasecharacter}{Numberofallcharacters}$$

Case1: Dr.Damien Graux speaks English
Result = 4 / 30 = 0.133

Case2: Dr.damien graux speaks English
Result = 2/ 30 = 0.066

(B) Missing Values Process. After getting vectors of the values based on the feature extraction process, we have missing values. Filling the missing values process is an essential factor in successful implementation in Apache Spark because almost all of the algorithms in Spark ML cannot deal with missing values. In this context, we fill the missing numerical values by the calculated **median** and **mean** values for the numerical column. Regarding the Boolean values, we fill the missing values by the most frequent value comparing with the string values; we use **"NA"** for filling the missing values.

(C) Features Vectors Engineering. After feature extraction and processing the missing values, we should collect all the values for each revision in one vector, but the problem is the data types of the values are different. Regarding the Spark Machine learning library, it takes a **dense vector** as input, and the values in the dense vector must be double type. Hence, we

convert the integer values to double value, convert the Boolean values from **True/False** to zero/one values and use **Word2Vec** technique to convert the string values to numerical double values.

(D) Vectorization Process. Finally, after applying all the previous steps, we have numerical values for all values of the vector for each revision. We assemble all of the vectors in one vector by applying **vector assembler**. Figure**(5.2)** reviews an example about how we convert the values from its original type to double then assemble all of the double values in one dense vector for each revision.

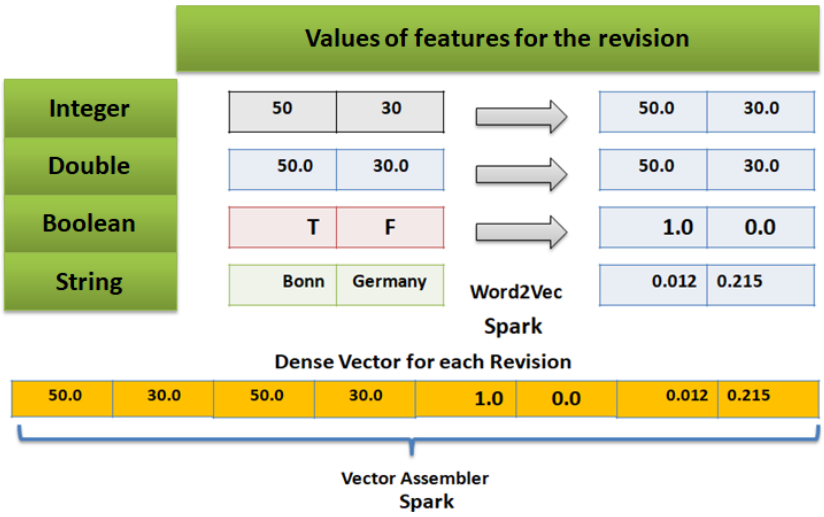

Figure 5.2: Converting the data type values to double values and creating a Dense Vector by Vector Assembler

5.4 Classifiers Module

In the Bonn-Berry approach, we use classifiers for training Wikidata as a dataset and apply a binary classification evaluator to get the area under the ROC curve. Hence ROC value measures the performance of the approach in detecting the vandalism in Wikidata. If the ROC value is near one, that means the performance of the approach is perfect. We will illustrate these concepts in great details in the evaluation chapter.

For training the vandalism detection module, we negotiate five classifiers to choose the best result: Random Forest classifier, Decision Tree classifier,

Logistic Regression classifier, Gradient-boosted Tree classifier and Multilayer Perceptron classifier. On the one hand, **Figure 5.3** illustrates how we implement the machine learning pipeline by a Spark. On the other hand, **Figure 5.4** reviews in more details machine learning cross-validation process for training and testing the data [**43**].

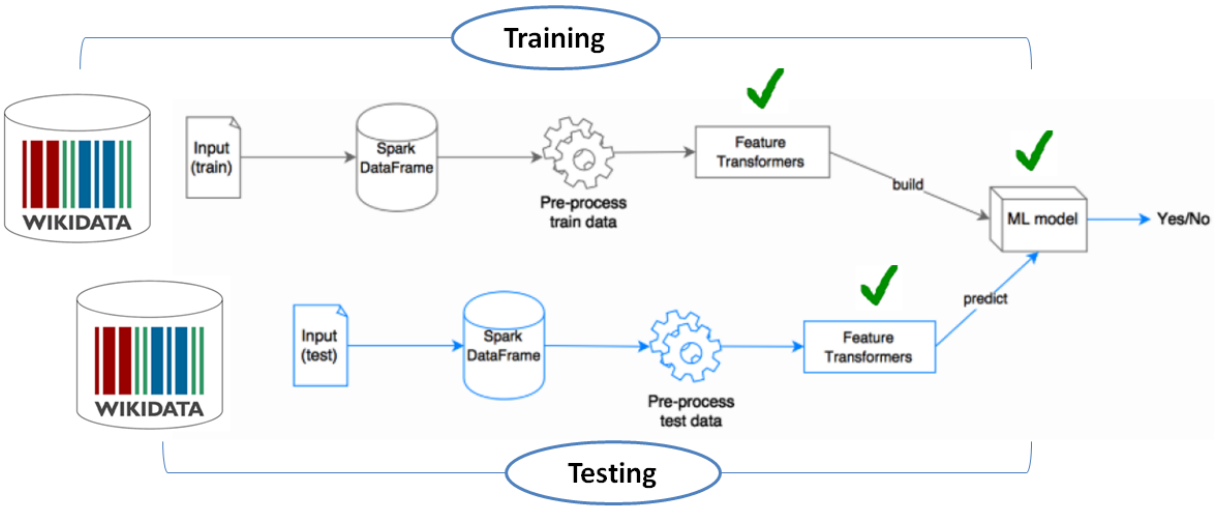

Figure 5.3: Machine learning pipeline by Apache Spark

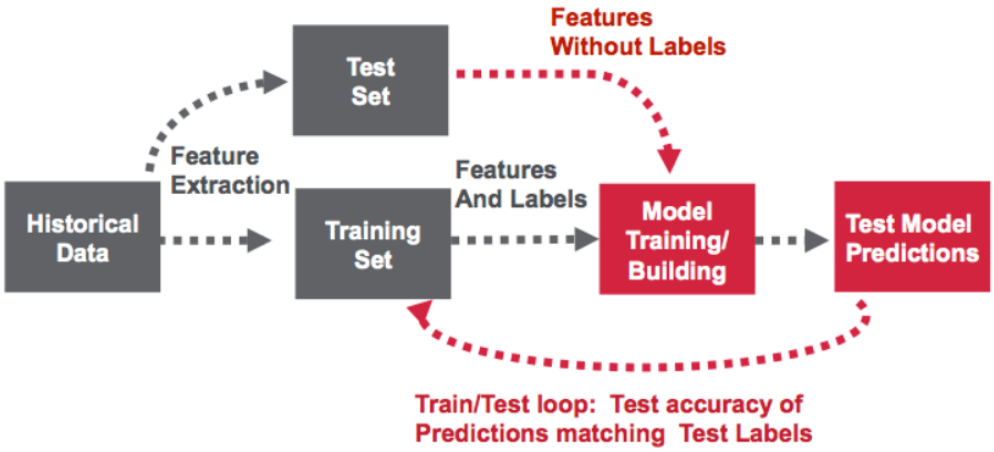

Figure 5.4: Machine learning cross-validation process

Different steps have summarized more recent strategies against classifiers. In other words, despite its common use in Machine Learning, classifiers are

implemented in a different discipline. The following steps categorized the most straightforward steps to implement the classifiers in our case study:

1. Loading and parsing the dataset.
2. Indexing labels, adding meta-data to the label column and fitting on the whole data-set to include all labels in the index.
3. Automatically identify categorical features, and index them.
4. Splitting the data-set into two parts, training and test datasets.
5. Based on classifier type, we train a classifier model.
6. Convert indexed labels back to original labels.
7. Chain indexers in a Pipeline then train model. This also runs the indexers.
8. Making predictions and applying the binary classification evaluator to produce the ROC value for measuring the performance of the approach.

Chapter 6

Evaluation

6.1 Experimental Setup

We considered two types of experiments, one runs on the local environment and the second runs on the Cluster environment. The local environment is used for experiments which are tested by 16 GB Memory, 4 cores, Hadoop ecosystem 2.7 and Apache spark 2.11. The Cluster environment are used for the final experiments which are tested by 240 GB Memory, 64 cores for each node where we have 3 nodes (central university server), Hadoop ecosystem 2.7 and Apache spark 2.11.

6.1.1 Spark Tuning

Although differences of opinion still exist, there appears to be some agreement that executors, cores and memory for a Spark application running cannot be managed without taking into account the size of data and properties of the machine. There are three approaches for tuning executives:

(1) Tiny executors. One executor per core will be used.
(2) Fat executors. One executor per node will be used.
(3) The balance between fat vs tiny. Based on resources and size of the data.

In the following, we can compare these three cases and how we calculate the number of executors, executor cores and executor memory as an example.

Case 1: Tiny executors. In this approach, we will assign **one executor per core**. Number-executors is:
= total-cores-in-cluster.

= num-cores-per-node * total-nodes-in-cluster = 16 x 10 = 160.
executor-cores = 1 (one executor per core).
executor-memory = amount of memory per executor = mem-per-node/num-executors-per-node = 64 GB/16 = 4 GB.

Case 2: Fat executors. In this approach, we will assign **one executor per node**. Number-executors is:
= total-nodes-in-cluster = 10.
executor-cores = one executor per node means all the cores of the node are assigned to one executor.
= total-cores-in-a-node = 16.
executor-memory = amount of memory per executor.
= memory-per-node/number-executors-per-node.
= 64 GB/1 = 64 GB.

Case3: Balance between Fat (vs) Tiny.
In this case, we assign 5 core per executors. Executor-cores = 5. Accordingly, we leave one core per node for Hadoop/Yarn daemons and that means number cores available per node = 16 - 1 = 15. Hence the total number of available cores in cluster = 15 x 10 = 150. Number of available executors = (total cores/number-cores-per-executor) = 150/5 = 30. Also we leave one executor for Application Manager and number-executors = 29. Number of executors per node = 30/10 = 3, Memory per executor = 64 GB/3 = 21 GB. Counting off heap overhead = 7.

6.2 Data sets

Wikidata Standard XML Dataset. Wikidata is an open-source, free knowledge base which includes more than 65 million revisions are provided officially by Wikidata Vandalism Corpus 2016 [**41**]. It includes Wikidata item pages between 2012 and 2016. In current experiments for evaluation, we use a subset from the Wikidata item pages from 2015-2016 [**44**].

Geo–species RDF/XML Dataset. Geo-species is an RDF/XML data set which includes 168, 750 triples. It is provided officially by Peter J. DeVries, UW-Madison and published on W3C communities. It includes details about biological orders, families, species as well as species occurrence records and related data [**38**].

Charging-station TRIX(RDF Triple in XML) Dataset. Charg-

ing–the station is a data-set which includes more than 8000 triples related to addresses and postcodes. It is provided by "enel" which included a wide range of data-sets with different representation formats [**39**].

6.3 Data Parsing Evaluation

6.3.1 Correctness of the Data Parsing

(1) Correctness of the RDF Parsing Module. RDF/XML have been used for representing a Geo species dataset [**38**]. As a result of using Apache Jena parser as a sequential parser, we get precisely 168, 750 triples compared with running the new distributed parser which produces 168,750 triples.

As a result of the matching in results of these experiments, we conclude the correctness of the new distributed RDF/XML parser is 100% because it produces the same number of triples which are produced by Apache Jena.

Charging-station dataset has been represented by TRIX [**39**]. Based on the sequential parser, we get 8900 triples compared with running the new distributed TRIX parser, we get the same number of the triples, and that illustrates the correctness of the new distributed TRIX parser is 100 %. Table(6.1) outlines all the results of these experiments.

RDF Parser	
Apache Jena - Geo Space Dataset	**Bonn-Berry Parser**
168,750 Triples	168,750 Triples
RDF4j - Charging-station Dataset	**Bonn-Berry Parser**
8,900 Triples	8,900 Triples

Table 6.1: RDF Parser

Bases on table(6.1), the **Figure 6.1** represents the results which are the same in both cases. Hence, we can conclude the correctness of the RDF parsing module in the Bonn-Berry approach matches the correctness of the parsing by Apache Jena.

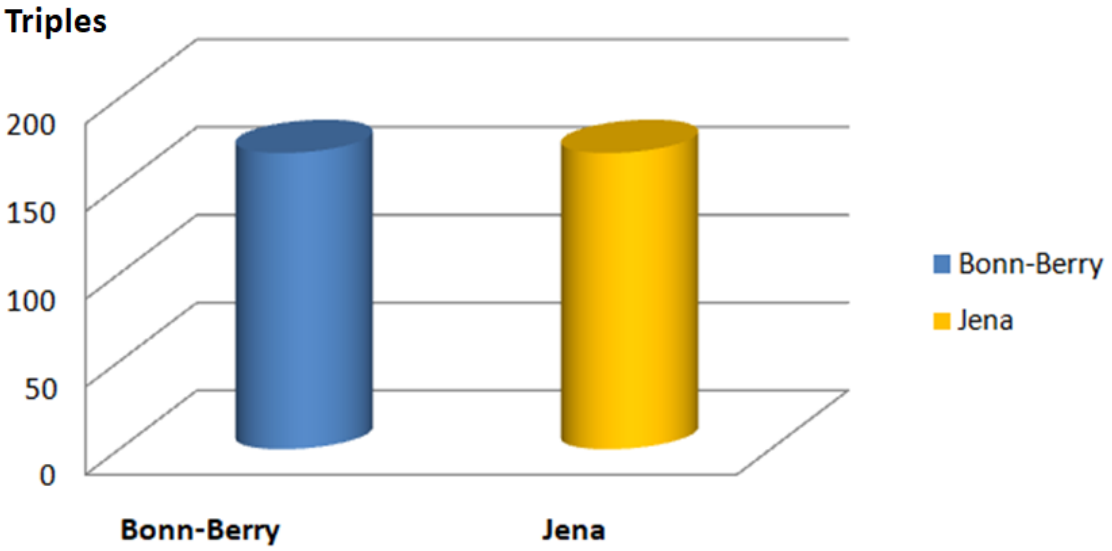

Figure 6.1: Comparing the correctness of Bonn-Berry RDF Parsing Module and Apache Jena

(2) Correctness of the standard XML Parser Module. Wikidata relies on the standard XML representation, and all of the revisions in Wikidata have a unique ID number which cannot be repeated. We can get a number of the revisions in the dataset which is used in the evaluation by Heindorf parsing module which is a part of Heindorf vandalism detection approach [**2**]. After running the parsing module in the Bonn-Berry approach, we get the same number of the revisions which is provided by the Heindorf parsing module [**2**]. In this pioneering experiment of correctness, the standard XML parser module's correctness in the Bonn-Berry approach is 100 % because it produces the same number of the revisions which is produced by Heindorf. Table (6.2) reviews number of the revisions in each case. Hence, **Figure 6.2** shows how the results are the same in both cases.

Standard XML Parser -Wikidata	
Heindorf Parser	**Bonn-Berry Parser**
270,104,943 Revisions	270,104,943 Revisions

Table 6.2: Standard XML Parser -Wikidata

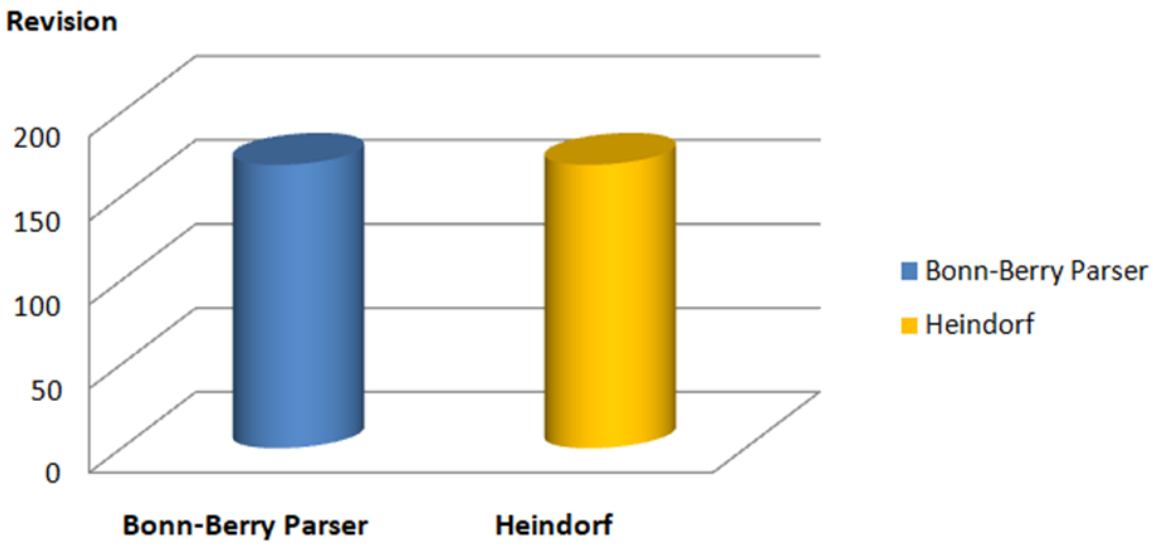

Figure 6.2: Comparing the correctness of Bonn-Berry standard XML Parsing Module and Heindorf parser

6.3.2 Scalability of the Data Parsing

Scalability of the data parsing module reflects the runtime which will be needed for running the tasks in this module. Range partitioning plays a vital role in improving the scalability.

In the first case, we illustrate the scalability without using Range Partitioning, then we compare the same experiments with the second case where we use a Range Partitioning technique.

Case 1: Hadoop and Spark without using Range Partitioning.
An experimental demonstration of this effect was first carried out by the change of cores number. By increasing the cores number in Spark context, the runtime is decreased. The relation between a cores number and execution time has been widely investigated in this experiment.

(1) Scalability of the RDF Parser Module. Table(6.3) outlines the evaluation cases for the RDF parser on Geo–species RDF/XML dataset and Charging-station TRIX dataset. Obviously, by this experiment, with increasing the cores number in Spark context, the rumtime is decreased, hence in current case, the scalability is better than the case before increasing the cores number.

Case 1 - RDF Parser: *without* Range Partitioning	
RDF/XML RDF	
30 Cores	**60 Cores**
Time:2 minutes	**Time:1.1 minutes**
TRIX RDF	
30 Cores	**60 Cores**
Time:1.2 minutes	**Time:50 second**

Table 6.3: Scalability of the RDF Parser Module Without Range partitioning

(2) Scalability of the standard XML Parser Module. Regarding the size of the dataset, in current experiments for parsing Wikidata item pages, we use different sizes of the subsets from Wikidata (100 MB, 200 MB, 500 MB, 1 GB, 1.8 GB, 5 GB, 10 GB) with taking into account the reliability and availability of the server where we run the experiments. Although we tested all of Wikidata subsets with sizes were mentioned before, we found it is more simple to explain the experiment related to the scalability based on using sub-set from Wikidata 2015-2016. Table(6.4) shows results of evaluation. The performance of the standard XML parser is improved by increasing the cores number in Spark context.

Case 1 - Standard XML Parser: *without* Range Partitioning	
30 Cores	**60 Cores**
Time: 55 minutes	**Time:37 minutes**

Table 6.4: Scalability of the standard XML Parser Module Without Range partitioning

Case 2: Hadoop and Spark with using Range Partitioning.

(1) Scalability of the RDF Parser Module. In this section, we illustrate the evaluation cases for the RDF parser which includes the RDF/XML and TRIX parseres. Hence, the table(6.5) provides a clear view about results of evaluation for both cases RDF/XML and TRIX on Geo–species RDF/XML

dataset and Charging-station TRIX dataset. Using Range partitioning technique in this experiment improved the performance apparently.

Case 2 - RDF Parser:*with* Range Partitioning	
RDF/XML RDF	
30 Cores	**60 Cores**
Time:75 seconds	**Time:40 seconds**
TRIX RDF	
30 Cores	**60 Cores**
Time:65 seconds	**Time:30 seconds**

Table 6.5: Scalability of RDF Parser Module With Range partitioning

(2) Scalability of the standard XML Parser Module. Regarding the size of the dataset, to test the parsing module, we use different sizes of subsets from Wikidata (100 MB, 200 MB, 500 MB, 1 GB, 1.8 GB, 5 GB, 10 GB) with taking into account the reliability and availability of the server where we run the experiments. Although we tested all of Wikidata subsets with sizes were mentioned before, we found it is more simple to explain the experiment related to the scalability based on using sub-set from Wikidata 2015-2016 . Table(6.6) shows the results of evaluation in this context.

The performance of the standard XML parser is improved by increasing the cores number in Spark context. By comparing case 1 (without using Range Partitioning) with the case 2 (with using Range Partitioning), we can note how the performance in case 2 is better than the performance in case 1 as a result of using Range partitioning.

Case 2 - standard XML Parser:*with* Range Partitioning	
30 Cores	**60 Cores**
Time: 30 minutes	**Time:18 minutes**

Table 6.6: Scalability of the standard XML Parser Module With Range partitioning

6.4 Vandalism Detection Evaluation

6.4.1 Accuracy of the Vandalism Detection Module

In this section, we measure the accuracy of the vandalism detection module depending on the Area Under an **(ROC)** Curve. The new vandalism detection module in the **Bonn-Berry** approach achieves the highest ROC score comparing with the previous approaches. As a result of the Wikidata being too big (in total more than 1TB), the researcher in the previous approaches could not use all of the data due to their system's inability to handle it. In this context, the datasets which are used for evaluation the accuracy in current experiments can be categorized under three cases: First, Wikidata 2015-2016 subset **(e.g. Buffalo-Berry and Conker-Berry [4, 5])**. Second, Subset based on the positive and negative revisions **(e.g. Logan-Berry [6])**. Third, Wikidata 2016 subset **(e.g. Honey-Berry [7])**. Tables (6.7), (6.8) and (6.9) illustrate the accuracy values for previous sequential vandalism detection approaches, compared with the Bonn-Berry approach which uses the same experiment's data. All of the label data in truth file which is provided by the WSDM corpus constructor are used [**44**].

Approach	**Case 1: Subset/WD(2015-2016)**	**Label**	**ROC**
Bonn-Berry	*Subset 2015 and 2016	All	0.991- 0.993
Buffalo-Berry[**4**]	*Subset 2015 and 2016	All	0.937
Conker-Berry[**5**]	*Subset 2015 and 2016	All	0.937-0.960

Table 6.7: Comparing ROC Values for subset Wikidata 2015-2016

Approach	**Case 2: Subset/WD with positive and negative revisions**	**Label**	**ROC**
Bonn-Berry	*Subset(174,427)positive,(4,360,675) negative	All	0.991-0.992-0.993
Logan-Berry[**6**]	*Subset(174,427)positive,(4,360,675) negative	All	0.920-0.986

Table 6.8: Comparing ROC values for subset includes specific number of positive and negative revisions

Approach	Case 3: Subset/WD(2016)	Label	ROC
Bonn-Berry	*Subset Jun and Feb 2016	All	0.991- 0.993
Honey-Berry[7]	*Subset Jun and Feb 2016	All	0.9441-0.959

Table 6.9: Comparing ROC Values for subset Wikidata 2016

By comparing among the all of the cases **(1, 2, 3)** in the tables (6.7), (6.8) and (6.9) , the Bonn-Berry approach achieves the best ROC value which reflects the accuracy of the vandalism detection module in the Bonn-Berry approach. In this context, we should remember that the previous approaches focused on improving the results related to accuracy compared with the Bonn-Berry approach which aims to improve the accuracy and scalability. The Buffalo-Berry [**4**] was implemented by research team in the Austral University in Argentina in 2017, they achieved ROC value 0.937. The return ROC rates were achieved by Conker-Berry approach [**5**] which was implemented by Searchmetrics GmbH in 2017 in Germany with values of 0.937 and 0.960. **Figure 6.3** reviews the comparison for case 1 where the Bonn-Berry versions achieve the best results.

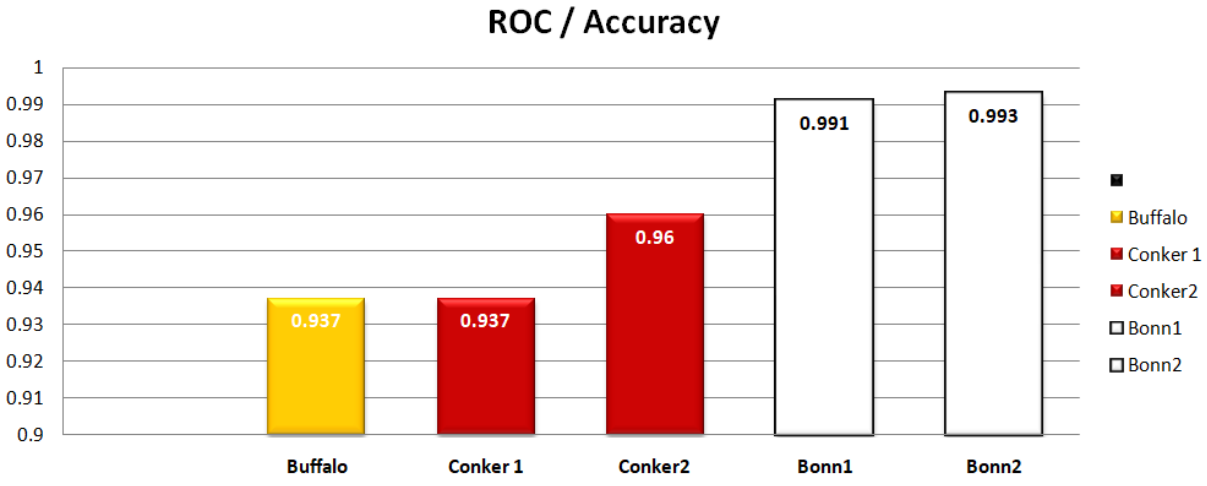

Figure 6.3: Comparison ROC of the Bonn-Berry's experiment with previous approaches-Case1

Also in 2017 in USA, the Longan-Berry approach [**6**] was implemented by research team in Illinois university and achieved ROC values 0.920 and 0.986

and **Figure 6.4** reviews the comparison for case 2 where the Bonn-Berry versions also achieves the best results.

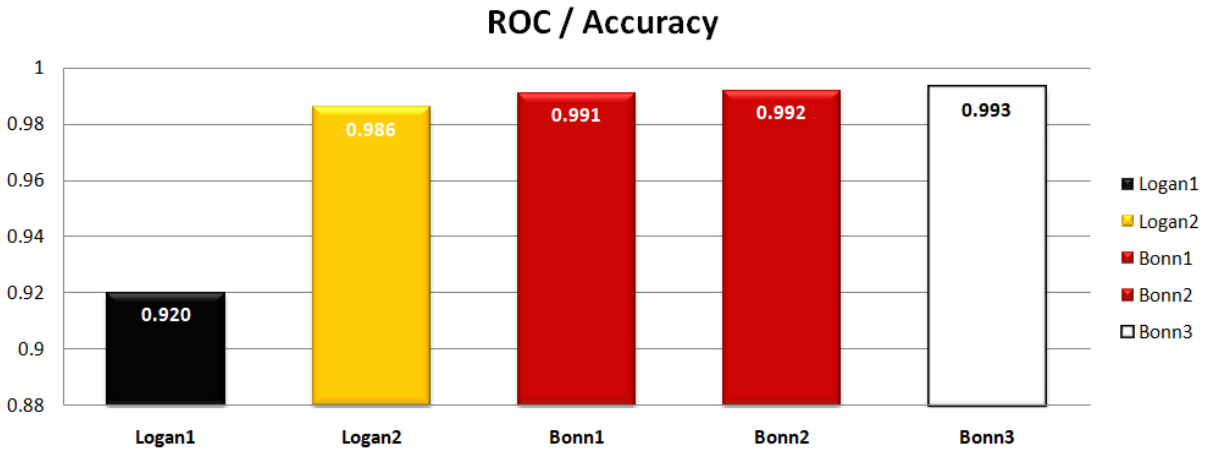

Figure 6.4: Comparison ROC of the Bonn-Berry's experiment with previous approaches-Case2

Yahoo corporation in Japan took part in this challenge. In 2017 in Japan, they produced ROC values 0.944 and 0.959 [**7**]. **Figure 6.5** reviews the comparison for case 3, where the Bonn-Berry versions produce the best results.

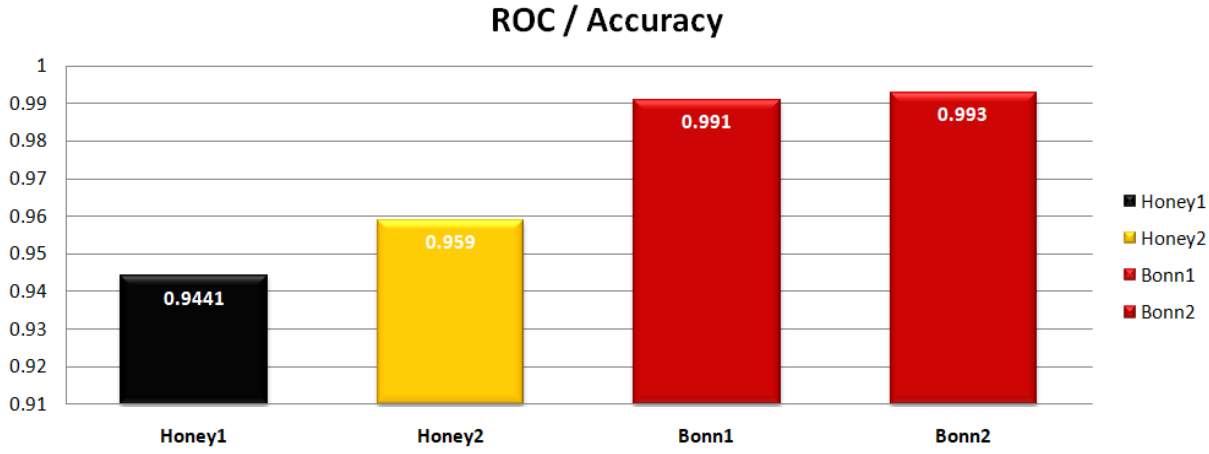

Figure 6.5: Comparison ROC of the Bonn-Berry's experiment with previous approaches-Case3

In 2018 in Germany, the research team in Smart Data Analytic Group at University of Bonn achieved the ROC values of 0.991 and 0.993 for sub-

sets Wikidata 2015-2016 and produced the values 0.991, 0.992 and 0.993 for subset Wikidata with specific number of positive and negative revisions in addition to produce the values 0.991 and 0.993 for subsets Wikidata 2016.

The classifier which achieved these results in the Bonn-Berry approach is the Random Forest classifier. The reason behind using the Random Forest classifier is that it does not have a problem with over-fitting. Furthermore, it is easy to set its parameters, and accuracy is generated automatically. We can learn from existing different ROC values in the previous comparisons (Figure 6.3, Figure 6.4, Figure 6.5) that the features and classifiers play vital roles in improving the results, hence giving more versions from vandalism detection approaches. On the one hand, to review the ROC value for each feature category individually in the Bonn-Berry approach, table(6.10) reviews a comparison of the ROC value for each features category.

To review the ROC value for each feature category individually in the Bonn-Berry approach, table(6.10) reviews a comparison of the ROC value for each features category. Based on a different overview, to review the ROC value for each feature category cumulatively, table(6.11) outlines the comparison among features categories and ROC values which were produced.

Based on these results, the cases demonstrate that the revision features play a vital role in improving the results. All of the features together achieved the satisfactory results but each one individually does not have the same result.

ROC value for each feature category individually-WD 2015-2016	
Feature Category	**ROC**
Characters Features Level	0.875
Words Features Level	0.864
Sentences Features Level	0.815
Statements Features Level	0.869
User Features Level	0.874
Item Features Level	0.897
Revision Features Level	0.930

Table 6.10: ROC value for each feature category individually

ROC value for feature categories cumulatively-WD 2015-2016	
Feature Category	**ROC**
Characters/Word Features Level	0.897
Characters/Word/Sentences Features Level	0.843
Characters/Word/Sentences/Statement Features Level	0.834
Characters/Word/Sentences/Statement/User Features Level	0.990
Characters/Word/Sentences/Statement/User/Item Features Level	0.991
Characters/Word/Sentences/Statement/User/Item/Revision Features Level	0.993

Table 6.11: ROC value for each feature category cumulatively

Figure 6.6 shows the ROC values for all features accumulatively. We can see how the feature's categories can give different ROC values because each category has a particular effect and rank.

The Character, Word, User, Item, and Revision features have a positive effect because its rank is very high in the training model, compared with the Statement and Sentence features which have a low effect and decreases the ROC value as a result of **over-fitting**.

6.4.2 Scalability of the Vandalism Detection

Case 1: Hadoop and Spark without Range Partitioning.

Input of the Vandalism Detection module is output of the standard XML parser module which gives a structured data from a semi-structured data (Wikidata in current case study). Essentially, table(6.12) clarifies values of the runtime for different cores number and illustrates role of the cores number in improving the scalability in vandalism detection module.

Vandalism Detection : *without* Range Partitioning	
30 Cores	**60 Cores**
Time: 125 minutes	**Time: 89 minutes**

Table 6.12: Scalability of Vandalism Detection Module Without Range partitioning

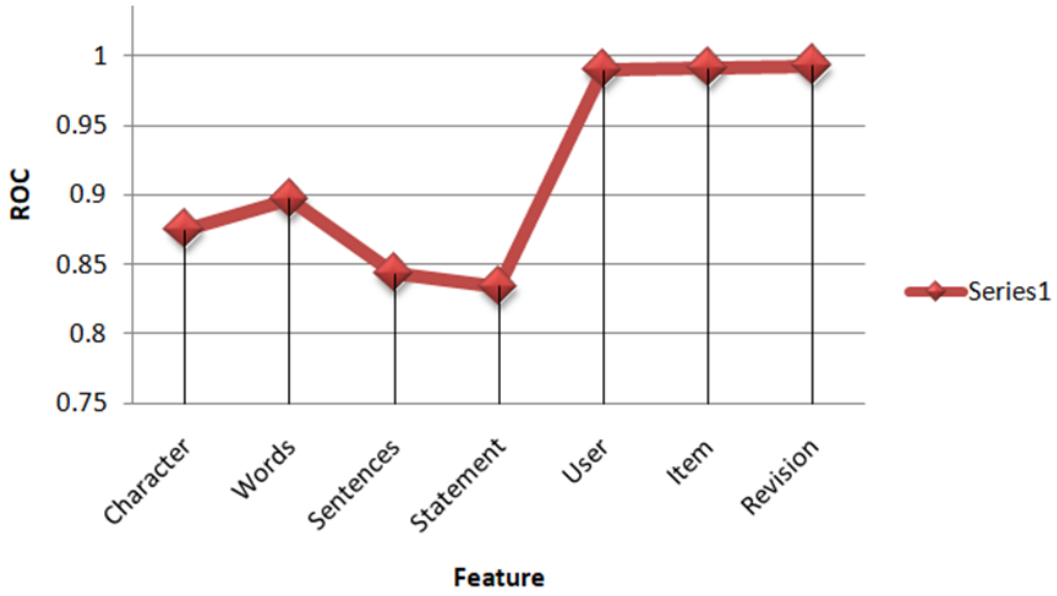

Figure 6.6: ROC values of the Bonn-Berry approach with all features accumulatively

Case 2: Hadoop and Spark with Range Partitioning.

The new considerations have been shown in this experiments due to use of the Range Partitioning which contributes in improving the performance. The runtime in this case is better comparing with case 1 where we do not use Range partitioning. Table(6.13) reviews the results of using different cores number and illustrates how the range partitioning in case 2 improves the runtime comparing with case 1.

Figure 6.7 shows role of the cores number in reducing the execution time. For instance, if we have 1.8 GB dataset with 30 cores, the runtime is approximately 87 minutes. Compared with case of using 60 cores, the runtime is approximately decreased to 50 minutes for the same size of dataset. Accordingly, we can learn the role of Range partitioning in reducing the runtime for any application is implemented by Apache Spark.

Vandalism Detection: *with* Range Partitioning	
30 Cores	**60 Cores**
Time: 87 minutes	**Time: 50 minutes**

Table 6.13: Scalability of Vandalism Detection Module Without Range partitioning

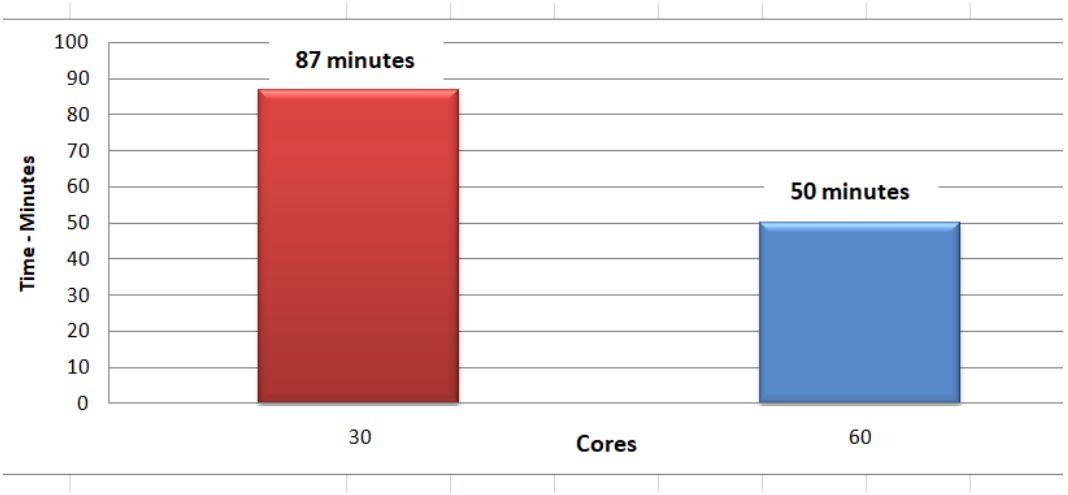

Figure 6.7: Comparison the run-time and cores number based on specific size of dataset

6.5 Enriching Data With External Sources Limitations

Enriching Data With External Sources. User features which were used in previous vandalism detection approaches depended on using an official geographical data-set for enrichment of the training data-set and deriving more important user features, such as user country-code, user continent-code, user timezone, user region-code, user city-name and user county-name.

In the new vandalism detection approach, we could not use an official geographical data set as result of budgetary constraints. We depended solely on the enrichment file which was provided by Wikidata vandalism corpus 2016 and included a poor geographical information. This issue decreased rank of the user features in measuring the performance of the Bonn-Berry approach in detecting the vandalism.

Chapter 7

Conclusions and Future Work

7.1 Summary

In response to the growing needs to support extracting structured data from unstructured and semi-structured data, in addition to increasing the importance of the data quality in context of the large-scale knowledge bases, we proposed the Bonn-Berry approach "Distributed data parsing and Vandalism detection on large-scale knowledge bases using Apache Spark and Hadoop Ecosystem". In this case study, the contributions can be illustrated in two core parts:

1. Distributed data parsing module. This module includes three distributed parsers: RDF/XML distributed parser, TRIX distributed parser and Standard XML distributed parser.

2. Distributed vandalism detection.

The Bonn-Berry approach proposed new distributed data parsing and vandalism detection modules on the large-scale knowledge base. We shows this idea could work in a modern distributed data storage and distributed data computing environment, such as the Hadoop ecosystem and Apache Spark. Evaluation of current case study investigated two parameters, **accuracy and scalability**. Accordingly, the new distributed vandalism detection module achieved the highest results of accuracy on the large-scale subset of Wiki-Data which included revisions from 2015-2016. Moreover, the data parsing module achieved a perfect correctness in the results and better performance. Furthermore, the experimental results proved the effectiveness and efficiency of the proposed approach.

7.2 Future Work

By way of illustration, the Bonn-Berry approach showed how the features extracting, representing and engineering played a crucial role in achieving high results in the vandalism detection process. Hence, the effectiveness of the generation features technique constructs the backbone for an accurate vandalism detection process. Also, studying the relations among the revisions for each item page improves the predictive power of current algorithm significantly.

With the massive amount of data and importance of size of the data for training the module in neural networks (NN) which may give the chance to improve the results of the Bonn-Berry approach. Specifically, convolutional or recurrent neural networks. However, by using the neural network, we must follow a new strategy for creating the features which should be a compatible features to neural network module.

Bibliography

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," 2017.

[2] S. Heindorf, M. Potthast, B. Stein, and G. EngelsErmilov, "Vandalism detection in wikidata," Paderborn University and Bauhaus-Universität Weimar, 2016.

[3] A. Spark, "Apache spark offcial website," 2018.

[4] R. Crescenzi, M. Fernandez, F. A. G. Calabria, P. Albani, D. Tauziet, A. Baravalle, and A. S. D. A. University), "A production oriented approach for vandalism detection in wikidata - the buffaloberry vandalism detector," 2017.

[5] A. G. S. GmbH), "Large-scale vandalism detection with linear classifiers," 2017.

[6] Q. Zhu, H. Ng, L. Liu, Z. Ji, B. Jiang, J. Shen, and H. G. U. of Illinois Urbana-Champaign), "Wikidata vandalism detection - the loganberry vandalism detector," 2017.

[7] T. Yamazaki, M. Sasaki, N. Murakami, T. Makabe, and H. I. Y. J. Corporation), "Computer science - information retrieval ensemble models for detecting wikidata vandalism with stacking - team honeyberry vandalism detector," 2017.

[8] E. RDF4J, "The eclipse rdf4j framework - java framework for processing rdf data," 2018.

[9] T. A. S. Foundation, "Apache jena - a free and open source java framework for building semantic web and linked data applications.," 2018.

[10] P.-N. Tan, M. Steinbach, and V. Kumar, "Classification and decision tree classifier introduction -introduction to data mining," 2017.

[11] J. Broekstra and A. Kampman, "Sesame: An architecture for storing and querying rdf data and schema information," 2017.

[12] S. Powers, "Solving problems with the resource description framework," 2017.

[13] N. S. Software, "Logistic regression," 2017.

[14] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," 2017.

[15] P. Geurts and G. Louppe, "Learning to rank with extremely randomized trees," 2017.

[16] H. Ramchoun, M. A. J. Idrissi, Y. Ghanou, and M. Ettaouil, "Multilayer perceptron: Architecture optimization and training," 2017.

[17] J. Lehmann, G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, I. Ermilov, S. Bin, N. Chakraborty, M. Saleem, A.-C. N. Ngonga, and H. Jabeen, "Distributed semantic analytics using the sansa stack," in *Proceedings of 16th International Semantic Web Conference - Resources Track (ISWC'2017)*, 2017.

[18] I. Ermilov, J. Lehmann, G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, S. Bin, N. Chakraborty, H. Petzka, M. Saleem, A.-C. N. Ngonga, and H. Jabeen, "The Tale of Sansa Spark," in *Proceedings of 16th International Semantic Web Conference, Poster & Demos*, 2017.

[19] J. J. Carroll and P. Stickler, "Trix : Rdf triples in xml," 2004.

[20] K. Boyd, , K. H. Eng, and C. D. Page, "Area under the precision-recall curve:point estimates and confidence intervals," 2017.

[21] D. H. Miller, "Big data analysis with scala and spark - coursera École polytechnique fédérale de lausanne," 2017.

[22] J. Y. Monteith, J. D. McGregor, and J. E. Ingram, "Hadoop and its evolving ecosystem," 2017.

[23] M. B. AhmedAnouar and A. Boudhir, "Improving online search process in the big data environment using apache spark," 2017.

[24] V. Viswanath and D. Engineer, "Rdd structure - different technologies in spark data structure," 2017.

[25] J. Laskowski and I. C. passionate about ApacheSpark, "Mastering apache spark," 2017.

[26] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark sql: Relational data processing in spark," 2017.

[27] X. Rong, "word2vec parameter learning explained," 2016.

[28] C. Liu and H. Chen, "A hash partition strategy for distributed query processing," 2017.

[29] K. Psarakis, "Outlier detection using spark streaming," 2017.

[30] L. Breiman, "Random forests," Statistics Department University of California 2001.

[31] T. Segaran, "Classification and decision tree classifier introduction - programming collective intelligence," 2017.

[32] P. Thorwart, "Decision trees cores," trainer and system designer 2017.

[33] N. S. S. NCSS.com, "Logistic regression," Chapter321 - 2017.

[34] T. O. COMPUTING, "A short introduction – logistic regression algorithm," Chapter321 - 2017.

[35] A. Olinsky, K. Kennedy, and B. B. Kennedy, "Assessing gradient boosting in the reduction of misclassification error in the prediction of success for actuarial majors," 2017.

[36] G. Al-Naymat, M. AlkasassbehNosaiba, and A.-S. Sakr, "Classification of voip and non-voip traffic using machine learning approaches," November 2016Journal of Theoretical and Applied Information Technology 3192(2).

[37] A. S. O. Webpage, "Apache spark ml," 2018.

[38] W. DataSetRDFDumps, w. D. U. a. a. R. D. Linked Data Sets (i.e., and U. M. Peter J. DeVries, "Rdf/xml - geospecies knowledge base," 2017.

[39] Enel, "Enel open data - open sharing of corporate data," 2017.

[40] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of jaccard coefficient for keywords similarity," 2016.

[41] WikiDataGermany and A. Sponsor, "Wsdm cup 2017," 2017.

[42] W. Du, Z. Zhan, and S. University, "Building decision tree classifier on private data," 2017.

[43] MapRTechnologies, "Predicting breast cancer using apache spark machine learning logistic regression," 2017.

[44] W. Official and W. 2016, "Wiki data model/ wsdm 2016," 2016.