Faculty of Science, Technology and Communication

Advanced Content Analysis for Web Pages in Apache Spark

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Information and Computer Sciences

Author:
Dioni Rebonato Endringer

Student ID:
0170168801

Supervisor:
Prof. Dr. Martin Theobald

Reviewer:
Prof. Dr. Christopher Schommer

Advisor:
MSc. Paul Joseph Yves Meder

May 2019

Abstract

The term Content Analysis, by itself, might be broader than we may wonder and its use may be applied in different areas to make multiple types of analyses from any kind of content's sources. Turning to a specific area, the Web, it is still broad, but now we have a context to work on. Further, the "Web Universe" grows in large scale time by time. Thus, how could it be "known" and "indexed" in order to make it easily accessible by the end-users? At this point, the essential idea is to analyse the content of web pages in an automatic way, in order to find "good data" and to discover common standards behind, and finally to produce tools which may gather information for future browsing, improving constantly the user-experience.

The main focus of this work is to introduce a process or procedure on how we analyse the content of a set of web pages, starting from the Web Content Mining process to build a dataset, until the Web Content Analysis process to find relevant information about those. Further, Machine Learning techniques are applied to build a best model, to classify and to evaluate our experiments. We introduce, as well, the issues we may face during the whole process. In addition, the use of Apache Spark, a very powerful framework for Big Data analysis, is present in the beginning, to extract features, until the end, to evaluate the achieved results. The key point is how to analyse and classify the data, setting the most relevant content areas in the Web Pages.

Declaration of Honor

I hereby declare on my honor that I am the sole author of the present thesis. I have conducted all work connected with the thesis on my own.

I only used those resources that are referenced in the work. All formulations and concepts adopted literally or in their essential content from printed, not printed or Internet sources have been cited according to the rules or academic work and identified by means of footnotes or other precise indications of source.

This thesis has not been presented to any other examination authority. The work is submitted in printed and electronic form.

Luxembourg, May 2019.

Dioni REBONATO ENDRINGER

Acknowledgments

First of all, I would like to express all my gratitude to my supervisor, Prof. Dr. Theobald, who advised me so many times since the beginning. As well, my advisor MSc. Paul Meder, who gave me valuable comments to help me writing this paper and improving the source-code. I really appreciate their efforts and time spent.

Further, the experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [35]– see `https://hpc.uni.lu`, it has been very useful to improve our algorithm.

Additionally, during this research and my studies, all my colleagues were very friendly and supportive. I could not forget them, I really appreciate their support and their help so many times. Next, my friends outside the University, many of them very far. All of this was very encouraging. All my gratitude to everyone who followed and supported me during this journey.

Lastly, but the most important in my life, my thanks to my family. It has been a long time in which I am far from them. However, in all those moments, either good or not, they were supporting me. I could not reach and even to start it without them. During the whole time here, they were there looking forward to hearing about me, my studies and all my experience. I am extremely grateful for all of those people mentioned hereby, as well as those who were not mentioned directly.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The constant growth of the Internet shows that we may do, nowadays, anything "connected" or "online". We might do any kind of research, in any area. Products are sold over the Internet, a market which tends to become more expressive with every year. Social Networks have been highlighted as one of the biggest time consuming activities, being daily used by millions and millions of people around the globe. Financial services such as, buying stocks or making transaction in bank accounts, in a bank which may be located in another country or continent. E-government, e-voting and several other "e-terms" were not possible some decades ago. Actually, they are so common in our lives, nowadays, that we do not realize how important and essential they are for a great part of the companies.

Further, a huge amount of information is stored around the world on web servers, over the Internet. Information is in our hands and we can access everything from our smart devices, in some cases, even in a smart watch. However, in this sense, to find some specific information, we have to search for it. Before the internet, we used "Summaries" or "Table of Contents" to find specific content in a book. Nowadays, with the Internet, the idea is the same, but with technology, algorithms, search engines and so on. We use the same idea, the same "index" of contents to find some web page, it is in that way how the search engines work.

The main idea of a search engine is to index the information from some web page and present it in the future to any user looking for something related. In our research, we are not going to cover this indexing process, but the idea is very close, in order to gather the information, as we have done, it was necessary. We set up an environment to collect every web page from some defined domain, to build our dataset. In order to work with these information as many times as we want, without having to access the website every time and most importantly not taking the risk of this information to have been changed in the meantime.

The first tool applied in this step is a web crawler, a tool which gather information, content and the whole structure of a determined set of web pages. We are going to talk briefly about the whole process later in this chapter and also deeply in a more specific section.

Besides building a dataset, which is going to be well explained in this work, we have also written some code to extract information from those files, the features to analyse. To do so, we have used the Apache Spark environment, a very concise and powerful framework for Big Data problems. However, we only used a small portion of data in the beginning, but increased the amount over time. A chapter is dedicated to cover the Apache Spark framework.

Finally, the last process was application of Machine Learning techniques, still in Spark environment. The framework is build on top of libraries, including a Machine Learning library, and is therefore specially designed and developed to deal with Big Data problems. Techniques broadly known and used, e.g cross-validation for model selection, classification algorithms and evaluations were applied in practice and we introduce it in the end.

1.1 Research Question

The main contribution of our research is to study a reasonable way to build a model, in order to analyse the content of a set of news web pages using the Apache Spark. It has to comprise the whole process since building the dataset to evaluate the results. Thus, we come up with those research questions:

1. **Is it possible to build a Machine Learning model to extract the information from some web pages and analyse them in order to classify the best content part in a web page?**
2. **If so, how could we evaluate the performance or the precision of the model?**
3. **Is it possible to apply the model to all types of web pages?**
4. **What could be the best collection of features for analysing and classifying the web content?**

1.2 Methodology

The presented work is a practical experiment, based on some theoretical researches, where we want to come up with the answers for previous research questions. The programming language used to develop and implement some techniques was Scala, which is based on Java, but very concise to work on Big Data problems. In addition, we applied Machine Learning techniques, once our main focus is how to analyse the information gathered from those web pages. The dataset was built from news web pages from 4 different domains and we previously defined manually the "good content part", in order to make possible to evaluate in the end.

However, before we move to our experiment, we are going to present some fundamentals for our studies, based on researches and techniques well known and studied

in the field as well, practical studies developed and published by the academia. In chapter 2, we present the fundamentals over all techniques which we are going to talk in our research work. Starting from the Web Mining process to Web Content Analysis process and Evaluation techniques, covering in the middle the Information Extraction. The focus is to analyse the features of those web pages and then to build a common model to classify the content.

Further, chapter 3 is dedicated to Apache Spark, which is not used by many researches in Web Content Analysis. We introduce in Apache Spark, the Machine Learning library, our focus in the presented study. Chapter 4 and 5 are where we explain our practical experiments and present our achieved results. Chapters 2 and 4 have similar structures, once the steps follow the same idea. However, they have different goals, the former is more theoretical, while the prior one is practical.

Finally we come up, in the conclusion, with a brief dedicated part for future work, to give an idea what might be done in the future following our study, or what we could have done in a different way, to guide possible interests in this field. This section concludes the work by presenting the results and answering the research questions.

1.3 Related Work

Many researchers have been studying and publishing works in the field of Web Content Mining, Web Content Analysis and so on. In addition, the Content Analysis area, may be wider as we think and it might express any kind of analysis. One of the most important paper has been published by Herring [22] in 2010, where the explanation about the area is very well covered, introducing since from the beginning to the Web Content Analysis. The author highlights in her paper, one of the first application of "pure" Content Analysis which has records, occurred in the 17th century, when the Church has examined amounts of early newspapers to analyse their contents in a systematic way [25]. Further, according also introduced by Herring [22], it has become a paradigm between 1940s and 1950s, where Berelson and Lazarsfeld [15, 14] uses the Content Analysis to analyse a mass of information about communication and journalism.

In 2010, Kim and Kuljis introduced an experimental study about Content Analysis of a set of Web Blogs from users in the United Kingdom and South Korea. The goal of their study was to investigate the users' behaviours in order to find patterns or preferences in the dataset [24]. For our motivation, they conclude that the application of Web Content Analysis is an easy process and which may produces very good results in the end, depending on the field of the analysis.

Prasad and Paepcke (in 2008) [31] and Ramisa (in 2017) [32], introduce in both papers Web Content Analysis focused on news articles web pages, which is our idea with the difference linked to the tools and frameworks we have used. While Prasad and Paepcke presents a tool to extract information from those news articles, without analysing them using, machine learning, Ramisa focuses more on the

Natural Language processing. Prasad work [31] is used as a base for our study to extract information, in DOM (Document Object Model) trees, which classify the HTML code of the web pages in nodes. The former approach was the same as Wu et al. present in their paper [36], they published a paper explaining how they achieved very good results using the DOM trees.

Furthermore, Chau and Chen [18], as well as Song et al. [34], have obtained promising results, which based they study in DOM tree from news web articles. Both introduce the machine learning techniques used to evaluate their model and we will relate it with our present study. In the end, both studies come up with the conclusion about the needs and motivations to study the subject, specially regarding the growth of the internet, which requires each time more precision of the web applications, such as the search engines.

Chapter 2

Theoretical Background

Mining data from any source may be an easy task, if the data is structured. However, it might also be difficult if the data is not structured, if it comes from many sources or in many different standards or even, does not have any standard. In addition, those sources might be different, with individual patterns, as well as, it may be stored locally in files, or on some Web Server. The main task of mining the data, is how to gather it and then, how to deal with the resulted data.

In this chapter we come up with the background for our study. Starting from the Mining process, to the Analysing part and ending with the evaluation techniques. Hereby, we relate all those theories, which will be linked in the chapter 4 with the experimental setup of our research.

2.1 Data Mining & Web Mining

Data Mining is the process of extracting unknown and potentially useful records like patterns, trends and so on, from stored historical data in order to gather information for business proposals, decision making or classification [17]. Regarding the source of these records (data), we may highlight the Data Mining process as a general idea. Hence, the Web Mining process, simply may be a subdivision of Data Mining, and where the data is collected from the Web. It might be obtained from Web Pages, Web Services, Web logs, as well as, Web Traffic and many other data sources.

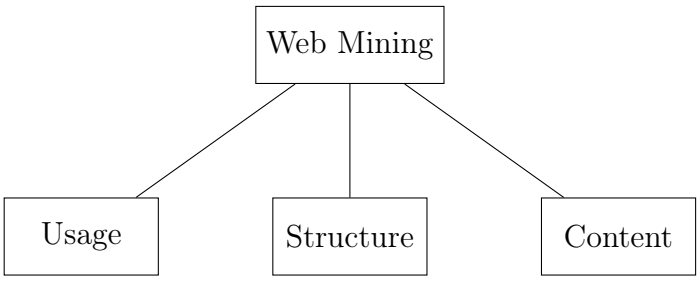


Figure 2.1: Web Mining Categories

The use of Web Mining applications is growing exponentially with each day. Following the idea introduced in [17] but adopting more recent approaches [28, 19], we introduce the Web Mining approaches subdivided in three categories: Web Usage Mining, Web Structure Mining and Web Content Mining, as shown in the Figure 2.1.

2.1.1 Web Usage Mining

Since it focus on the usage of the Web, the main idea behind Web Usage Mining is to improve the user experience when browsing over some Web Page. It analyses the Web log files to discover patterns for some specific user. Some examples of the extracted information might include: page clicks order, time spent in each page and so on [17, 19].

This technique is a very powerful strategy for Digital Marketing, which may offer advertisements for specific groups of potential customers, with less probability to fail and to waste time and money, due to the user-experience. Additional techniques in this field, as cited in [19], are: Data Collection, Data Pre-processing, Data Clustering and Pattern Discovery.

2.1.2 Web Structure Mining

This technique focuses on the structure of the Web Pages, where the idea is to improve the knowledge about the links among several Web Pages [19]. Two of the most popular algorithms for Web Structure Mining are PageRank [20](which is the base of Google Search Engine) and HITS [17]. PageRank ranks the popularity of web pages according the number of incoming links. It forms the base for the search engine, which is the principal tool for browsing nowadays, due to the huge amount of available content in the World Wide Web.

2.1.3 Web Content Mining

Compared to the previously mentioned techniques, Web Content Mining is a wider category, in the sense of opportunities for analysis. It may mine all the content of a web page, e.g, links, pictures, text, etc. Some authors [28] may classify this category into four different categories, depending on the type of mined data.

Unstructured Data Mining: The data may come in a completely unstructured form and it has to be structured in order to be analysed. Normally, those require the most part of the effort in structuring. Some applications related to this process include: Information Extraction, Summarization, Categorization, Clustering, etc [28].

Semi-Structured Data Mining: To give a better understanding of this approach, we give as an example the comment's area of a web page, where the users might write their own comments about a subject. It is said to be semi-structured in the sense that we may know in which area of the web page the comments are located, although, those comments are written by the users which may use any language to write it and they also might do it in an improperly way. That is why we may cite here applications like: Web Data Extraction Language and others related to language or object extraction [28].

Multimedia Data Mining: This technique is related which is well used in facial recognition nowadays. Multimedia Miner and Shot Boundary Detection are the main applications in this area [28].

Structured Data Mining: Usually those techniques are easier to be applied and require less effort than the previous ones, because it seeks for structured information which normally has some standards assigned [28]. Examples of this process are the mining in XML files or HTML files. The application of Page Content Mining or Web Crawler are the most well known.

The Web Crawler is the first tool which we practically used in the presented research, in order to gather information to build our dataset. That is the reason why we dedicate the next section to cover the application in more details.

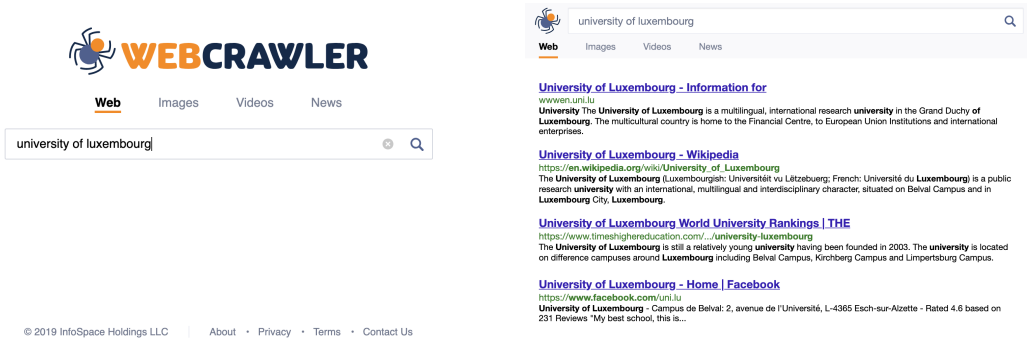
2.2 Web Crawling

In our work, Web Crawling is necessary since we do not have a dataset about news articles. Hence, we have to build one by gathering information directly from the news web site domains. We will explain the practice in more detail in chapter 4. Still important to mention, the idea is to build a dataset which is not under possible changes on the web pages.

According to Chattamvelli [17], a Web Crawler is classified by as part of a search engine. Although it might be used as a single task, the author relates this topic as a second part in a search engine, where the first is the query engine, the third is the indexer and finally, the forth is the user interface. The goal of this technique is, simply speaking, to save a copy of the web page for a later access [17]. So, it "takes a picture" of the current content of the web page. The term is also known as Web Spider or Web Robot [1].

In the context proposed by Chattamvelli[17], a search engine is an engine seeking for links and URLs in the websites related to the input query. For each link found, it browses over those to crawl the website and then, it indexes the results in a database to retrieve it back in the future, once the user seeks for it.

[1] `https://www.sciencedaily.com/terms/web_crawler.htm`



(a) User Interface

(b) Query Outcome

Figure 2.2: Search Engine example

In practice, once the user type some specific research topic into the user search bar (Figure 2.2a)[2], the search engine retrieves the crawled content from the indexes and then, it will show the results to the user (Figure 2.2b)[3].

On the other hand, the context of Web Crawler as single task has been applied in this work. Since we do not need to index the web pages to retrieve those in the future, we simply set up the environment to download the HTML files from those news web pages (our initial dataset). The experimental setup is detailed covered in chapter 4, in order to show how we have done, the issues that we faced and the achieved results.

The initial conception of this research is based on the paper of Kim and Kuljis [24]. In their experiment, they have downloaded the web pages to gather those "frozen in time" as they call it, in order to avoid the possible changes which could impact in our study.

2.2.1 Apache Nutch

Apache Nutch is an open source web crawler of Apache Software Foundation. It can run on single and local machine but it may also run over a Hadoop cluster. There are two different versions. The first is more suitable for batch processing, related to Hadoop data structures. The second, based on the prior version, is more complete and flexible. It allows to work with many other storage models solutions [8].

As Nutch is built on top of Apache Hadoop and Apache Lucene, it is easy to integrate Apache Solr[11], an indexing Apache tool. This means that, it would be possible to create a custom search engine and gather information from the specific domains, either over the Internet or in an Intranet[26].

Regarding the functionalities, the main idea is to crawl the web pages content. Clearly speaking, it allows to download of the content of web pages. In addition,

[2] `https://www.webcrawler.com`
[3] `https://www.webcrawler.com/serp?q=university+of+luxembourg`

the tool may search for links on specific pages/domains, as well, to index the links or contents for future access, as we mentioned before.

The Nutch's architecture, well documented in [23], is modular and divided into 4 steps:

- *Searcher*: This first step is responsible to find all the links and content in a given list of domains or web pages.
- *Indexer*: This step might be optional if the goal is to build a search engine or just to "download" the content from the prior step. It is responsible to index the data, in order to provide faster searches in the future.
- *Database*: It stores the content data as well as the links in the gathered information. In addition, it may store the indices.
- *Fetcher*: It is the step where is gathered more specific information from the content, exactly where the information is extracted, e.g links and contents.

2.3 Information Extraction

The second approach, still in the Web Mining field, is Information Extraction. While Information Retrieval seeks for important documents in a set of documents, Information Extraction seeks for relevant information inside those important documents, which are known to contain relevant information. However, this information may be unstructured [27].

As discussed in the previous sections, it may be classified as an unstructured data mining technique. However, in our case we prefer to establish it as a semi-structured technique, since we have a dataset, composed of HTML files and those are written in a specific programming language, which means that we might find the desired information easier than in completely unstructured data.

In addition, for this specific study, the extraction of information from HTML files follows the DOM (Document Object Model) Tree structure. The HTML language, very popular for Web Development, is structured as a tree, where the elements are encoded as nodes and leafs [36]. The figure 2.3 shows a DOM tree sample [16].

2.3.1 JSoup

JSoup is a HTML Parser suitable for extracting information from HTML codes easily and quickly. The most used approach is extracting data using the DOM-tree method. Besides, it also has methods to extract data from CSS sites. In addition, the library is able to manipulate the data in HTML codes[10].

Available to download in [10], the library comes in a "jar" file format. The code is written in Java, hence, it may be easier to develop due popularity of Java among the development community. Just like Nutch, the tool provides a "Fetcher"

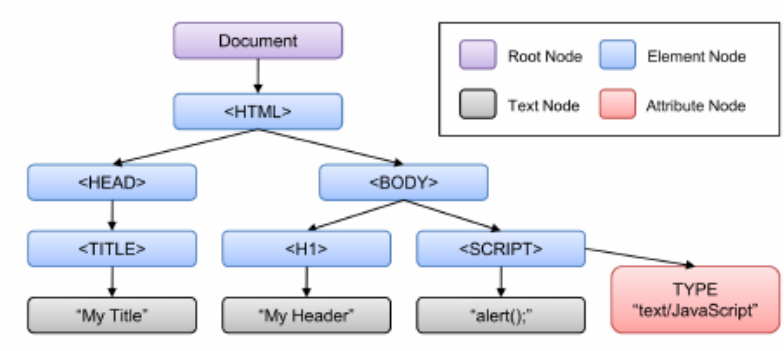


Figure 2.3: DOM tree sample

method, although it captures only the actual state of the specified web page. As we used Apache Nutch to crawl, we do not make use of this method.

Furthermore, JSoup may extract information from files. Thus, once it parses the file, it is possible to navigate though the DOM-tree structure, e.g figure 2.3, to gather all the desired or necessary information.

2.4 Knowledge Discovery

In this paper, we refer to Knowledge Discovery as all the discovered information which has been reached from extracted information, being good or not. If the extracted information from the dataset may produce in the end, good results, as knowledge.

In order to build a model, which techniques we should apply then? Hence, since we have the extracted information, then we have to analyse it. In this section, we discuss the Web Content Analysis of this gathered information from news web pages and which techniques we may use to make it run.

2.4.1 Web Content Analysis

Based in the conceptualization proposed by Herring[22], the figure 2.4 introduces the mostly used topics of Web Content Analysis. According to the author, the Image Analysis appears in one category, even it might be related to Features or Theme, because it might be interpreted by itself to be analyzed, being the only topic of the analyse.

On the other hand, the Theme Analysis is the most general topic and comprises the structure of the page itself. Following, the Link Analysis works on the structure and relations among web pages, e.g, the amount of links which redirects for external web pages or internal; or maybe, the amount of clicks in each link.

Further, the Language Analysis is the most popular, when it comes to Natural Language Processing. It is commonly used to study the semantics of the languages, in academic area. For industries, it may be used to capture the behaviour of the users, and then, to use those information for business purposes.

The study published in [24] is an example of Language Analysis, where, mainly, the behaviour of the users is the goal to be reached. They compare the content of blogs in South Korea and in the United Kingdom, with the idea to catch the cultural impact in the way how the users browser on those tools.

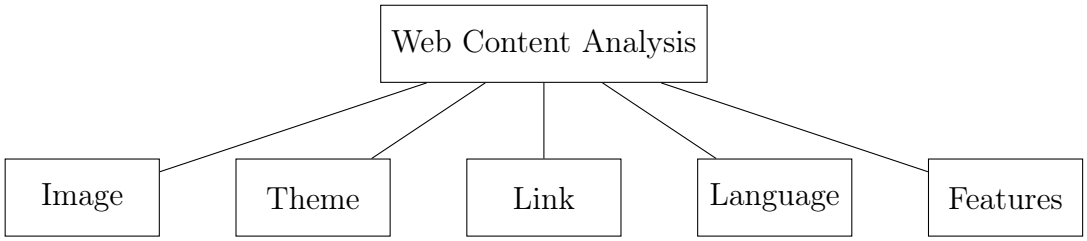


Figure 2.4: Web Content Analysis: Categories

Finally the last topic in figure 2.4, Features Analysis, may also be related to the previous mentioned study[24]. The Features could have been used in the prior case to be aware about some behaviour. For example, if in some blog, they publish so many pictures or not, or maybe about the amount of links, or still, the number of words and so on. This former is the topic of Web Content Analysis which we explore in this study.

2.5 Machine Learning

According to the very simple definition of Wikipedia[4]: *"Machine learning is the study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions. [...] Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task."*

In Machine Learning, the type of the techniques is often divided into two categories: supervised learning and unsupervised learning. The main difference is: while the unsupervised techniques group instances without pre-defined attributes, the supervised learning usually knows what kind of attributes are expected in the results [33].

Prediction models are often defined as supervised methods due to its characteristics knowing in advance the possible outcomes for a given task. Common techniques are Decision Trees. According to Rokach [33], the supervised approaches are commonly split between *Classification* and *Regression* models.

[4]Machine Learning `https://en.wikipedia.org/wiki/Machine_learning`

2.5.1 Decision Tree and Random Forest

Decision Trees are commonly used to predict some information using some previously trained data. Regarding the previous section, it is a supervised learning. It means that the results are expected in some way how. Examples might be: classification for medical issues, financial market, those where the "answers" are known previously, e.g, "good or bad", "diseased or not diseased" and so on.

The Decision Trees are also used to represent some classifier or regression model [33]. Classification is used to organize the data into distinct and previously defined class or category [17]. On the other hand, the regression uses information from the past to predict some information for future cases by probabilities.

Random Forests are, simply speaking, the ensemble of many Decision Trees, normally with the motivation to improve the results of the prior. The main idea is still the same, and it is used with the same proposals, although it might require more parameters.

For instance, once classifying some data is expected some type of outcome previously defined. The proposal of this work is the classification of some contents from web pages as "good" or "bad". Thus, based on previously trained data, a model is set and hence, it may predict if the given content is "good" or "bad".

2.5.2 Model Selection

The parameters to build a Machine Learning model are very important in order to avoid over-fitting in the model. In other words, we have to avoid unnecessary parameters, in order to achieve the best processing time. The task to select the best parameters is usually called Model Selection. The main idea is to compromise between fit the data and the complexity of the model [13].

One of the most popular technique in this field is k-fold Cross Validation[13]. If the amount of data is slightly enough, k-fold Cross Validation splits it into "k" parts, then it fits a model with one part and tests it with another[21].

The methodology works fitting the model with "k-1" parts and testing "kth" part. It repeats the test "k" times to ensure that all parts were tested. For each iteration, it evaluates the model, calculating, the prediction error for each part.

For Decision Trees, the class specifications are defined according the type of analysis and the type of data (we show in practice in chapter 4). In addition, some other parameters might tested using Model Selection the catch the best model.

The number of departments and the impurity are common parameters tested in k-fold Cross Validation, to define the best model. The first one is simply the number of layers which the decision tree can maximally grows. However, impurity is used to measure the homogeneity of the labels at the node [3].

2.5.3 Evaluation Techniques

Besides training the model in Machine Learning, the evaluation of the results is one of the most important step. However, the application of determined technique will depend of the type of "learning". Relating to this study, we consider "Binary Classification" learning. We say "binary", because we only have two possible outcomes for each test data classified, in this case: "good" or "bad". These two possible outcomes, might be known as "classes", we will see more in the practical experiments chapter.

Furthermore, the application of those techniques are related to the category of classified data. Thus, before to cover those techniques is necessary the understanding of this four categories [4]:

- **True Positive (TP):** When label and prediction are both positive;
- **True Negative (TN):** When label and prediction are both negative;
- **False Positive (FP):** When label is negative but prediction is positive;
- **False Negative (FN):** When label is positive but prediction is negative.

The understanding of the previous categories is necessary, once those are used in the computation of evaluation techniques. Regarding binary classifiers methods, the most common evaluation techniques are: Precision, Recall and F-measure.

Precision and Recall

The definition of Precision and Recall was first published in [30]. The authors contextualized the terms with retrieved and relevant documents, considering the previously mentioned four categories. Bringing to the context of our study, we may use the definition proposed in [29].

Precision represents the positive predicted cases where: TP (label and prediction are both positive) is divided by all positive predicted cases ($TP + FP$). On the other hand, Recall considers the positive labeled cases: TP (label and prediction are both positive) is divided by all positive labeled cases ($TP + FN$).

$$Precision = \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN}$$

F-measure

F-measure, also known as F-score or F1 score, is the harmonic mean of Precision and Recall (formula below).[5] In other words, F-measure is "some desired balance" between the previous methods [4].

$$F = 2 * \frac{Precision*Recall}{Precision+Recall}$$

[5]Wikipédia F-measure: `https://en.wikipedia.org/wiki/F1_score`

Chapter 3

Apache Spark

Spark is a project that has been initially developed in the academic area at the AMPLab Berkeley UC[1] in 2009 and became open source in early 2010. It grew exponentially into development community, which ensured it to move to the Apache Software Foundation in 2013. The main motivation comes with the amount of data we have to manage nowadays, as well the growth of the information searching to make business by companies. Recently, many companies collaborate with the project [1].

Apache Spark is considered as an unified engine for Big Data processing. The framework combines a wide range of applications to deal with Big Data processing, such as Machine Learning, Streaming, SQL and Graph Processing. The easy and efficient way of Spark is ensured by to be built on top of libraries linked to the prior applications [37]. The figure 3.1 depicts the four main Spark libraries [1].

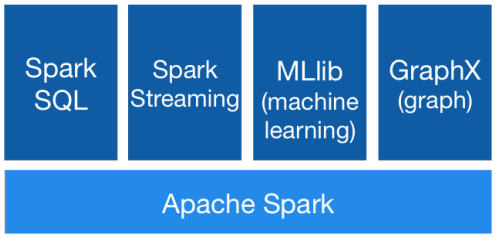


Figure 3.1: Spark Libraries

Apache Spark's efficiency comes from the fact that it already includes the most important libraries to deal with Big Data and Machine Learning problems in a single engine. It makes the framework very efficient in the sense that is not necessary to transfer data between frameworks to finish one task, e.g, combining SQL Data Frame and any Machine Learning algorithm available in Spark.

Additionally, Apark Spark provides the user with a data structure, called "RDD" (*Resilient Distributed Data*), which gives the framework an advantage over others. This technique auto-parallelizes the tasks among all available partitions equally, in order to increase the processing performance.

[1] AMPLab Berkeley UC: `https://amplab.cs.berkeley.edu/`

3.1 RDD - *Resilient Distributed Data*

Besides the parallelism, which will be more discussed in next sections, RDDs are automatically recover from failures, ensuring fault tolerance. According to [37], Spark tracks for each RDD the graph of transformations running. If necessary, it reruns to reconstruct any lost partition. The partition might be lost if any node downs from the cluster. Thus, Spark rebuilt the data reapplying the same transformation.

Additionally, RDDs may be persisted in memory for quick used, in order to increase the performance computation. The commands used to do it are: *"persist()"* and *"unpersist()"* [37, 6]. The persistence is able to be done in some ways, such as, memory and/or disk. Key-words, such as, "MEMORY_ONLY" and "MEMORY_AND_DISK", will cache the RDD's partition according specified [6].

3.1.1 Actions and Transformations

Regarding the usual operations, users make use of Spark's parallelism when writing in their code some transformations on data, such as *map* and *filter* [37]. According to Spark's documentation [6], RDDs works with two different operations: actions and transformations. The first one makes some action on data, e.g, calculations, count, max.

On the other hand, transformations usually create new data from a previous dataset. The most common transformation is mapping, however, Spark also supports others such as filter, join, grouping and others.

The table 3.1 contains some of the most common actions and transformations supported by Spark. The different actions and transformations are well documented in [6].

In addition, transformations in Spark are "lazy evaluated" [6]. This means that one transformation is just "remembered" by the application, which will take the information only if one action requires it. It enforces an efficient performance of the framework, avoiding unnecessary processing.

3.2 Libraries and Programming Languages

Following, we briefly discuss some characteristics of those four mentioned libraries [37], depicted in figure 3.1. However, MLlib Machine Learning will be discussed in more details in the upcoming sections due to the fact that it is linked to the presented research.

- **GraphX**: It provides graph computations to deal with problems in "node-edge-vertex" schema, making use of RDDs to distribute the data among the partitions [37].

Actions	
count	return the total number of records in the dataset
first	return the first record in the dataset
take(n)	return the n first records
collect	collect all records in the dataset

Transformations	
map(function)	it pass each element in the source though a $function$ and return a new dataset with the result
filter(function)	it filter according the $function$ and return a new dataset containing the result
union(otherDataset)	it merges the dataset with $otherDataset$ and return a new dataset with the result
intersection(otherDataset)	it takes the intersection of two dataset and return a new dataset with the result

Table 3.1: Examples of some actions and transformations on Spark.

- **Spark SQL**: SQL is the most common language to deal with relational databases. Further, those databases are largely used by companies to store structured data. By using Spark SQL, it is possible to deal with those data without making use of any other additional application.
  The library allows to run SQL queries directly within the code. Thus, it is possible to store those data on RDDs [37]. Hence, it increases the analysis performance and it further saves time when reusing the code.

- **Spark Streaming**: This library allows stream processing over Spark, splitting the input into small batches. It may be combined with batch queries. The initial input may be stored on RDDs, as well, it may run any MLlib algorithm [37].

- **MLlib Machine Learning**: This library provides around 50 Machine Learning algorithms, such as Decision Trees, Support Vector Machines, Tokenization, LDA (Latent Dirichlet Allocation), etc [37]. It impacts directly on the usability of the framework, combining very useful algorithms with its efficient parallel model.

All those libraries might be combined in order to reuse the code or increasing the performance. For instance, one task may first build an initial data by using SQL queries. Afterwards, it may apply some algorithms from MLlib to receive/gain/-gather some results.

Apache Spark supports 4 different programming languages, such as, Java, R, Python and Scala [37]. Scala has been chosen to deal with this project due to the fact to be very simple and concise. It handles big data problems in a very simple way, focusing on the RDDs approach to make it. Besides, even it might be considered a new language, comparing to the other options, it has a very complete documentation [3, 4, 5, 6].

3.3 Parallel Model of Spark

Spark is able to parallelize sequential data structures, such as arrays, by using the code as shown in listing 3.1 sample, in Scala programming language. If necessary, it is also possible to directly load data from external sources, such as *textfiles*, into an RDD. See listing 3.2. It will auto-parallelize the loading and the incoming transformations will be performed faster on this data [6].

```
val rddData = sc.parallelize(seqData)
```

Listing 3.1: Parallelizing Sequential Data Array

```
val rddData = sc.textFile("dataSeq.txt")
```

Listing 3.2: Parallelizing Sequential External Data

3.3.1 Running on Cluster Mode

Apache Spark is a framework well suitable to run parallel computations, it means that as many worker nodes are available as higher might be the performance. Even on a local machine, it is possible to run computations in parallel, if the machine has multiple cores and the source code is correctly written.

In order to use the parallelism on a local machine, Spark's parameters have to be configured accordingly (see listing 3.3). The "local[*]" parameter means that Spark runs in local mode using all available cores as workers, otherwise only "local" would run the framework in only one worker, without any parallelism.

```
spark-shell --master local[*]
```

Listing 3.3: Crawling Command Line

However, in order to improve the performance with Big Data, Apache Spark may also run in standalone cluster mode. For this study, we run our experiments either locally to find the best approaches, models, etc or using the facilities of HPC (High Performance Computing) at University of Luxembourg [35], in the end to reach results from bigger datasets. Figure 3.2 shows a brief idea of running Spark on cluster mode [2].

Following the idea in [2], the schema shows how Spark acquires the executors according the availability in the cluster. The executors are processes which run the computations. Next, the frameworks sends the tasks (codes) to the executors, in order to run it. Additionally, picture 3.2 shows that each executor works with some attached cache to manage the tasks.

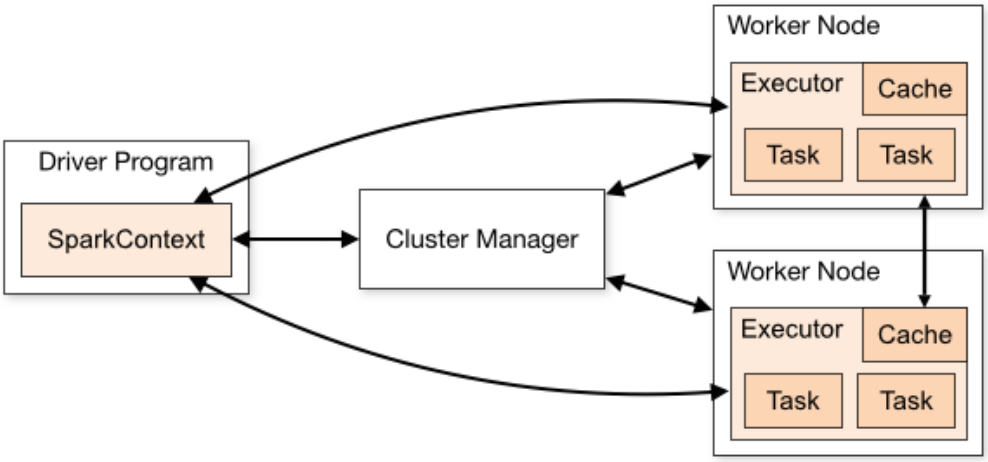
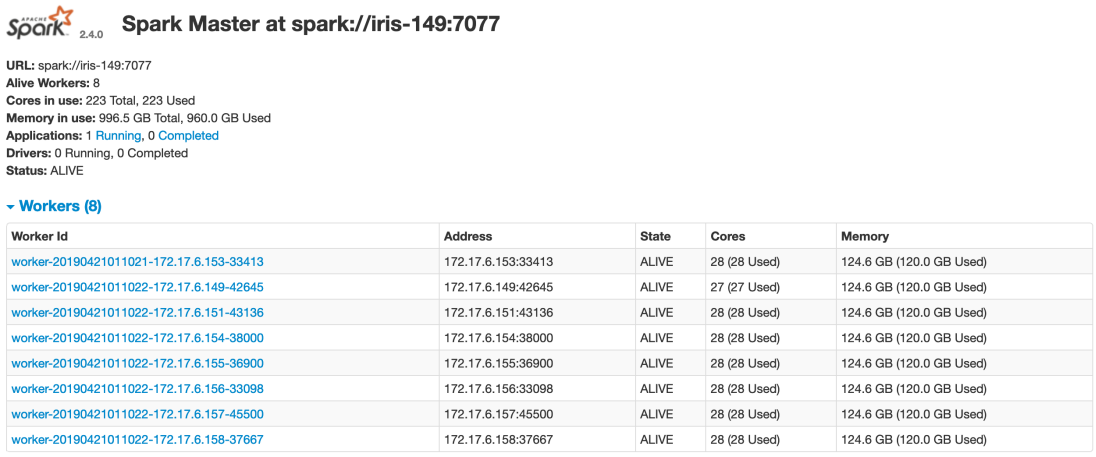


Figure 3.2: Spark Cluster Overview

Furthermore, they are linked to the Cluster Manager, which distributes the tasks. We have used these facilities on a cluster at University of Luxembourg [35], which we will cover in more detail in chapter 4. Figure 3.3 shows 8 worker nodes set up on HPC Cluster at University of Luxembourg.



Spark 2.4.0 **Spark Master at spark://iris-149:7077**

URL: spark://iris-149:7077
Alive Workers: 8
Cores in use: 223 Total, 223 Used
Memory in use: 996.5 GB Total, 960.0 GB Used
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (8)

Worker Id	Address	State	Cores	Memory
worker-20190421011021-172.17.6.153-33413	172.17.6.153:33413	ALIVE	28 (28 Used)	124.6 GB (120.0 GB Used)
worker-20190421011022-172.17.6.149-42645	172.17.6.149:42645	ALIVE	27 (27 Used)	124.6 GB (120.0 GB Used)
worker-20190421011022-172.17.6.151-43136	172.17.6.151:43136	ALIVE	28 (28 Used)	124.6 GB (120.0 GB Used)
worker-20190421011022-172.17.6.154-38000	172.17.6.154:38000	ALIVE	28 (28 Used)	124.6 GB (120.0 GB Used)
worker-20190421011022-172.17.6.155-36900	172.17.6.155:36900	ALIVE	28 (28 Used)	124.6 GB (120.0 GB Used)
worker-20190421011022-172.17.6.156-33098	172.17.6.156:33098	ALIVE	28 (28 Used)	124.6 GB (120.0 GB Used)
worker-20190421011022-172.17.6.157-45500	172.17.6.157:45500	ALIVE	28 (28 Used)	124.6 GB (120.0 GB Used)
worker-20190421011022-172.17.6.158-37667	172.17.6.158:37667	ALIVE	28 (28 Used)	124.6 GB (120.0 GB Used)

Figure 3.3: HPC 8 Worker Nodes

3.4 MLlib: a Machine Learning library in Apache Spark

In response to the increasing demand to analyse "big data", considering the efficiency of Spark regarding to the parallel model, the community started to develop MLlib, a complete Machine Learning library available for Apache Spark.

Hereby, beyond make use of the parallelism, the main point is to keep the data in only one framework, which decrease the efforts to deal with determined problem by reusing the code and without transferring to another framework.

3.4.1 Classification and Regression Algorithms

Classification algorithms implemented in MLlib, such as Decision Trees and Random Forest, are the most common used and powerful to deal with classification, as well, with regression problems. Setting as example, any forecast prediction, where we may have the use of regression trees, in order to check the probability with past data, attempting to predict the most likely for the future.

We may apply the same idea to sales information in supermarkets, considering past data and giving some important information to base decisions for future cases. Those are applications for Regression, where a model is trained from past information to predict or, to return the most probable case for the future.

However, following the same idea to train a model from some information, classification algorithms make use of some data provided as training data, where a model is also trained to, in the present case, classify the data in some way, e.g: scale classification, relevant or not, true or false probabilities, etc.

The presented study makes use of classification algorithms, in order to predict relevant content in web pages. Decision Trees and Random Forests are the chosen techniques, due to their efficiency and scalability. Usually, the parameters to reach a best model are tested under the "model tuning" technique (next section), the important remark is that some parameters are not tunable, which have to be checked manually or according the classification problem.

3.4.2 Model Tuning: Cross-Validation *k*-fold in Spark

In this section, we give some explanations of Cross Validation, as provided in Apache Spark. We have seen previously concepts of Cross Validation k-fold in chapter 2. However, besides "*CrossValidator*", in Apache Spark we may use "*TrainValidationSplit*" for hyper-parameter tuning [5].

The main difference between "*CrossValidator*" and "*TrainValidationSplit*" is that the first splits the data into *"k"*-folds where the former one runs only once, which makes it less expensive in terms of performance [5].

Apache Spark requires from the methods, three input variables for model tunig using "*CrossValidator*"; an "Estimator", which is the algorithm applied to evaluate; a "Parameter Grid", which is a set of possible parameters to test and finally; an "Evaluator", which is the metrics to evaluate each combinations. The former variable might be: a *RegressionEvaluator* when running regression problems; a *BinaryClassificationEvaluator* for binary data; or a *MulticlassClassificationEvaluator* when dealing with multiclass problems [5].

Regarding the fact that we classify the data into two categories in this study, we use *BinaryClassificationEvaluator*. However, we do not use the technique exactly like it is defined in [5]. We adapted it to save the evaluations, instead of only fitting and returning the best. We see this step in more details in chapter 4. Additionally, we show the code used in Appendices.

3.4.3 Evaluation: Precision, Recall and F-measure

Apache Spark provides several algorithms to evaluate results from a model. In the previous chapter, we gave the theoretical background of those techniques. In this section, we introduce those in the Apache Spark environment and in the next chapter will show how we practically apply those techniques in our experiments to obtain those metrics.

Metric	Definition
Precision (Positive Predictive Value)	$PPV = \frac{TP}{TP+FP}$
Recall (True Positive Rate)	$TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$
F-measure	$F(\beta) = \left(1+\beta^2\right) \cdot \left(\frac{PPV \cdot TPR}{\beta^2 \cdot PPV + TPR}\right)$
Receiver Operating Characteristic (ROC)	$FPR(T) = \int_T^{\infty} P_0(T)\, dT$ $TPR(T) = \int_T^{\infty} P_1(T)\, dT$
Area Under ROC Curve	$AUROC = \int_0^1 \frac{TP}{P} d\left(\frac{FP}{N}\right)$
Area Under Precision-Recall Curve	$AUPRC = \int_0^1 \frac{TP}{TP+FP} d\left(\frac{TP}{P}\right)$

Figure 3.4: Apache Spark MLlib: available evaluation metrics

Figure 3.4 depicts the available metrics for evaluating a Binary Classifier in Apache Spark. We highlight the Precision and Recall, which are related to the positive cases only. The main conceptual points were shown in chapter 2. Additionally, in Spark we may test the P-R curve to plot the points for different threshold values (regarding Precision and Recall), which represents the percentage of predictions under the P-R curve. Besides, ROC (receiver operating characteristic), plots the points (regarding recall, false positive rate) [4].

Chapter 4

Experimental Setup

We focus our research in the Content Analysis, hence, the Machine Learning will be covered in more details in this chapter. However, it is necessary to detail each step from the beginning. Figure 4.1, depicted below, shows how our experiment is divided.

Contextualizing with the chapter 2 and 3, the step: *Mining*, comprehends "Crawling Web Pages" in order to produce our dataset, until extracting information from the web pages with Jsoup [10] in the process "Parsing Features", which is a information extraction technique.

The step: *Analysing* is related to apply Machine Learning techniques in Apache Spark, in order to achieve the results, e.g. classification by decision trees, selection of the best parameter for the model and evaluation.

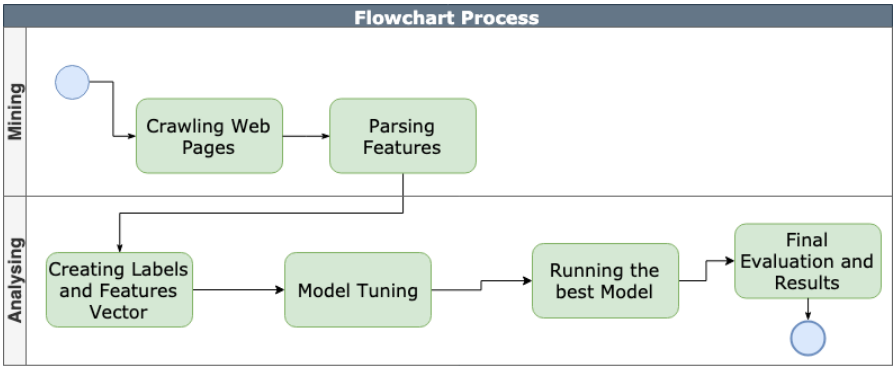


Figure 4.1: Experiments Flowchart

In the upcoming sections, we will present in more details how we have set up the environment in Apache Nutch [8], as well as how it worked in our experiment and the issues we have faced. Further, we will introduce how we worked with JSoup [10], and finally the Machine Learning techniques we applied. Those include Decision Trees, Model Tuning and Evaluation Metrics.

4.1 Crawling with Apache Nutch

Apache Nutch has been chosen as our Web Crawler for many good reasons. First, it is an open-source application, based on Apache Hadoop [8]. In addition, the installation process was straight forward, as we did not have to deal with indexing the result. Therefore, the tool is concise and simple to work with and hence, it is easy to set up the environment without spending too much time on building the dataset (which was not the focus on our study).

4.1.1 Installation

The Nutch installation requires Java Runtime/Development Environment (JDK 1.8/Java 8)[1] and Apache Ant[2] installed. Apache Ant is a library, written in Java and is commonly used to build Java applications [7], that is what we need in this work.

One important remark about the Nutch's version; it has two different versions which have different crawling ideas. The documentation is not much clear about the technical differences but in summary, version 1.x is a well matured web crawler, ready to crawl any website. Version 2.x, which is based on version 1.x, is a new project using a different approach to store the data. Following this idea we have chosen to work with the version 1.x with 1.15 being the latest version. (Released in August 2018 [8]).

If the previous required applications are installed and the chosen version is downloaded then, the installation is simple and very fast. After to extract the zip file, accessing by "Terminal" on macOs, for example, we navigate into the main folder which we will adopt the name "$NUTCH\_HOME$" from now on. Thus, we have to run the command-line "ant" to build our Java applications for Nutch. After successful building, there exists now, a path folder "runtime/local" containing everything we need to configure. Afterwards,we are able to run our first web pages crawling.

However, inside our "$NUTCH\_HOME/runtime/local$" we may check the installation running the command-line "bin/nutch", which has to show the command examples to run. After the installation step to be done, is necessary then, to configure our web crawler.

4.1.2 Configuration

In the folder "$NUTCH\_HOME/runtime/local/conf$", specifically in the file $nutch\_default.xml$, the following default configurations have been modified in order to reach our goals:

[1] Java. `https://www.java.com/`
[2] Apache Ant. `http://ant.apache.org/`

- *http.agent.name*: This configuration comes empty value by default, it is mandatory to be fulfilled in order to identify the crawler in the website we are crawling.
- *http.content.limit*: By default, the value is set to 65536 bytes. We changed it to -1, as the documentation says that the value defines the length limit for downloaded content using the http protocol. If there is a non-negative number it will be truncated at this number, otherwise it will never be truncated. As example, we tried to crawl `https://www.nytimes.com` and the return was `https://www.nytimes.com` skipped. Content of size 100120 was truncated to 65536".
- *db.ignore.external.links*: By default, it set to "false" but we changed it to "true". It allows Nutch to crawl and download URL which are not interesting for our study. As example the website `https://luxtimes.lu` has some branches in other languages, besides English and before to have changed it, those web pages were crawled too.
- $fetcher.server.delay$: This configuration is by default defined as 5.0, which means that each 5 seconds the application will "fetch" a new page. In order to be polite with the websites we are crawling, we changed to 30 seconds, then each 30 seconds we crawl one page for each domain in the seed list.

Those were the main configurations to start to crawl any website. However, according the task some other configuration might be necessary. We also took a look over: "file.content.ignored", "db.update.additions.allowed", "fetcher.parse", "fetcher.store.content". However, in the end, we left as the default configuration.

4.1.3 Crawling

Regarding the simplicity of Nutch as well as the fact that we do not want to make this step to be complex. We have taken the easiest way in Nutch to crawl the web pages. The referred process, object of this subsection, is defined and described deeper in the documentation [9]. It is interesting to understand how the process works behind, but for now, as the Nutch developers built a great script to crawl using the main functions, we adopted to use the command-line as in listing 4.1.

```
bin/crawl -s urls/seed.txt <outputDir> 2
```

Listing 4.1: Crawling Command Line

The command-line in listing 4.1 is slightly different from the one given in the documentation [9]. For this task, as we do not need to index the results only download the information. Hence, we suppressed the parameter "-i".

Once inside the main folder - "$NUTCH\_HOME/runtime/local$" - we run the command, where "-s" means "seed file", which preceding then, the seed file path.

Following by the folder where the crawlings will be saved. In the end, the number represents the number of rounds that the application will run.

In our case, we run 2 times, it means that the crawler will start in the seed file and will crawl all the URL which wherein. In the second round, it will take the URLs which where in the pages crawled in the first round, and so on, if it was the case.

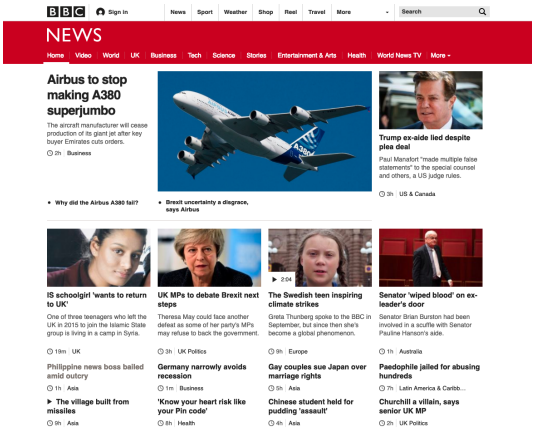


Figure 4.2: BBC News

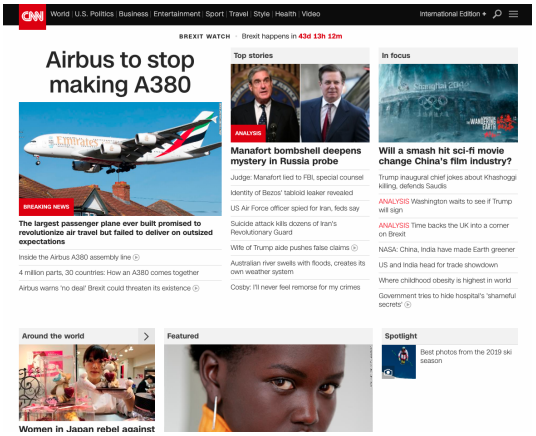


Figure 4.3: CNN News

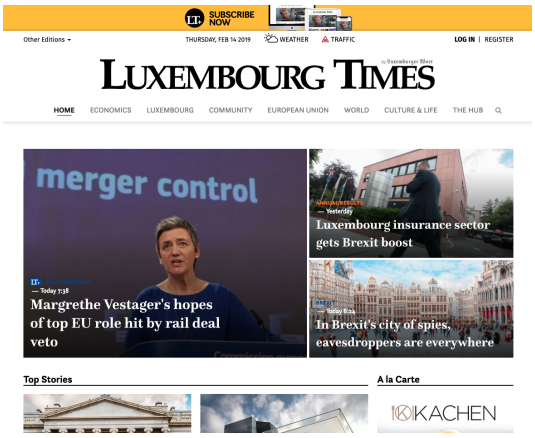


Figure 4.4: Luxembourg Times

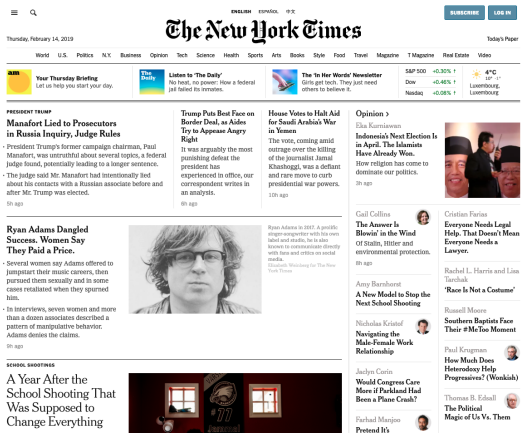


Figure 4.5: The New York Times

One remark about the parameter *db.ignore.external.links* in the configuration file: setting the referred as true, we only store the URLs which are internal links, starting from the seed file. The reason why is that in our study, we do not want to manage the external links are in the web page, once those would affect the results.

Concerning the seed file information, it must have the complete and final URL to start to crawl. Otherwise, it can count as a round (last number in listing 4.1) on Nutch crawling, to find the initial point. In addition, the "outputDir" must be different if running multiple tests, otherwise it will continue crawling all URLs containing in the folder, not from the seed.

We used as data from News Websites, the four following domains, also depicted

in the Figures 4.2-4.5, where we may see their Home Pages on February 14, 2019 around 10:40am.

- **BBC News:** `https://www.bbc.com/news`
- **CNN:** `https://edition.cnn.com/`
- **Luxembourg Times:** `https://luxtimes.lu/`
- **The New York Times:** `https://www.nytimes.com/`

4.1.4 Dumping

Once crawled, the data is stored inside the defined folder in the last step (<outputDir> in listing 4.1), split in different segments. However, the output files of Nutch, are not understandable or readable by the parser which we have chosen to use.

The Nutch application has, by itself, one command to change the whole crawled content to HTML files (listing 4.2), it is *dump* in the folder *bin/nutch*, then, our parser may works properly to do the analysis which we expected to reach, we will show it in the next section.

One important remark in that case: there is no documentation about the command we used in the Nutch website. However, there is a broader study about how to setup the Nutch environment and integrate it with other frameworks[26].

```
bin/nutch dump -flatdir -segment <segmentDir> -outputDir <outputDir>
```

Listing 4.2: Dumping Crwaled Data Command Line

We found the information in the source-code and we tried to use it, then we achieved the expected result, the HTML files. The parameter *-flatdir* set only HTML files inside the defined output folder. Otherwise, it could split each HTML file though many sub-folders. The prior parameter is followed by the source folder, under the parameter *-segment*. In the end, the destination folder, after the parameter *-outputDir*.

4.2 Parsing data with JSoup

The present section describes the development part using the JSoup [10] library to extract information from the HTML files generated by Apache Nutch [8]. From this step ahead, we turn to use the Spark environment. The only configuration necessary before encoding is to attach to the project, the Jsoup JAR file library, available in [10].

4.2.1 Library and Documentation

The Spark environment is able to run JSoup, either copying the JAR file to the spark jars folder or attaching it once running the following command line to run the spark-shell. It is also possible to attach a JAR File inside the shell, calling *"sc.addJar"* followed by the JAR file path.

```
spark-shell --jars  <Jsoup JAR path>
```

Listing 4.3: Attaching JAR Files to Spark Shell

The documentation available in [10] is very useful for either beginners or advanced users. The provided examples give the idea how it is easy to develop. Furthermore, the development community has given contributions in this field. Additionally, due to be built based on Java language, it is also possible to apply some pieces of Java codes into Scala (*spark-shell*), or converting those.

4.2.2 Parsing Data

The Scala code used in this project (available on the Appendices) starts parsing the information from HTML files, previously gathered using Apache Nutch. The main processing, regarding the parsing process, is written in the function "processData", where we may see how to reach the features for our experiments. We detail those features in the next subsection.

Basically, the parsing function starts gathering the HTML list in the folder, which are called "documents". For each document, we store some common data, such as, title and information for all document. Next, we do the same for each element in the document. There is a command where would be possible to get all elements, although it is not necessary in this study. Hence, we gather only the "DIV" elements, considered "divisions" in the web pages. Those are the most numerous and it has more important information than other elements.

4.2.3 Extracted Features

The first feature is the title of the document and the second to last is the class name of the respective DIV" element, both are not considered for the next step (which we will focus next section) to analyse the content of the files. We only use those to label our final results, if necessary.

1. **docTitle:** The title of the web page;
2. **docLinks:** Total number of links in the web page;
3. **docWords:** Total number of words in the web page;
4. **docDensity:** Total number of words divided by links in the web page;

5. **docParag:** The number of paragraphs in the entire web page;
6. **docImages:** Number of images in the web page;
7. **docChildSize:** Number of child nodes in the DOM-tree on root node;
8. **docDiv:** Total number of "DIV" elements in the web page;
9. **divLinks:** Number of links in the actual "DIV";
10. **divWords:** Total number of words in the actual "DIV";
11. **divStopWords:** Total number of stop-words in the actual "DIV";
12. **divDensity:** Total number of words divided by links in the actual "DIV";
13. **divParag:** The number of paragraphs in the actual "DIV";
14. **divImages:** Number of images in the actual "DIV";
15. **divChildSize:** Number of child nodes in the DOM-tree on actual "DIV";
16. **divClass:** The "classname" of the actual "DIV";
17. **goodBad:** It is the featured created from the "DIV" manually set to be the relevant information. It is the Labeled point for the Features Vector and it is used to evaluate the results. Its values are only "1" or "0".

In the Jsoup code part, the first decision was which features we needed to capture and send it to the machine learning model analyse. We looked at all documents (HTML files) and also for each "DIV", inside each file, to achieve the analysis for each one. The above information is captured from the HTML files. The density is the total amount of words divided by the total amount of links and it is not considered in the analyze if the number is zero.

Our focus in the presented research is related to the features. If there is some way how to improve the content analysis using different features. Therefore, that is the reason why we test different feature vectors. We present the results of these tests in the proper section, in the next chapter.

4.3 Applying Machine Learning

Once we gather the input data for our experiments and extracting the features from previous section, we begin to build a model to classify it. In the current section, we will show how we have applied the Machine Learning techniques to classify our data.

First, the initial concept is to create training data by using the features, in order to train the model. Further, we have to find the best parameters to run our model. We will see the application of Cross-Validation in our tests.

Finally, we run our classifier, based on the model learned from training data, applying the best parameters. We analyse and explain the achieved results. In

the end, we evaluate the results, applying some well known techniques, as we have shown in theory in chapter 2.

4.3.1 Decision Trees and Random Forests classifiers

The Machine Learning classifiers in Apache Spark, highlighting the Decision Trees and Random Forests, have to create a model in order to learn the features and them, to classify the test data afterwards. In the upcoming subsection, we show how to prepare the data to build a model, as well as the best parameters to apply.

However, to introduce the method, the first Decision Trees we run looked like the figure 4.6. It is clear that it is very small, but only a brief idea on how it should be in the end.

Each layer in the picture 4.6 represents one department, it is defined in the model selection together with the other parameters. We may see the train classifier builder in the listing 4.4, although we will explain all parameters in details later.

```
val modelDT = DecisionTree.trainClassifier(trainingData, numClasses,
    catFeature, bestDTImpurity, bestDTDepth, bestDTBins)
val modelRF = RandomForest.trainClassifier(trainingData, numClasses,
    catFeature, bestNuTrees, "auto", bestRFImpurity, bestRFDepth,
    bestRFBins)
```

Listing 4.4: Decision Trees and Random Forests train classifiers

On the other hand, Random Forests are simply an ensemble of Decision Trees. The amount of Decision Trees is also defined in the method *trainClassifier*, shown in the listing 4.4 as "bestNuTrees". The others parameters are the same as for Decision Tree *trainClassifier*.

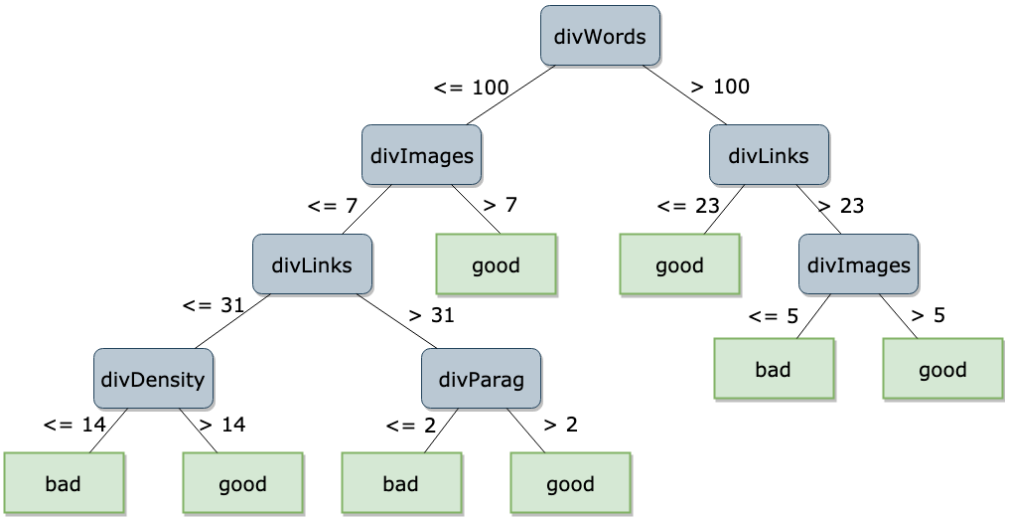


Figure 4.6: Decision Tree example

4.3.2 Training Data

It is well known that in the field of Machine Learning, it is necessary to create a training data previously. The idea is to create a learned model from this data. In the end, the technique will test the data, showing the result based on the model. In other words, the trained model learns from the data to produce results for the test data.

For this project, we needed to create a training dataset manually, in the sense of labelling the relevant information. It was necessary because we did not have any kind of information about what we could reach in the end. In order to classify which part of the web page is "good" or "bad" for our model, we needed to set it manually in the training data. We pointed in someway how, which parts are "good" or "bad". This task might requires big efforts, but it is the most important step to create a model which may produce good results.

On the other hand, the "definition" of which document compose the test or training data is random. Listing 4.5 shows how to split the dataset random, in this case 80% of Training Files and 20% as Test.

```
val Array(allTrainFiles, allTestFiles) =
   allFiles.randomSplit(Array(0.8,0.2))
```

Listing 4.5: Splitting Randomly the Dataset before Extracting Features

Further, it was necessary to decide which techniques to apply, before labelling manually the training data. It is necessary because we used the technique to validate our training data. Thus, it has been changed and improved many times. The Decision Trees are used in this study to classify the data, mainly due its efficiency. Considering a model trained, we used those to verify the consistency of our training data in small portions. Next, increasing and improving the labelling until to reach the best training dataset.

Regarding the dataset labelling task, we have labelled some amount of HTML files manually. However, we have built a bigger dataset using same automatic labelling technique to reach more HTML files, labelled as "good" or "bad". It is also covered in the results, because we test the efficiency of the model regarding the labelling technique used to label the elements.

While building a training dataset, we faced some issues which did not classified information properly in the end. First, if we put small portions of files as training data from the same web site, and we do not label half of them manually setting which parts are "good" or "bad", then the model will not learn the information properly. It happens because with few files, 50% manually labeled and 50% not, the model will not learn properly. It ensures the needs to have as many as possible cases from the same structure, with all possible different labelled cases.

A second issue: if we set only one "good" element in a file and not the whole relevant path (which contains the same information in a branch of DOM-tree), the

model will not be learned properly either. In this case, because the model may learn that information but will discard the others which are quite the same and hence, it may either "overclassify" or "misclassify" the labels.

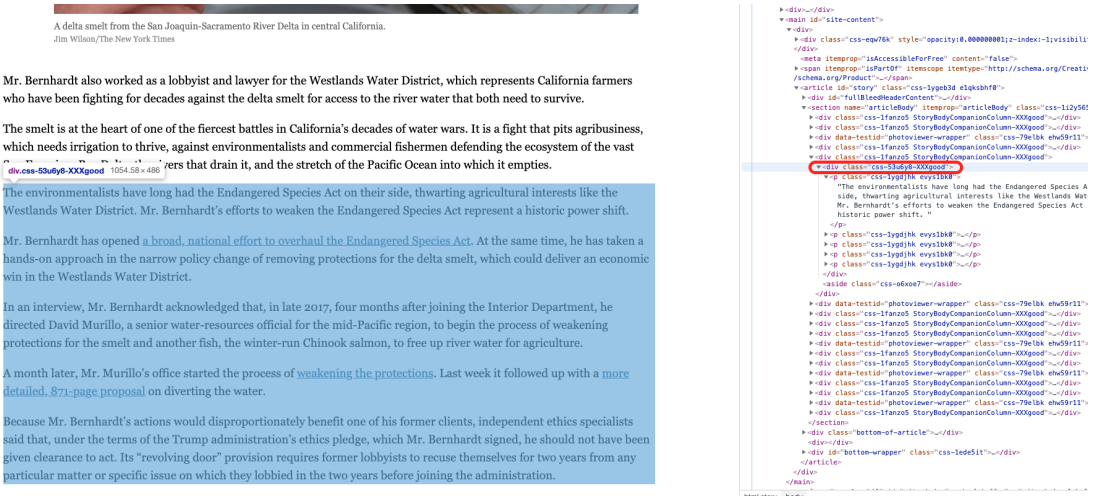


Figure 4.7: DOM-Tree Labelled Sample

Finally, the task has been executed manually, setting the "classname" of the "DIV" elements, adding the termination "-XXXgood", for those which are most relevant regarding our goals with the study. For some cases we also replaced the "classname" of the "DIV" elements using automatic text editor (automatic labelling).

Figure 4.7 shows the labelling for one HTML file, in whole relevant DOM-tree path as highlighted. It affects the initial code. Listing 4.6 shows how we take this information as a feature. When it is equal to "1", it means that the "DIV" element is "good" or relevant for the model, otherwise it is equal "0".

```
val goodBad = if (divClass.endsWith("-XXXgood")) {1} else {0}
```

Listing 4.6: Feature: the Labeled Point

This feature is considered the "Labeled Point" in the Vector to train the Decision Tree classifier. It will be the information to compare with the predicted information, in order to evaluate the model tuning, as well, the final classification.

```
val vector01 = inputRDD.map(x => (LabeledPoint(x._17.toDouble,
   Vectors.dense(x._2.toDouble, x._3.toDouble, x._4.toDouble,
      x._5.toDouble, x._6.toDouble, x._7.toDouble, x._8.toDouble,
      x._9.toDouble, x._10.toDouble, x._11.toDouble, x._12.toDouble,
      x._13.toDouble, x._14.toDouble, x._15.toDouble))))
```

Listing 4.7: Feature: the Labeled Point

All numerical features are part of the Vector, which will be the input for the Decision Tree and Random Forest classifiers. The construction of this Vector is in the listing 4.7. However, we previously mentioned that we have done some tests

to compare efficiency of the features. We have done a kind of ablation study with those features. It will be covered in more details in chapter 5.

We may noticed that "docTitle" and "divClass" (shown in section Extracted Features) are not part of this vector. Those are labels to identify the file and the element class name, respectively. Firstly, we thought in applying those as features. However, due time constraints and the high effort to adapt the code, we decided to leave them out of this study's scope.

4.3.3 Discovering the Best Model

Apart from the training data, the decision tree classifier requires some additional parameters (see listing 4.4). The variables "numClasses" and "catFeature" are related to the type of prediction.

- **numClasses**: It is the number of possible outcomes. In this case, the predictions will set either "1" meaning "good" or "0" meaning "bad". Thus, it will be always "numClasses = 2" in our experiments.
- **catFeature**: It maps the features into categories. By default it is set as "catFeature = Map[Int, Int]()". Spark takes care about this parameter if it is not defined, when using only numerical features [3]. However, it might be checked carefully in order to improve the model training, once dealing with categorical features.

The next three and last necessary variables to be set are: "bestDTImpurity", "bestDTDepth", "bestDTBins". They are related to the performance of the test, e.g, as high as "bestDTDepth" and "bestDTBins" are set, higher will be the time to train the model. That is the reason why we test all possible combinations in Cross Validation k-fold, to find the best parameters regarding the performance and metrics of each combination. However, before we explain the process, we have to understand the purpose of each variable:

- **bestDTImpurity**: There are two possible impurity index, implemented in Apache Spark for Decision Trees and Random Forests train classifiers, Gini and Entropy. It represents the homogeneity of the labels in the node [3].
- **bestDTDepth**: It is the max quantity of layers which the tree is allowed to grow, as higher as it is, the computation will increase, affecting the performance, but improving the prediction.
- **bestDTBins**: It is the maximal number of bins used when discretizing continuous features [3]. It follows the same logic as the former about performance

Listing 4.8 shows the Scala code for Cross Validation k-fold of our Decision Tree train classifier. Firstly, we take all training data and we divide into 5 k-folds if the dataset has more than 100 files. Otherwise, it divides only into 3 k-folds. It is necessary due the amount of data that is not enough, if split into many folds.

The variables "numClasses" and "catFeature" are defined according the model, in this case to a Binary Classifier, only 2 integer classes.

Further, we really start the tuning evaluation to reach the best model. The variable "modelTuningDT" will be in the end, one array with all stored evaluations. It will execute once for each combination. In other words, it will consider each k-fold (5 or 3), each possible impurity (2), each input depth (3 different in this case) and each max number of bins (4). For each set of combination, one model is trained using the partition data and parameters of the actual set. Following, the predictions are computed according the created model.

Finally, the prediction is evaluated according the metric area under precision-recall. For each combination, the evaluation is stored for future access. In the end, the set of parameters which shows the best "areaUnderPR" is selected to store the most better tunable parameters for each: "bestDTImpurity", "bestDTDepth" and "bestDTBins".

```
// Setting initial parameters
val k = if (allFiles.count > 100) {5} else {3}
val cvData = MLUtils.kFold(trainingData, k, 0)
trainingData.unpersist()
val numClasses = 2
val catFeature = Map[Int, Int]()

// Looking for the best DT model
val modelTuningDT = for (
   (cvTrain, cvVal) <- cvData;
   impurity <- Array("gini", "entropy");
   depth <- Array(5, 10, 20);
   bins <- Array(50, 100, 200, 300)) yield {
   val model = DecisionTree.trainClassifier(cvTrain,
      numClasses, catFeature, impurity, depth, bins)
   val predicLabels = cvVal.map(example =>
       (model.predict(example.features), example.label))
   val areaUnderPR = new
       BinaryClassificationMetrics(predicLabels).areaUnderPR
   ((impurity, depth, bins), areaUnderPR)
}

// Capturing the best model parameters
val bestDTImpurity = modelTuningDT.maxBy(_._2)._1._1
val bestDTDepth = modelTuningDT.maxBy(_._2)._1._2
val bestDTBins = modelTuningDT.maxBy(_._2)._1._3
```

Listing 4.8: Cross Validation k-fold for Decision Tree train classifier

Besides the Decision Trees, we test the number of trees for Random Forest train classifier, with some possible input numbers. The only additional change from the previous code (listing 4.8) is shown in the listing 4.9. We omit here to repeat all code and explanation, once it will be completely covered in the Appendices. The concept is the same of the previous one.

```
// Looking for the best RF model
val modelTuningRF = for ((cvTrain, cvVal) <- cvData;
   trees <- Array(5, 10, 20);
   impurity <- Array("gini", "entropy");
   depth <- Array(5, 10, 20);
   bins <- Array(50, 100, 200, 300)) yield {
      ...
}

// Based on Model Tuning, training the classifier
val modelRF = RandomForest.trainClassifier(trainingData, numClasses,
   catFeature, bestNuTrees, "auto", bestRFImpurity, bestRFDepth,
      bestRFBins)
```

Listing 4.9: Additional parameter for Random Forest train classifier

In the code, only one "Array" is added with some possible input number representing the number of trees ("trees <- Array(5, 10, 20)") in the Random Forest. It allows testing possible combinations of models which may return better results in our tests, for each amount of "trees in the forest".

4.3.4 Evaluation

We evaluate all possible model combinations in "model tuning" to find the best model. Additionally, we stored those evaluations for future checking ("modelTuningDT" and "modelTuningRF" arrays generated in listings 4.8 and 4.9). However, we do it again in this step but only once. It is necessary because we only evaluated the training data before, split into k-folds. Thus, in this section, we show the evaluation results for final tested data. It has been done using the best parameters found on last section and all training data to create a final model.

The evaluation follows the concepts in [4] for "Binary Classification", once our data is classified only with predictions "1" or "0". Thus, we are based the same code provided in there with changes. The listing 4.10 shows the code used to reach the final metrics for our experiments in Decision Trees (we omit the code for Random Forest, it may be seen in Appendices).

Firstly, we generate a RDD with all predictions and labels which will be used as the input for the method "BinaryClassificationMetrics". It uses the same Decision Tree model learned, but in the test data. Further, it computes all available metrics (figure 3.4). The results are presented in the next section.

Precision, Recall and F-Measure are calculated by threshold by the method "BinaryClassificationMetrics" in MLlib. It is usual when the classifier works with many thresholds according the classifiers goal. However, in this study we will consider only the threshold equal to "1". That is the reason why we take only the maximum one, according listing 4.10. We also take the percentage of the area under Precision and Recall curve. As well, the receiver operating characteristic (ROC Curve).

```
// Predicting based on the best learned model and generating metrics.
val predicLabelsDT = testData.map {
   case LabeledPoint(label, features) =>
   val prediction = modelDT.predict(features)
   (prediction, label)}
val metricsDT = new BinaryClassificationMetrics(predicLabelsDT)

// Printing Metrics - Results
println("Precision DT: " + metricsDT.precisionByThreshold.max._2)
println("Recall DT: " + metricsDT.recallByThreshold.max._2)
println("F-Measure DT: " + metricsDT.fMeasureByThreshold.max._2)
println("AUC-PR DT: " + metricsDT.areaUnderPR)
println("AUC-ROC DT: " + metricsDT.areaUnderROC)
```

Listing 4.10: Final Evaluation

Chapter 5

Results

Depending on the context to be analysed, we may show the results in different perspectives. If our goals were related to the performance of the cluster, we could analyse results under this view. Still, if it was mostly related to the predictions by itself, we could present the data predicted. However, we wish to show if the model works and its possible assertively. In addition, what is the best collection of features to classify the content. Hence, we focus here on the evaluation's results and the model tuning of parameters. Applying the same methodology for different datasets and sets of features. Therefore, we present the results for each technique available in Spark and applied on each dataset and features set.

5.1 Datasets and "Features Set"

Since the datasets always have the same data, it is not necessary to run the same test several times to compare the results. Actually, only one run of the application will produce the needed results. Even when splitting the data randomly, the results will not change drastically. Thus, we created different dataset to have possible different results and analysis.

As we have mentioned previously, we have HTML files from 4 "News Agencies Web Site". For each domain, we labelled some amount of files to compose our datasets. There are two different ways to label: strictly manual or using some automatic tool. The manual labelling is done by opening file after file, checking each element to reach the relevant content and labelling according we shown previously (figure 4.7). On the other hand, by automatic labelling, we mean that the files are used as input for some text editor, which adds "-XXXgood" termination to the *classname*.

The table 5.1 shows the amount of manually and automatically labelled HTML files for each domain. The goal is to compare the model's efficiency for each website structure. LuxTimes, for example, has an small structure, compared to the others. During our analysis, each HTML file of this domain contains around 90 or 100 "DIV" elements, while the files of the others domains may contain up to 300 "DIV" elements.

		Manual	Automatic
BBC	https://www.bbc.com/news	60	390
CNN	https://edition.cnn.com/	42	420
LuxTimes	https://luxtimes.lu/	60	200
NYT	https://www.nytimes.com/	62	340

Table 5.1: HTML files per Dataset and Labelling Technique

On the other hand, in order to have different results and to achieve the best collection of features, we establish different features vector sets. Table 5.2 shows the different Vectors of Features. This will be compared under Decision Tree and Random Forest techniques, for each domain dataset and kind of labelling.

	Labelled Point	**Features Vector**
Vector 01	goodBad	[docLinks, docWords, docDensity, docParag, docImages, docChildSize, docDiv, divLinks, divWords, divStopwords, divDensity, divParag, divImages, divChildSize]
Vector 02	goodBad	[docLinks, docDensity, docParag, docImages, docChildSize, docDiv, divLinks, divDensity, divParag, divImages, divChildSize]
Vector 03	goodBad	[docWords, docDensity, docParag, docImages, docChildSize, docDiv, divWords, divStopwords, divDensity, divParag, divImages, divChildSize]
Vector 04	goodBad	[docLinks, docWords, docParag, docImages, docChildSize, docDiv, divLinks, divWords, divStopwords, divParag, divImages, divChildSize]
Vector 05	goodBad	[docLinks, docWords, docDensity, docParag, docChildSize, docDiv, divLinks, divWords, divStopwords, divDensity, divParag, divChildSize]
Vector 06	goodBad	[docLinks, docWords, docDensity, docImages, divLinks, divWords, divStopwords, divDensity, divImages]
Vector 07	goodBad	[divLinks, divWords, divStopwords, divDensity, divParag, divImages, divChildSize]

Table 5.2: Features Scenarios for Test

We have established some kind of ablation study, where we start by testing the vector with all features. In each case, we remove one group of features. Thus, the vector 01 parse as features all the numerical features, regarding the document level and the "DIV" element level. Next, for vector 02, we excluded all the features related to the "words", such as, "docWords", "divWords" and "divStopwords".

Further, for vector 03 we do not consider the features related to the "links", either for document or element ("docLinks" and "divLinks"). Next, we remove the density from vector 04, this feature is a computation of "Words" divided by "Links",

we want to see if it impacts the results. It is also done in both levels (documents and "DIV"). Besides, we do the same for "images" in vector 05. However, for vector 06, we want to remove of this scenario all features more related to the structure of the web page, such as, "paragraphs", "childNodeSize" and "docDiv" (total number of "DIV" in a document).

Finally, the vector 06 has the conception to remove all the features from the document level. The final goal testing different features vector is to reach the best collection of features. We run the tests for each dataset with each one of those referred "Features Set".

5.2 Final Results

We evaluated our results by using the following 5 metrics: Precision, Recall, F-Measure, Area Under Precision-Recall Curve and Area Under ROC Curve. For each metric, we show the results for each news website in a graph: BBC, CNN, LuxTimes and NYT. Further, each graph represents the results in 4 groups: Automatic Labelling for Decision Tree (Auto DT), Automatic Labelling for Random Forest (Auto RF), Manual Labelling for Decision Tree (Manual DT) and, Manual Labelling for Random Forest (Manual RF). Finally, each group in the graph has the results for each features vector, 7 in total, as previously mentioned.

5.2.1 Precision

The Precision is the measure of positive cases which have been correctly classified as positive. The approach applied in Spark, as we have seen in previous chapters, computes the Precision dividing the TP by the sum of TP and FP. We compare the Precision for each dataset and features set. Plus, comparing the technique used to label (automatic or manual) and further, using Decision Trees and Random Forests.

Figure 5.1 shows the Precision for the BBC news pages. First, we highlight the slightly difference between the manually labelled and the automatic labelled dataset. There are 390 automatically labeled files in contrast to only 60 manually labeled files. Between the techniques: Decision Trees (DT) ad Random Forests (RF), the precision is similar. Among the vectors, the results are all similar, either. Hence, we may conclude that the automatic labeling has been done properly.

Regarding the CNN files, in figure 5.2, its classification's precision are similar for almost all cases among the manual labelling. However, in automatic labelling there are some differences among the vectors. It may happens because it is the bigger dataset with 420, where all labelling were done automatically. Possibly, some information has been missed in this labelling. In addition, this site has many elements to classify in the whole HTML structure. Hence, in automatic, it classified better when there are more features in the vector, except when removing "words" in vector 02.

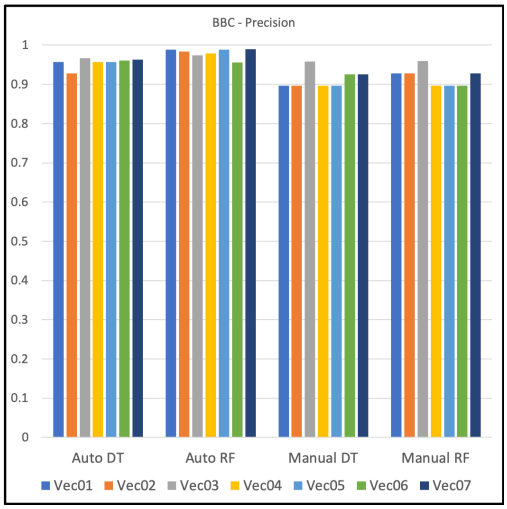


Figure 5.1: Precision - BBC

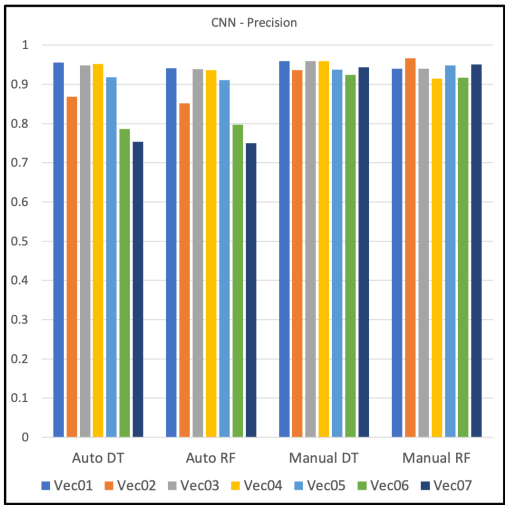


Figure 5.2: Precision - CNN

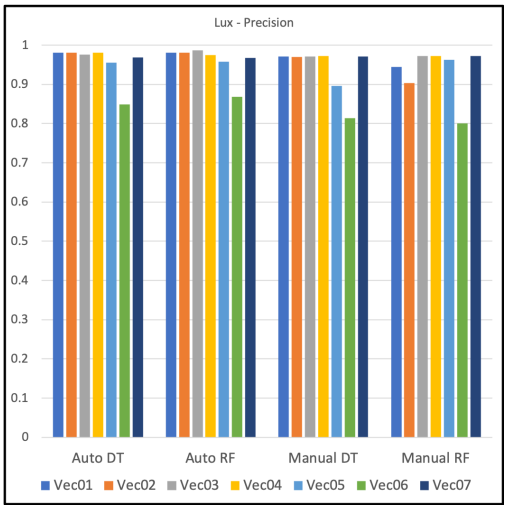


Figure 5.3: Precision - LuxTimes

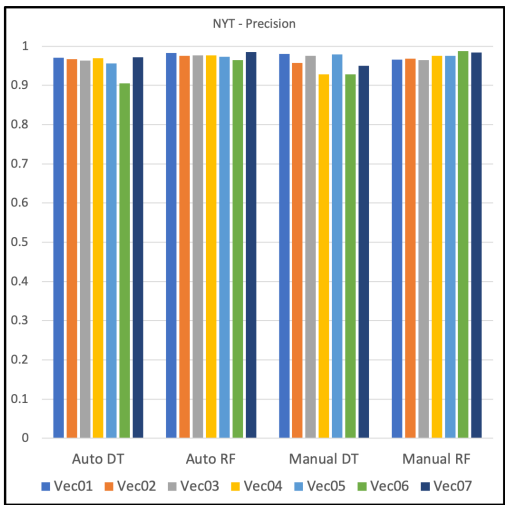


Figure 5.4: Precision - NYT

On the other hand, figure 5.3 shows the precision for LuxTimes. It has the easier and smaller HTML structure among our dataset. That is the reason why the tests performs similar in almost all the cases. However, the vectors which have more features perform better than others, except vector 07 which has less features but only for "DIV" elements. In this case, it happens because of its smaller structure, as well as, the automatic labelling probably done properly.

Finally, the New York Times HTML files have also similar precision results for all scenarios (figure 5.4). It may ensures that the automatic labelling might has been done correctly. Further, it was observed during the manual labelling that: NYT HTML files normally have only two types of "DIV" elements which we could classify as good in its structure. Regarding the slightly difference in vector 06, between DT and RF, we may notice that it has improved in Random Forest.

5.2.2 Recall

Recall is the rate of *TP* (label and prediction positives) over all positive labeled cases ($TP + FN$). While precision considers the predictions, recall consider the labels. Together with Precision, Recall is the base for the next evaluations: F-Measure, Area Under P-R Curve and Area Under ROC Curve. Hence, the results presented in the previous section with this section will affect directly to the next ones.

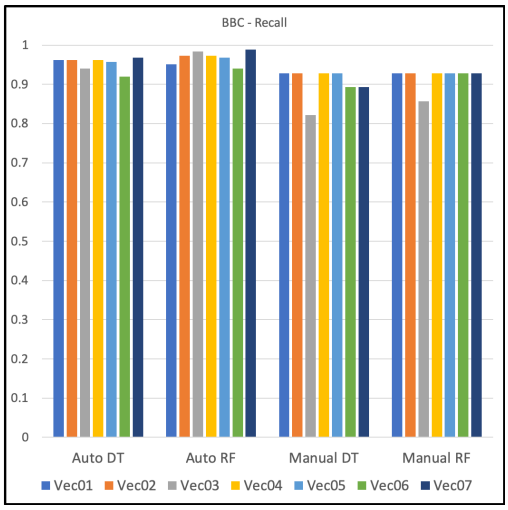


Figure 5.5: Recall - BBC

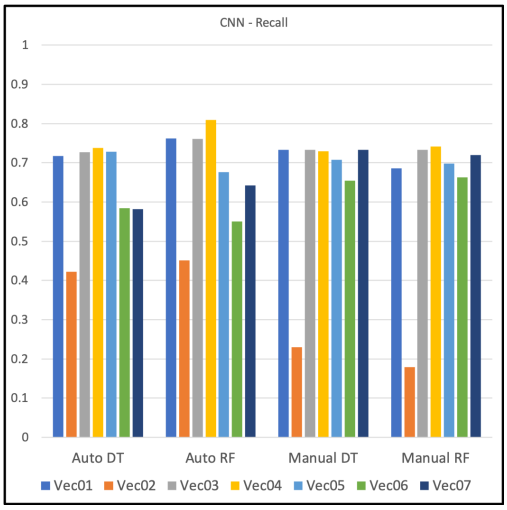


Figure 5.6: Recall - CNN

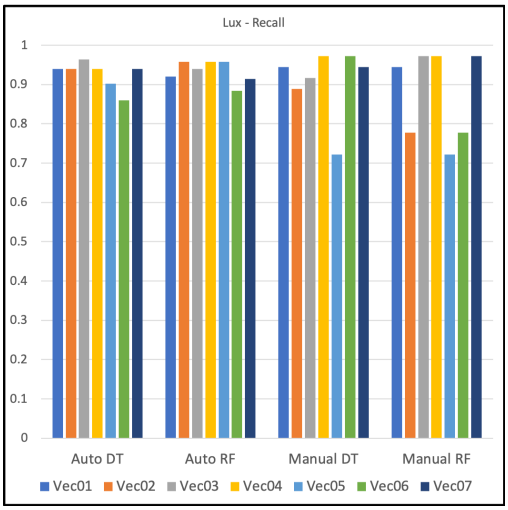


Figure 5.7: Recall - LuxTimes

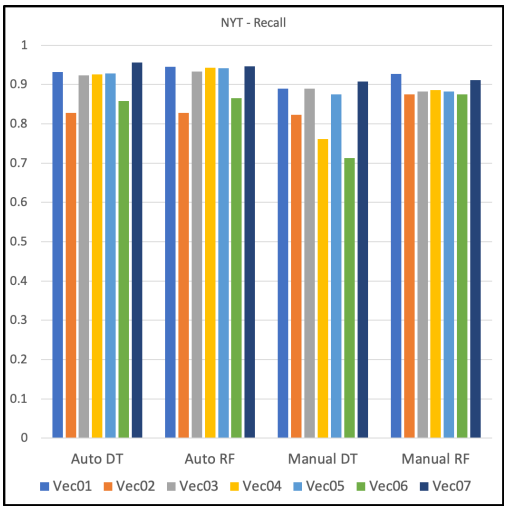


Figure 5.8: Recall - NYT

The Recall results for BBC (figure 5.5) are similar for almost all cases, or even, the same when looking the manual labelling. The better results for automatic labelled dataset ensures the previous explanations for Precision: we may consider the automatic labelling properly done. Additionally, the manual labelling has less amount of HTML files. The slightly difference for vector 03, in manual labelling, may be due to the random process of splitting the data.

Figure 5.6 depicts the most interesting case in this section. For Recall - CNN,

the differences among all scenarios have explanations. Firstly, this site has many different elements to classify in the whole HTML structure. Secondly, it is hard to gather HTML files with news, because this site has many pages which are called sections pages (with many links to the news and it is not a news page indeed). This prior fact, together with the difficulty to manually label, is the explanation why we have few files in manual. It impacts in the results.

However, we may highlight important outcomes: vector 01, 03, 04 and 05 perform good in almost all the cases, while vector 02 and 05 do not. It proves the importance of the "words" and the "structural" features in this case. In addition, vector 07 (the one with only features related to the elements) performs better with manually labeled HTML files, it may prove the needs to improve the automatic labelling for CNN.

Regarding LuxTimes in figure 5.7, we may highlight the behaviour when removing the "images" features in vector 05, which impacts directly in the manual labelling. It may be due the amount of files in manual labelling. LuxTimes dataset has 60 HTML files in manual labelling, while there are 200 HTML files in automatic labelling dataset.

One important information is about vector 06. It has a large difference in manual labelling, between Decision Tree and Random Forest. The possible explanation may be the fact that we have more than one tree, normally 10, at least. Once the dataset is shorter, the segregation of structural features (vector 06) might have impacted directly in the prediction, when running multiples trees (Random Forest).

Finally, according figure 5.8, the Recall results for NYT are in general pointing out the needs to keep the "words" in the features set. The results for Precision shown that the over-classification is not that much impacting. However, the most affected cases by "misclassification" are related with the ablation of "words" or the structural features (vectors 02 and 06).

5.2.3 F-Measure

We have covered previously the definition of F-Measure. It is the harmonic mean between Precision and Recall. Hence, the results which we are going to present below will reflect the combination of the two previous metrics. Thus, we will not add any further explanations as they follow from the results for Precision and Recall.

However, we may highlight some points. Regarding BBC (figure 5.9), it is ensuring the good results we depicted previously. For CNN (figure 5.10), it highlights the impact in vector 02 (segregating the "words" features). It is more expressive in manual labelling.

Furthermore, LuxTimes and NYT (figures 5.11 and 5.12) have performed as good as before. The exceptions, again, the structural features in general for LuxTimes. In addition, the "words" have impacted in NYT.

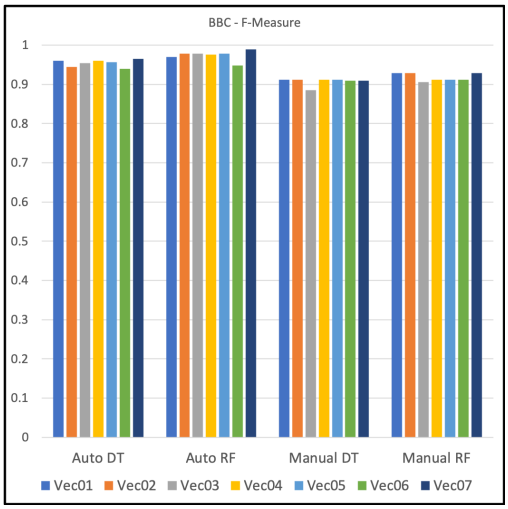


Figure 5.9: F-Measure - BBC

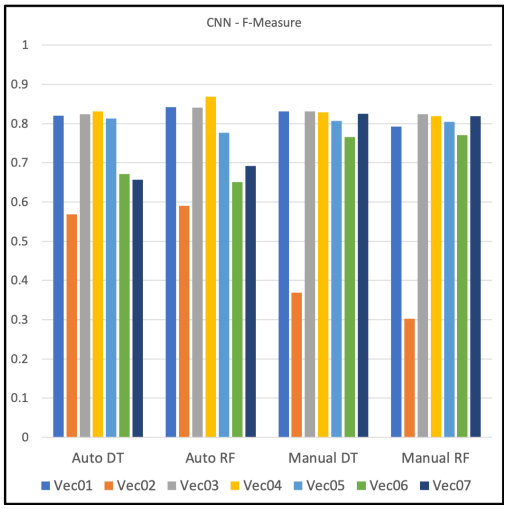


Figure 5.10: F-Measure - CNN

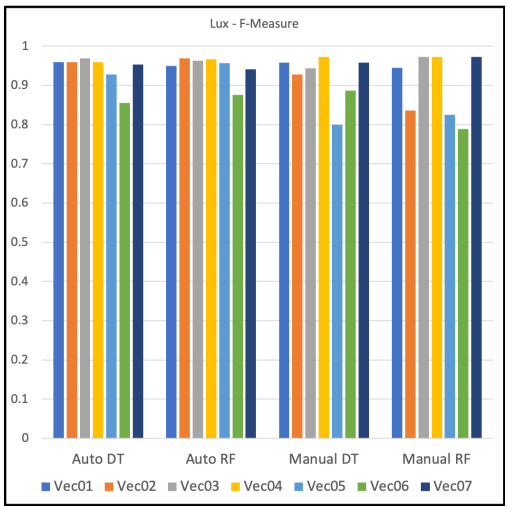


Figure 5.11: F-Measure - LuxTimes

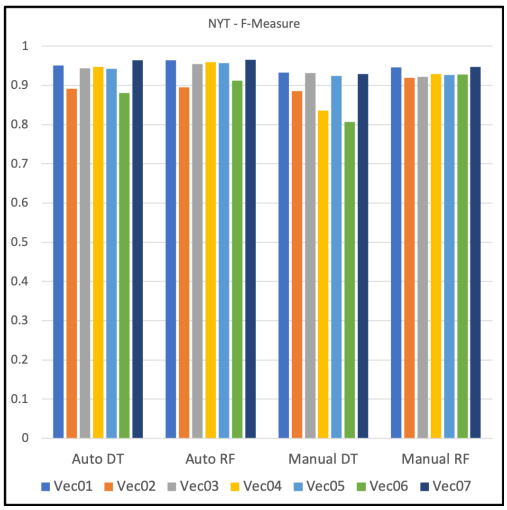


Figure 5.12: F-Measure - NYT

5.2.4 Area Under PR Curve

The Precision-Recall curve generates a curve with points of Precision and Recall for each threshold. Firstly, thresholds are not important in the scope of this project, since we were using a Binary Classifier. Secondly, the curve visualization would not be the better approach to compare so many points as we want to do. Thirdly, Spark provides the percentage of the area under the Precision-Recall curve, it is the same number which we used to evaluate the Cross-Validation. Therefore, instead of plotting the curve and its under area, we plot a comparison between the features set, for each dataset (see figures 5.13-5.16). This comparison represents in each point on the graph, a percentage of the total area under the curve. It may give a better understanding, besides showing a better way to compare each set of features, dataset and techniques.

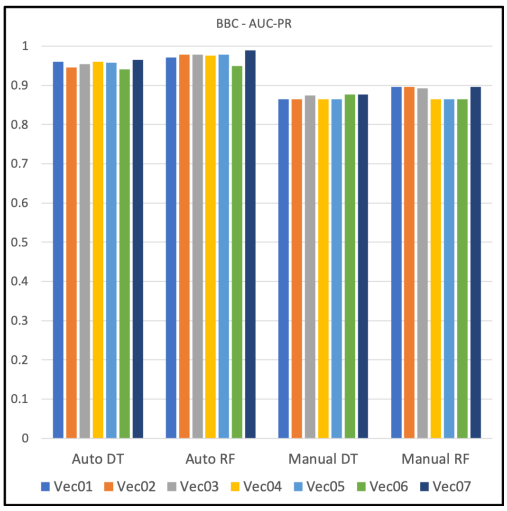


Figure 5.13: AUC P-R - BBC

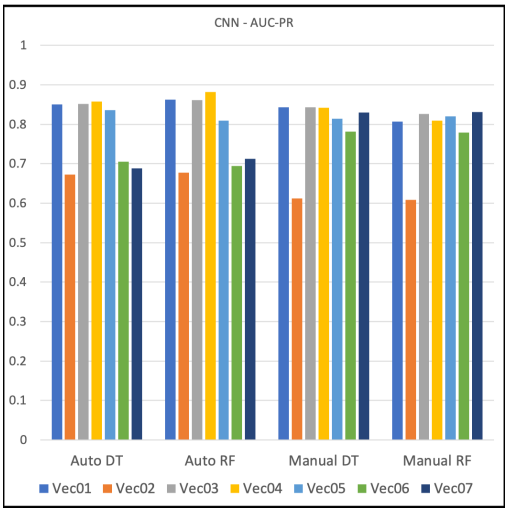


Figure 5.14: AUC P-R - CNN

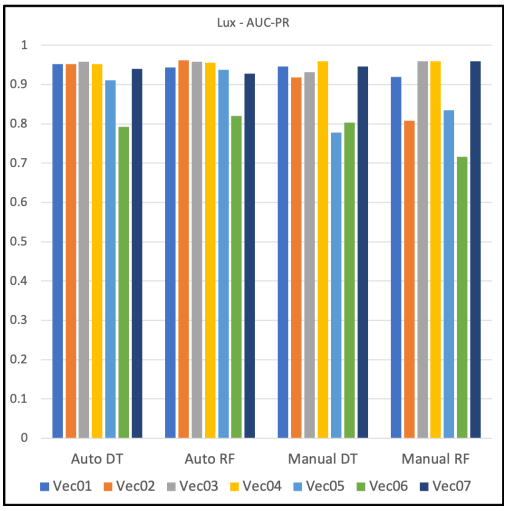


Figure 5.15: AUC P-R - LuxTimes

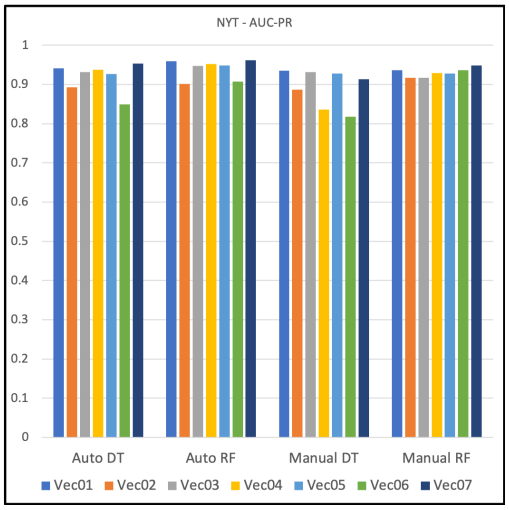


Figure 5.16: AUC P-R - NYT

5.2.5 Area Under ROC Curve

The ROC Curve (receiver operating characteristic curve) plots the Recall against the False Positive Rate.[1] The prior approach is the rate of FP divided by the sum of FP and TN.[2] This technique does not depend of Precision, hence, the results will be slightly different from the previous ones. However, it is still ensuring the same discoveries which we faced in those previous ones (see figures 5.17-5.20).

According to Lior [33], one characteristic of the area under the ROC curve is the fact that it does not depend on the imbalance of the training dataset. Hence, it may compares fairer two classifiers, with more information than only comparing

[1] https://en.wikipedia.org/wiki/Receiver_operating_characteristic
[2] https://en.wikipedia.org/wiki/False_positive_rate

"misclassification" rates. Additionally, Spark provides the percentage of the area under the curve, similar to the one covered in the prior section. It confirms our results, showing the slightly differences while segregating "words" and the structural features in some cases.

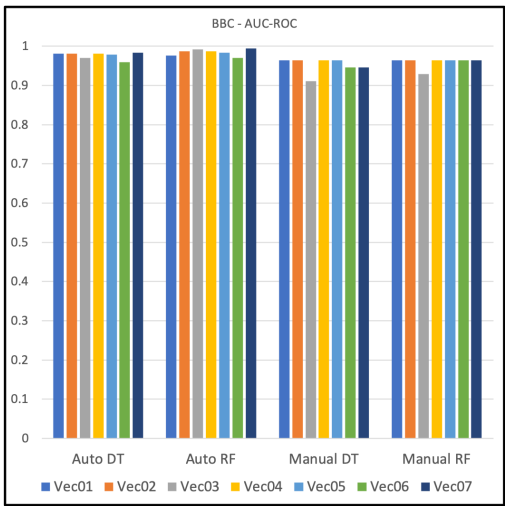


Figure 5.17: AUC ROC - BBC

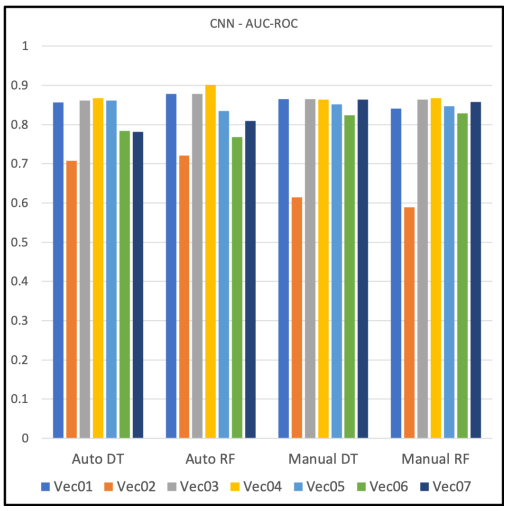


Figure 5.18: AUC ROC - CNN

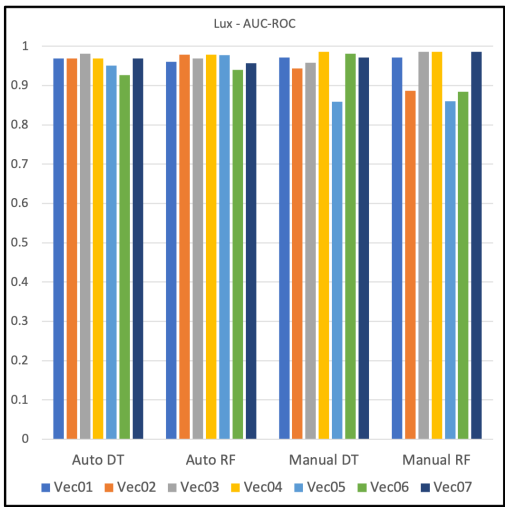


Figure 5.19: AUC ROC - LuxTimes

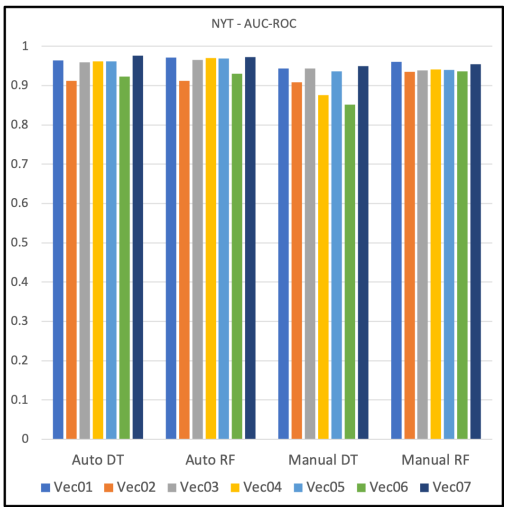


Figure 5.20: AUC ROC - NYT

Lastly, all results depicted in the previous graphs are available in the appendix section. In addition, some results for initial tests are also graphically represented, as well as the source-code applied. Besides, it is possible to check all project information, tests and codes in the repository for this project on GitHub [12].

Chapter 6

Conclusion

We have presented the theoretical aspects of some well known techniques in the field of Machine Learning and Data Mining, and applied them in the domain of Content Analysis. Besides, due to its parallelism concept, Spark provides us with the efficiency to ensure that the tool is suitable to deal with Big Data problems. Despite the actual study, the tool is very concise and a solid choice, nowadays, to deal with a large variety of classification or regression problems.

Further, we set up our environments to create our datasets from the news web pages. We take care about politeness and efficiency while building those datasets, neither crawling too many sites at the same time, nor too many pages from the same time. Moreover, Nutch creates a hash name for each HTML file after "dumping". This ensures we do not use the same HTML file during training or testing phases. Additionally, the overfitting of the model is avoided and it ensures the fairness of our tests.

Using JSoup, in order to extract features, is considered as very efficient. This is combined with the Spark's RDD concept, which guarantees us that all our tasks will be running in parallel mode and will be completed therefore much faster as if they were executed in sequential mode. We have used those RDDs concept since from the beginning, to reach the HTML files in some folder, until the end, once evaluating the models.

Decision Trees and Random Forests are very efficient methods for classification tasks. Besides, those tasks are not considered hard to be implemented, if we only have two classes of results. This is the case for us, we have to classify the content in two classes/categories: "1" for "good" or "0" for "bad". It is called a Binary Classification.

Furthermore, the model tuning ensures reaching the best model to train a classifier. We make use of "area under the curve Precision-Recall" as a parameter to evaluate the likelihood of the model. The technique is used when evaluating the final results. We have trained some amount of data, randomly split for each dataset. Next, we do model tuning to obtain the best model and then we are building the model to test it with the remaining data. This process gets repeated according to the

number of feature vectors that we are using. The goal of this test is to figure out which collection of features fits the best, in terms of good results, with our experiments.

Finally, we have reached the final results and compare/evaluate the datasets and features sets by using: Precision, Recall, F-Measure, Area Under Precision-Recall Curve and Area Under ROC Curve. Those metrics are available in Spark, for Binary Classification. We make use of them to check the best fit for each test scenario. During this research we conclude that it is possible to build a model in the referred tool. Further, using all the available techniques which are well known and necessary to ensure the efficiency of the model.

6.1 Research Questions

We have presented the results found while running the experiments. Additionally, the techniques available are very well used by the community and many studies have been published making use of them. Besides, the fact that the published studies have been well succeed in many cases as we have seen. Therefore, we may answer the research questions presented in the Introduction section, also reminded below.

1. **Is it possible to build a Machine Learning model to extract the information from some web pages and analyse them in order to classify the best content part in a web page?**

   We have built our model using Apache Spark and its library for Machine Learning. The tool is complete to deal with this kind of problems, starting from the data extraction to the final evaluations. We have seen that the model works in all cases, which has been proved by the test results. Not all results are considered great. However, all of them are good as experience to improve the model for further studies. Therefore, we conclude that it is possible to be done. Indeed, we have done it and it worked to classify the data, as better as it was possible.

2. **If so, how could we evaluate the performance or the results of the model?**

   The evaluations might be done, simply using the well known techniques to evaluate classification models, as we have seen in the theoretical background chapter. Those techniques, such as, Precision and Recall, are available in Spark's Machine Learning library. It may be done easily, with few lines of code as we have seen. Precision evaluates the rate of positive predicted cases, while Recall evaluates the positive labeled cases rate. Both techniques are essential to compute the F-Measure, which is considered as the harmonic mean between those techniques. Besides, the area under the curve Precision-Recall considers both techniques either. In order to calculate Precision and Recall, True Positives, True Negatives, False Positives and False Negatives,

which were covered in this paper, are applied in Spark, hence, in this study, to evaluate the performance of the model.

3. **Is it possible to apply the model to all types of web pages?**

   The algorithm written in Scala, as well as the web crawler, are able to deal with any kind of HTML pages. The JSoup library may also gather information from CSS pages. On the other hand, we have seen the high importance of training as many HTML files as possible, as well as to have as many features as possible, only taking care to avoid overlapping. Thus, we may conclude that it is possible to apply the model to any type of web page. However, depending on the context of the web pages and/or the classifications goals, it might be necessary to adapt the features accordingly.

4. **What could be the best collection of features for analysing and classifying the web content?**

   The results depicted especially for the area under the Precision-Recall curve may represent a good view for understanding this issue. We have tested HTML files from 4 different websites, but with the same goal. Those 4 different sites are news websites. It is the same kind of content and the context should not change. Further, it is important to take into account the structure of them. Due to business reasons or any other, they may also have different structures internally. However, in general, the most impacting ablation study was noticed when removing "words" features and structural features (vectors 02 and 06).

   On the other hand, vector 01 (the one with all available features) is consistent in all tests. Besides, vector 07 (the one with only element features) is also very consistent. This former has only inconsistency in CNN, which we have pointed out possible improvements. Hence, we may conclude that it is important to train as much as features as possible, but always taking into consideration the conflict among those. In this study, vector 01 is the best collection of features.

   In addition, vector 07 may be considered a second-best collection, even if not the most expressive in the number of features. However, in this case, would be necessary to investigate deeper the reasons why CNN performs worst for automatic labelling. Finally, it ensures the needs to train as many features as are available, once it reduces the needs to investigate deeper the datasets.

6.2 Future Work

Regarding the experience concluding this work, we may leave some improvements/optimizations of the Scala source-code as future work. It is possible to use more available techniques to evaluate or plot the results in Spark. In addition, the model tuning may also be done using different techniques, such as, methods available in others versions of the machine learning library in Spark. There is a

more concise method for model tuning. However, we have not tested or implemented it, once it requires another version of machine learning library in Spark. As well, it could require more efforts in coding. Thus, it could be done as future work.

Besides the improvements in the code, the dataset composition could also be improved. By increasing the amount of manually labeled HTML files, one may be able to achieve better results for the evaluations. Further, the features available might be improved and the scenarios increased to test different possibilities. Plus, in some automatic labelling, maybe, would be better to compare some files, in order to ensure if they have indeed the same structure. Some news web sites may have different branch, hence they may have different elements to label as good content.

Finally, all the content of the presented study will be available in [12]. It includes the used dataset, codes, results of the experiments and any additional and relevant document. It might be used for any academic purposes and improvements in this study.

Appendices

We present in this chapter some additional information about our study, such as, the Scala source-code applied; one additional experiment which we have done in the beginning, as well as some complementary result tables.

Source Code

In order to ensure the reliability in this study, we share the source-code applied in listing below. It has some comments, in order to make it easier to be understood. It is also available in [12].

```
// All necessary import to deal with this code.
import java.io
import scala.io.Source
import org.jsoup.Jsoup
import org.jsoup.nodes._
import org.jsoup.select._
import org.apache.spark.mllib.util._
import org.apache.spark.mllib.tree._
import org.apache.spark.mllib.evaluation._
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import scala.collection.JavaConverters._
import scala.collection.mutable.ArrayBuffer
import org.apache.spark.rdd.RDD

// Loading Stopwords, the path has to be revised.
val stopwords = Source.fromFile("./stopwords.txt").getLines.toArray

// Def to extract the data from HTML files. It has to flatMap in the
   end because there are two maps one insie the other.
def processData (allFiles: RDD[java.io.File]) = {
   val toFlatData = allFiles.map(file => {
   val doc = Jsoup.parse(file, "UTF-8")
   val dataDiv = doc.getElementsByTag("div").asScala.map(div => {
      val docTitle = doc.title.replaceAll("[,.;!?(){}\\[\\]<>%]", "")
      val docLinks = doc.select("a[href]").size
      val docWords = doc.body().text.split("[\\W]+").size
      val docDensity = if (docLinks > 0 && docWords > 0)
         {docWords / docLinks.toFloat} else {0}
      val docParag = doc.select("p").size
```

```
      val docImages = doc.select("img").size
      val docChildSize = doc.childNodeSize
      val docDiv = doc.getElementsByTag("div").size
      val divLinks = div.select("a[href]").size
      val divWords = div.text.split("[\\W]+")
      val divStopwords = divWords.filter(stopwords.contains(_)).size
      val divDensity = if (divLinks > 0 && divWords.size > 0)
         {divWords.size / divLinks.toFloat} else {0}
      val divParag = div.select("p").size
      val divImages = div.select("img").size
      val divChildSize = div.childNodeSize
      val divClass = div.className
      val goodBad = if (divClass.endsWith("-XXXgood")) {1} else {0}
      (docTitle, docLinks, docWords, docDensity, docParag,
         docImages, docChildSize, docDiv, divLinks, divWords.size,
         divStopwords, divDensity, divParag, divImages,
         divChildSize, divClass, goodBad)
      })
      (dataDiv)
   })
   toFlatData.flatMap(x => x.map(s =>s))
}

// Def to create different features vector scenarios to train and
   test.
def createFeaturesVector (inputRDD: RDD[(String, Int, Int, Float,
   Int, Int, Int, Int, Int, Int, Int, Float, Int, Int, Int, String,
   Int)]) = {
   val vector01 = inputRDD.map(x => (LabeledPoint(x._17.toDouble,
      Vectors.dense(x._2.toDouble, x._3.toDouble, x._4.toDouble,
         x._5.toDouble, x._6.toDouble, x._7.toDouble,
         x._8.toDouble, x._9.toDouble, x._10.toDouble,
         x._11.toDouble, x._12.toDouble, x._13.toDouble,
         x._14.toDouble, x._15.toDouble))))
   val vector02 = inputRDD.map(x => (LabeledPoint(x._17.toDouble,
      Vectors.dense(x._2.toDouble, x._4.toDouble, x._5.toDouble,
         x._6.toDouble, x._7.toDouble, x._8.toDouble,
         x._9.toDouble, x._12.toDouble, x._13.toDouble,
         x._14.toDouble, x._15.toDouble))))
   val vector03 = inputRDD.map(x => (LabeledPoint(x._17.toDouble,
      Vectors.dense(x._3.toDouble, x._4.toDouble, x._5.toDouble,
         x._6.toDouble, x._7.toDouble, x._8.toDouble,
         x._10.toDouble, x._11.toDouble, x._12.toDouble,
         x._13.toDouble, x._14.toDouble, x._15.toDouble))))
   val vector04 = inputRDD.map(x => (LabeledPoint(x._17.toDouble,
      Vectors.dense(x._2.toDouble, x._3.toDouble, x._5.toDouble,
         x._6.toDouble, x._7.toDouble, x._8.toDouble,
         x._9.toDouble, x._10.toDouble, x._11.toDouble,
         x._13.toDouble, x._14.toDouble, x._15.toDouble))))
   val vector05 = inputRDD.map(x => (LabeledPoint(x._17.toDouble,
      Vectors.dense(x._2.toDouble, x._3.toDouble, x._4.toDouble,
         x._5.toDouble, x._7.toDouble, x._8.toDouble,
         x._9.toDouble, x._10.toDouble, x._11.toDouble,
         x._12.toDouble, x._13.toDouble, x._15.toDouble))))
   val vector06 = inputRDD.map(x => (LabeledPoint(x._17.toDouble,
      Vectors.dense(x._2.toDouble, x._3.toDouble, x._4.toDouble,
```

```
         x._6.toDouble, x._9.toDouble, x._10.toDouble,
         x._11.toDouble, x._12.toDouble, x._14.toDouble))))
   val vector07 = inputRDD.map(x => (LabeledPoint(x._17.toDouble,
      Vectors.dense(x._9.toDouble, x._10.toDouble, x._11.toDouble,
         x._12.toDouble, x._13.toDouble, x._14.toDouble,
         x._15.toDouble))))
Array(vector01,vector02,vector03,vector04,vector05,vector06,vector07)
}

// It captures the entire path for all files in the current folder
   ending with "html" extension. It results in an array which will
   be parallelized to serve as an input in the def processData.
   Then, it is random split to be processed. Some prints are done to
   register the information. Finally the vector features are
   generated.
val allFiles = sc.parallelize(new
   io.File("./").listFiles.filter(_.getName.endsWith(".html"))
   .map(_.getCanonicalFile))
val Array(allTrainFiles, allTestFiles) =
   allFiles.randomSplit(Array(0.8,0.2))
val allTrainData = processData(allTrainFiles)
val allTestData = processData(allTestFiles)
println("\n\n\n>>>>>>> Total Files: " + allFiles.count)
println(">>>>>>> Number Train Files: " + allTrainFiles.count)
println(">>>>>>> Number Test Files: " + allTestFiles.count)
println(">>>>>>> Number Train Lines: " + allTrainData.count)
println(">>>>>>> Number Test Lines: " + allTestData.count)
val featuresVectorTrain = createFeaturesVector(allTrainData)
val featuresVectorTest = createFeaturesVector(allTestData)

/// ------------------- Testing Scenarios
// It prints the actual folder and enter in a loop to test all
   vectors for this dataset. The training and test data are gathered
   from the vector according actual experiment.
println("Files From: " + new java.io.File(".").getCanonicalPath)
for (x <- 0 to featuresVectorTrain.size-1) {
   val trainingData = featuresVectorTrain(x)
   val testData = featuresVectorTest(x)
   val k = if (allFiles.count > 100) {5} else {3}
   val cvData = MLUtils.kFold(trainingData, k, 0)
   val numClasses = 2
   val catFeature = Map[Int, Int]()

   // Evaluation to find the best model for DT and RF
   val modelTuningDT = for ((cvTrain, cvVal) <- cvData;
      impurity <- Array("gini", "entropy");
      depth <- Array(5, 10, 20);
      bins <- Array(50, 100, 200, 300)) yield {
      val model = DecisionTree.trainClassifier(cvTrain, numClasses,
         catFeature, impurity, depth, bins)
      val predicLabels = cvVal.map(example => (
         model.predict(example.features), example.label))
      val areaUnderPR = new
         BinaryClassificationMetrics(predicLabels).areaUnderPR
   ((impurity, depth, bins), areaUnderPR)}
```

```
   val bestDTImpurity = modelTuningDT.maxBy(_._2)._1._1
   val bestDTDepth = modelTuningDT.maxBy(_._2)._1._2
   val bestDTBins = modelTuningDT.maxBy(_._2)._1._3

   val modelTuningRF = for ((cvTrain, cvVal) <- cvData;
      trees <- Array(5, 10, 20);
      impurity <- Array("gini", "entropy");
      depth <- Array(5, 10, 20);
      bins <- Array(50, 100, 200, 300)) yield {
      val model = RandomForest.trainClassifier(cvTrain, numClasses,
         catFeature, trees, "auto", impurity, depth, bins)
      val predicLabels = cvVal.map(example => (
         model.predict(example.features), example.label))
      val areaUnderPR = new
         BinaryClassificationMetrics(predicLabels).areaUnderPR
   ((trees, impurity, depth, bins), areaUnderPR)}

   val bestNuTrees = modelTuningRF.maxBy(_._2)._1._1
   val bestRFImpurity = modelTuningRF.maxBy(_._2)._1._2
   val bestRFDepth = modelTuningRF.maxBy(_._2)._1._3
   val bestRFBins = modelTuningRF.maxBy(_._2)._1._4

   // Based on Model Tuning, training the classifier
   val modelDT = DecisionTree.trainClassifier(trainingData,
      numClasses, catFeature, bestDTImpurity, bestDTDepth,
         bestDTBins)
   val modelRF = RandomForest.trainClassifier(trainingData,
      numClasses, catFeature, bestNuTrees, "auto", bestRFImpurity,
      bestRFDepth, bestRFBins)

   // Predicting and evaluating the final model
   val predicLabelsDT = testData.map {
      case LabeledPoint(label, features) =>
      val prediction = modelDT.predict(features)
      (prediction, label)}
   val predicLabelsRF = testData.map {
      case LabeledPoint(label, features) =>
      val prediction = modelRF.predict(features)
      (prediction, label)}
   val metricsDT = new BinaryClassificationMetrics(predicLabelsDT)
   val metricsRF = new BinaryClassificationMetrics(predicLabelsRF)

   // Printing Metrics - Results
   println("\n\n====== Printing Results Scenario " + (x.toInt + 1))
   println("Precision DT: " + metricsDT.precisionByThreshold.max._2)
   println("Precision RF: " + metricsRF.precisionByThreshold.max._2)
   println("Recall DT: " + metricsDT.recallByThreshold.max._2)
   println("Recall RF: " + metricsRF.recallByThreshold.max._2)
   println("F-Measure DT: " + metricsDT.fMeasureByThreshold.max._2)
   println("F-Measure RF: " + metricsRF.fMeasureByThreshold.max._2)
   println("AUC-PR DT: " + metricsDT.areaUnderPR)
   println("AUC-PR RF: " + metricsRF.areaUnderPR)
   println("AUC-ROC DT: " + metricsDT.areaUnderROC)
   println("AUC-ROC RF: " + metricsRF.areaUnderROC)
}
```

Additional Experiment

Apart from the dataset and features vector presented in this study, we have also experimented in the beginning, the same methodology for one dataset with data from two different sources. The goal of this test is to ensure the needs to train correctly. We have tested the dataset below in one Decision Tree classifier:

Dataset Test Different Structure: This dataset has completely different data in train and test data. It is split in folders (not randomly split), different that we do with the others. Test data is composed by 40 files from LuxTimes and 40 from BBC. While, training data is composed by 40 HTML files from CNN and 40 from New York Times.

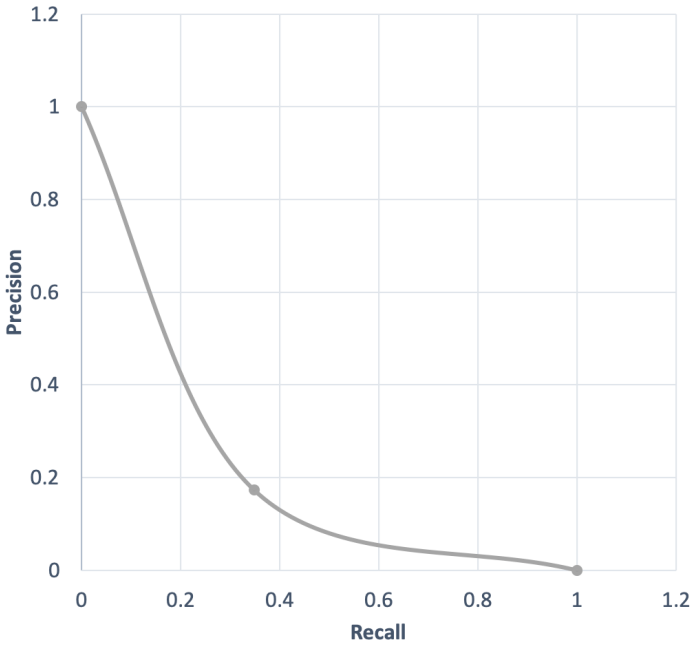


Figure 6.1: P-R Curve for Testing Structures

Figure 6.1 shows the Precision-Recall curve for the previous scenario. We highlight that the area under the curve has the Precision under 20%. It is intentionally very low. We want to prove that is highly necessary to train the data according the structure of the test data. It does not work because the features vector has information from one structure of websites and we want to test another one. As we proved in this study, good results are related with precision around 90% or higher.

Result Tables

We present in this section all numerical results which we have graphically presented in chapter 5.

	Vector	Auto DT	Auto RF	Manual DT	Manual RF
BBC	Vec01	0,957219251	0,988826816	0,896551724	0,928571429
	Vec02	0,92746114	0,983695652	0,896551724	0,928571429
	Vec03	0,966850829	0,973404255	0,958333333	0,96
	Vec04	0,957219251	0,978378378	0,896551724	0,896551724
	Vec05	0,956989247	0,989010989	0,896551724	0,896551724
	Vec06	0,960674157	0,956284153	0,925925926	0,896551724
	Vec07	0,962566845	0,989247312	0,925925926	0,928571429
CNN	Vec01	0,955607477	0,940836941	0,958847737	0,939655172
	Vec02	0,868832732	0,851157663	0,935897436	0,966101695
	Vec03	0,948893974	0,938672439	0,958847737	0,939516129
	Vec04	0,952416918	0,93576741	0,958677686	0,914728682
	Vec05	0,918201916	0,911023622	0,9375	0,948717949
	Vec06	0,786782061	0,796782388	0,924444444	0,917391304
	Vec07	0,753403933	0,75	0,943319838	0,950207469
Lux	Vec01	0,98089172	0,980519481	0,971428571	0,944444444
	Vec02	0,98089172	0,98125	0,96969697	0,903225806
	Vec03	0,975308642	0,987179487	0,970588235	0,972222222
	Vec04	0,98089172	0,97515528	0,972222222	0,972222222
	Vec05	0,95483871	0,957317073	0,896551724	0,962962963
	Vec06	0,84939759	0,868263473	0,813953488	0,8
	Vec07	0,968553459	0,967741935	0,971428571	0,972222222
NYT	Vec01	0,970052083	0,983094928	0,979757085	0,965517241
	Vec02	0,966423358	0,974963181	0,957264957	0,967479675
	Vec03	0,963494133	0,976439791	0,975806452	0,963855422
	Vec04	0,968627451	0,976683938	0,928251121	0,975708502
	Vec05	0,956241956	0,972868217	0,979423868	0,975609756
	Vec06	0,905013193	0,963788301	0,928229665	0,987551867
	Vec07	0,972045743	0,984395319	0,95	0,984126984

Table 6.1: Results for Precision

	Vector	Auto DT	Auto RF	Manual DT	Manual RF
BBC	Vec01	0,962365591	0,951612903	0,928571429	0,928571429
	Vec02	0,962365591	0,97311828	0,928571429	0,928571429
	Vec03	0,940860215	0,983870968	0,821428571	0,857142857
	Vec04	0,962365591	0,97311828	0,928571429	0,928571429
	Vec05	0,956989247	0,967741935	0,928571429	0,928571429
	Vec06	0,919354839	0,940860215	0,892857143	0,928571429
	Vec07	0,967741935	0,989247312	0,892857143	0,928571429
CNN	Vec01	0,71754386	0,762573099	0,732704403	0,685534591
	Vec02	0,422222222	0,451461988	0,229559748	0,179245283
	Vec03	0,72748538	0,760818713	0,732704403	0,732704403
	Vec04	0,737426901	0,809356725	0,729559748	0,742138365
	Vec05	0,728654971	0,676608187	0,70754717	0,698113208
	Vec06	0,584795322	0,550292398	0,65408805	0,663522013
	Vec07	0,58245614	0,642105263	0,732704403	0,720125786
Lux	Vec01	0,93902439	0,920731707	0,944444444	0,944444444
	Vec02	0,93902439	0,957317073	0,888888889	0,777777778
	Vec03	0,963414634	0,93902439	0,916666667	0,972222222
	Vec04	0,93902439	0,957317073	0,972222222	0,972222222
	Vec05	0,902439024	0,957317073	0,722222222	0,722222222
	Vec06	0,859756098	0,884146341	0,972222222	0,777777778
	Vec07	0,93902439	0,914634146	0,944444444	0,972222222
NYT	Vec01	0,93125	0,945	0,889705882	0,926470588
	Vec02	0,8275	0,8275	0,823529412	0,875
	Vec03	0,92375	0,9325	0,889705882	0,882352941
	Vec04	0,92625	0,9425	0,761029412	0,886029412
	Vec05	0,92875	0,94125	0,875	0,882352941
	Vec06	0,8575	0,865	0,713235294	0,875
	Vec07	0,95625	0,94625	0,908088235	0,911764706

Table 6.2: Results for Recall

	Vector	Auto DT	Auto RF	Manual DT	Manual RF
BBC	Vec01	0,959785523	0,969863014	0,912280702	0,928571429
	Vec02	0,944591029	0,978378378	0,912280702	0,928571429
	Vec03	0,953678474	0,978609626	0,884615385	0,905660377
	Vec04	0,959785523	0,97574124	0,912280702	0,912280702
	Vec05	0,956989247	0,97826087	0,912280702	0,912280702
	Vec06	0,93956044	0,948509485	0,909090909	0,912280702
	Vec07	0,965147453	0,989247312	0,909090909	0,928571429
CNN	Vec01	0,819639279	0,842377261	0,830659537	0,792727273
	Vec02	0,568280205	0,589988536	0,368686869	0,302387268
	Vec03	0,823568355	0,840439276	0,830659537	0,823321555
	Vec04	0,83124588	0,867983694	0,828571429	0,819444444
	Vec05	0,812520378	0,776510067	0,806451613	0,804347826
	Vec06	0,6709158	0,650985818	0,76611418	0,770072993
	Vec07	0,656992084	0,691871456	0,824778761	0,819320215
Lux	Vec01	0,959501558	0,949685535	0,957746479	0,944444444
	Vec02	0,959501558	0,969135802	0,927536232	0,835820896
	Vec03	0,969325153	0,9625	0,942857143	0,972222222
	Vec04	0,959501558	0,966153846	0,972222222	0,972222222
	Vec05	0,927899687	0,957317073	0,8	0,825396825
	Vec06	0,854545455	0,876132931	0,886075949	0,788732394
	Vec07	0,953560372	0,940438871	0,957746479	0,972222222
NYT	Vec01	0,950255102	0,963671128	0,93256262	0,945590994
	Vec02	0,891582492	0,895199459	0,885375494	0,918918919
	Vec03	0,943203574	0,953964194	0,930769231	0,921305182
	Vec04	0,946964856	0,959287532	0,836363636	0,928709056
	Vec05	0,942295498	0,956797967	0,924271845	0,926640927
	Vec06	0,880616175	0,911725955	0,806652807	0,927875244
	Vec07	0,964083176	0,964945825	0,928571429	0,946564885

Table 6.3: Results for F-Measure

	Vector	Auto DT	Auto RF	Manual DT	Manual RF
BBC	Vec01	0,959994734	0,970479975	0,864909236	0,895785379
	Vec02	0,945115678	0,978551475	0,864909236	0,895785379
	Vec03	0,954173441	0,978724317	0,873710898	0,892183004
	Vec04	0,959994734	0,975892838	0,864909236	0,864909236
	Vec05	0,957220461	0,978549873	0,864909236	0,864909236
	Vec06	0,940448024	0,948890103	0,876888576	0,864909236
	Vec07	0,965327801	0,989305115	0,876888576	0,895785379
CNN	Vec01	0,849986374	0,862977788	0,843382629	0,806831568
	Vec02	0,672959729	0,677353806	0,61192716	0,608579032
	Vec03	0,851128371	0,86110164	0,843382629	0,826634648
	Vec04	0,857388591	0,8816136	0,841877159	0,809027119
	Vec05	0,836311606	0,809170192	0,814289191	0,819839322
	Vec06	0,705502152	0,694889014	0,780969265	0,779015698
	Vec07	0,687754559	0,713045079	0,829929999	0,830517803
Lux	Vec01	0,952203622	0,943239706	0,945611306	0,919376738
	Vec02	0,952203622	0,961160685	0,918158638	0,807534828
	Vec03	0,958197913	0,958299689	0,931897351	0,959302566
	Vec04	0,952203622	0,955196035	0,959302566	0,959302566
	Vec05	0,910208631	0,937738522	0,777864957	0,835052413
	Vec06	0,79263559	0,820280284	0,803232009	0,715778556
	Vec07	0,940241528	0,928139869	0,945611306	0,959302566
NYT	Vec01	0,940648065	0,959213036	0,934432109	0,935824115
	Vec02	0,89295898	0,900762244	0,886729595	0,916878707
	Vec03	0,931132426	0,947354806	0,930699342	0,916444173
	Vec04	0,937137492	0,95190082	0,836201212	0,929103403
	Vec05	0,926260687	0,947658423	0,928076388	0,927507076
	Vec06	0,848700703	0,906472307	0,817772851	0,935696387
	Vec07	0,953290484	0,961021248	0,9135967	0,947674214

Table 6.4: Results for Area Under Precision-Recall Curve

	Vector	Auto DT	Auto RF	Manual DT	Manual RF
BBC	Vec01	0,980949069	0,97574802	0,96371385	0,963904471
	Vec02	0,980773774	0,986471492	0,96371385	0,963904471
	Vec03	0,970254812	0,991789405	0,910523664	0,928380807
	Vec04	0,980949069	0,986442276	0,96371385	0,96371385
	Vec05	0,978260897	0,983812536	0,96371385	0,96371385
	Vec06	0,959472908	0,970196381	0,946047329	0,96371385
	Vec07	0,983666457	0,994565224	0,946047329	0,963904471
CNN	Vec01	0,857023249	0,878770904	0,864703668	0,84045935
	Vec02	0,707767143	0,721589382	0,613955608	0,589292935
	Vec03	0,861687223	0,877801675	0,864703668	0,863879402
	Vec04	0,866780698	0,901763895	0,863131341	0,86744241
	Vec05	0,86092216	0,834837411	0,851300786	0,847078364
	Vec06	0,784083757	0,767783332	0,824241519	0,828628794
	Vec07	0,781226843	0,80982426	0,864044255	0,858084654
Lux	Vec01	0,969131871	0,959985529	0,971613209	0,971004195
	Vec02	0,969131871	0,978278212	0,943835431	0,887061849
	Vec03	0,981200218	0,969258645	0,95772432	0,985502098
	Vec04	0,969131871	0,978151437	0,985502098	0,985502098
	Vec05	0,950332088	0,977771113	0,859284071	0,860502098
	Vec06	0,926708678	0,939284124	0,981239004	0,884625795
	Vec07	0,968878321	0,956683199	0,971613209	0,985502098
NYT	Vec01	0,963763255	0,971447709	0,943129991	0,960133985
	Vec02	0,911888255	0,912373927	0,908318807	0,93474328
	Vec03	0,959608528	0,964792982	0,942785402	0,938075161
	Vec04	0,961182309	0,969792982	0,875001267	0,940947166
	Vec05	0,961622855	0,968925146	0,93577705	0,939108931
	Vec06	0,922921928	0,930395418	0,851448798	0,93646623
	Vec07	0,9763442	0,972153655	0,949564448	0,954503993

Table 6.5: Results for Area Under ROC Curve

Bibliography

[1] Apache spark. https://spark.apache.org/, Last accessed on 2019-04-15.

[2] Cluster mode overview - spark 2.4.1 documentation. https://spark.apache.org/docs/latest/cluster-overview.html, Last accessed on 2019-04-16.

[3] Decision trees - rdd-based api - spark 2.2.0 documentation. https://spark.apache.org/docs/2.2.0/mllib-decision-tree.html, Last accessed on 2019-04-15.

[4] Evaluation metrics - rdd-based api - spark 2.2.0 documentation. https://spark.apache.org/docs/2.2.0/mllib-evaluation-metrics.html, Last accessed on 2019-04-15.

[5] Ml tuning - spark 2.2.0 documentation. https://spark.apache.org/docs/2.2.0/ml-tuning.html, Last accessed on 2019-04-15.

[6] Rdd programming guide - spark 2.4.1 documentation. http://spark.apache.org/docs/latest/rdd-programming-guide.html, Last accessed on 2019-04-15.

[7] Apache ant, 1999. http://ant.apache.org/, Last accessed on 2019-03-10.

[8] Apache nutch, 2004. http://nutch.apache.org/, Last accessed on 2019-04-11.

[9] Nutch documentation, 2004. https://wiki.apache.org/nutch/NutchTutorial, Last accessed on 2019-04-11.

[10] Jsoup, 2009. https://jsoup.org/, Last accessed on 2019-04-11.

[11] Apache solr, 2017. http://lucene.apache.org/solr/, Last accessed on 2019-04-11.

[12] Github repository for: Advanced content analysis for web pages in apache spark, 2019. https://github.com/drendrin88/master-thesis.

[13] Vineeth N. Balasubramanian, Shen-Shyang Ho, and Vladimir Vovk. *Conformal prediction for reliable machine learning : theory, adaptations and applications*. Morgan Kaufmann, San Francisco, 2014.

[14] Bernard Berelson. *Content analysis in communication research*. Free Press, 1952.

[15] Bernard Berelson and Paul F. Lazarsfeld. *The analysis of communication content*. University of Chicago and Columbia University, 1948.

[16] Jian Chang, Krishna Venkatasubramanian, Andrew West, Sampath Kannan, Oleg Sokolsky, Myuhng Kim, and Insup Lee. Tomato: A trustworthy code mashup development tool. 09 2011.

[17] Rajan Chattamvelli. *Data Mining Methods*. Alpha Science International, 2009.

[18] Michael Chau and Hsinchun Chen. A machine learning approach to web page filtering using content and structure analysis. *Decision Support Systems*, 44:482–494, 2008.

[19] Claudia Elena Dinucă and Dumitru Ciobanu. Web content mining. (1):85–92, 2012.

[20] Massimo Franceschet. Pagerank: Standing on the shoulders of giants. *Commun. ACM*, pages 92–101, 2011.

[21] Trevor Hastie. The elements of statistical learning : data mining, inference, and prediction, 2009.

[22] Susan Herring. Web content analysis: Expanding the paradigm. In *International Handbook of Internet Research*, pages 233–249. Springer Science, 2010.

[23] Rohit Khare, Doug Cutting, Kragen Sitaker, and Adam Rifkin. Nutch: A flexible and scalable open-source web search engine. *Oregon State University*, 1:32–32, 2004.

[24] Inhwa Kim and Jasna Kuljis. Applying content analysis to web-based content. In *CIT*, volume 18, pages 283–288, 2010.

[25] Klaus Krippendorff. *Content Analysis: An Introduction to Its Methodology*. Sage Publications, 1980.

[26] Zakir Laliwala and Abdulbasit Shaikh. *Web crawling and data mining with Apache Nutch*. Packt Publishing, 2013.

[27] Lucia Helena Magalhaes. Uma análise de ferramentas para mineração de conteúdo de páginas web. *COPPE/UFRJ - Master Thesis*, 06 2008.

[28] R. Malarvizhi and K. Saraswathi. Web content mining techniques tools algorithms – a comprehensive study. In *International Journal of Computer Trends and Technology (IJCTT)*, volume 4, pages 2940–2945. Seventh Sense Research Group, 2013.

[29] David L Olson. Advanced data mining techniques, 2008.

[30] James W. Perry, Allen Kent, and Madeline M. Berry. Machine literature searching x. machine language; factors underlying its design and development. *American Documentation*, 6(4):242–254, 1955.

[31] Jyotika Prasad and Andreas Paepcke. Coreex: Content extraction from online news articles. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, New York, NY, USA, 2008. ACM.

[32] Arnau Ramisa. Multimodal news article analysis. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 5136–5140, 2017.

[33] Lior Rokach. Data mining with decision trees : theory and applications, 2008.

[34] Ruihua Song, Haifeng Liu, Ji-Rong Wen, and Wei-Ying Ma. Learning important models for web page blocks based on layout and content analysis. *SIGKDD Explor. Newsl.*, 6(2):14–23, December 2004.

[35] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos. Management of an academic hpc cluster: The ul experience. In *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*, pages 959–967, Bologna, Italy, July 2014. IEEE.

[36] Shanchan Wu, Jerry Liu, and Jian Fan. Automatic web content extraction by combination of learning and grouping. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1264–1274, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.

[37] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016.