Processing time of TFIDF and Naive Bayes on Spark 2.0, Hadoop 2.6 and Hadoop 2.7: Which Tool Is More Efficient?

MSc Research Project
Data Analytics

Erin Gilheany
x14106671

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet – 2015/2016
School of Computing

Student Name:	Erin Gilheany
Student ID:	x14106671
Programme:	Data Analytics
Year:	2016
Module:	MSc Research Project
Lecturer:	Vikas Sahni
Submission Due Date:	05/09/2016
Project Title:	Processing time of TFIDF and Naive Bayes on Spark 2.0, Hadoop 2.6 and Hadoop 2.7: Which Tool Is More Efficient?
Word Count:	5251

Signature:	
Date:	13th September 2016

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Processing time of TFIDF and Naive Bayes on Spark 2.0, Hadoop 2.6 and Hadoop 2.7: Which Tool Is More Efficient?

Erin Gilheany
x14106671
MSc Research Project in Data Analytics

13th September 2016

Abstract

There has been a large emphasis placed on the performance variations which occur when comparing Hadoop and Spark. This research paper will dive into the details of this comparison using TD-IDF and Naive Bayes algorithms on both applications to demonstrate the total processing time differences. It has been noted in literature that Spark goes a long way towards dealing with the limitations of Hadoop, in particular those issues which frequently arise in the application of iterative machine learning algorithms due to the slow processing of inputs/outputs to disc. This paper explores the difference from a text categorisation stand point. On a single computer there is a strong distinction in computing times of TFIDF on Spark Versus Hadoop, with Spark completing the application in a fraction of the time. Naive Bayes shows a contrasting picture with Spark's processing speeds on average twice as big as that of Hadoop. Given the additional costs of RAM on Spark, in this instance Hadoop would appear to be the better choice.

1 Introduction

1.1 Big Data and the Need for Efficiency and Flexibility

Big Data applications have fast become an immensely important area in the world of both science and business. The advancements in data storage cost minimizations has allowed for widespread adoption of distributed data storage. Frameworks have been developed to process this distributed data in a parallel and efficient manner. Flexibility has also become a focus which allows for processing of both structured, semi-structured and unstructured data.

For businesses, the ability to understand and efficiently utilise tools within this area offers a distinct advantage against their competitors. This is due to the fact that it leads to quicker and deeper insights relating to their product, from which they can make quick and informed decisions to advance their place in the market. In particular in the age of the Internet it has become extremely important to extract knowledge and patterns from all types of data, i.e. text, images, audio, tables, html, etc.

Similarly in the scientific community, advances that would have been difficult to achieve in the past have become much more attainable as a result of the technological leaps which have been taken within the last number of years and decades.

Assessing performance of tools in this area will therefore contribute more informed decisions when adopting these as well as identifying limitations within these tools that can lead to further development.

1.2 Research Objectives

The other side of this paper deals with not simply the technology used but also the algorithms which can be used to derive a predictive model to classify textual data. In this paper we will assess a process of classifying a document of text data, which sits in a larger collection of documents, into the most relevant category that fits each document. In order to do so we will perform the algorithm TFIDF (Term Frequency Inverse Document Frequency) and then a classification technique, multinomial Naive Bayes on two different versions of Hadoop and also the latest version of Spark.

The goal of this paper is to provide the measure of efficiency when dealing with a similar type of text classification process and to dig further into the individual steps within each environment which cause variations in performance. This will enable readers to determine the value of efficiency for large text classification problems in trade-offs which may arise when choosing between the two environments (i.e. cost trade-offs, knowledge base trade-offs, etc.).

TFIDF does not appear to have been compared on both applications in literature so this will offer new insights. Although we do see a comparison of a Naive Bayes algorithm in (Jiang et al.; 2014), this paper will offer a more specific implementation of the algorithm for comparison on the two applications.

1.3 Paper Overview

This paper will set out by providing an overview of information relating to any relevant features or topics carried out in our research experiment, most importantly giving an overview of the core components and make up of Apache Hadoop and Spark. A review of the published research that has occurred in relation to performance indicators of Spark Vs Hadoop, particularly for a number of iterative algorithms along with other processing comparisons, will follow. The research methodology and research experiment specification will be detailed clearly, giving reasons as to algorithms chosen and experiment design choices, etc. After this the paper presents experiment implementation information and evaluation of the results that were generated, discussing limitations and findings throughout.

2 Background

2.1 Big Data Open Source Solutions

There have been a multitude of tools that sit within the realm of Big Data applications, most of which have a common theme of distributed/parallel computing. While focusing mainly on the open source tools available, Hadoop, Spark, Storm, H20 and Flink are prominent with many other tools available which work in conjunction with these distributed frameworks including Kafka, Mahout, Mesos, Hive, Sqoop, Cassandra, etc. It is therefore very important for both industry and academia alike to understand the appropriate and best performing tools to use for any given problem.

2.2 Further Developments To Be Made

Much research has been carried out in the realm of data storage and querying. NoSQL databases have been developed which deal adequately with the three Vs of Big Data: Velocity, Variety and Volume (Russom; 2011). However somewhat lacking in the area of Big Data advancements, in particular within Hadoop's MapReduce programming paradigm, sits the area of machine learning algorithms. Although we can store large volumes of unstructured data, there are still improvements to be made when using techniques to draw predictive information from this datasets. The trend has been to move away from Hadoop as it is seems to be the common conception that Hadoop is not built in such a way that it can adequately tackle this type of machine learning problem.

2.3 Hadoop Vs Spark

Because of these limitations, we will make direct comparisons in processing time measurements between two of the leading open source tools, Hadoop and Spark, via building a model for classifying unstructured data held within documents. Hadoop is based on Google's big table, (Chang et al.; 2006) which uses a programming paradigm known as Map Reduce. Data is stored in a file system known as HDFS on top of distributed hard disks and algorithms are processed on these distributed devices so there is less reliance on one large computing server. Spark works in a different way and makes use of RDDs (Resilient Distributed Datasets) which temporarily store data in distributed memory making the data quicker to access and write, decreasing the computing time significantly as a result. Spark and Hadoop are both capable of maintaining the computing standards of fault tolerance (Zaharia et al.; 2012), resilient and consistency.

Although Hadoop was the original open source "Big Data" distributed computing framework, users have found limitations which occur for applications with heavy read or write proportions to their workflow. This is in fact the case for many popular data mining algorithms, which access the data from storage on an iterative basis. Spark is reportedly more efficient than Hadoop. However, it does typically incur a higher cost, as Spark requires RAM as opposed to Hadoop's less expensive hard disks. So, depending on the workload and urgency of output it may be more optimal to use Hadoop. This assessment is outside of the remit of this paper.

2.4 Machine Learning Tools: Mahout on Hadoop and Spark's MLlib

Mahout is an open source Apache application, built specifically to work with Hadoop, which implements machine learning algorithms on Hadoop using its MapReduce programming framework. It also has a number of algorithms that can work with Spark. However, these appear to be redundant and Spark's inbuilt machine learning tool MLlib is comprehensive with little mention of Mahout Spark implementations in literature. A limited number of algorithms currently sit within Mahout's repository,[1] with a noticeable lack of algorithms which typically use iterative I/O processes.

MLlib is Spark's answer to Apache Mahout, although it is built within Spark itself and does not require a separate download and install. MLlib is described in detail by (Meng et al.; 2015) as:

"The library targets large-scale learning settings that benefit from data-parallelism or model-parallelism to store and operate on data or models. MLlib consists of fast and scalable implementations of standard learning algorithms for common learning settings including classification, regression, collaborative filtering, clustering, and dimensionality reduction."

2.5 Caching in Spark

Caching is described in (Gu and Li; 2013) and similarly (Zaharia et al.; 2010) as a mechanism which signals that an RDD may be used at a later stage of the process and therefore keeps this particular RDD stored in memory, if there is sufficient space. We will use this in our experiment process when performing our TFIDF algorithm, in between calculating our term frequency and our inverse term frequency as the output of the former is the input for the later.

3 Related Work

There have been a number of papers which have compared the two applications of Hadoop and Spark across many of the main areas, including database querying, graph processing and machine learning models. It has been identified that one of the main reasons for such a large difference between the two applications is due to the speed of access to the data, this becomes an even larger problem when you need to access/write data multiple times in the one process which is the case with iterative machine learning algorithms. In the case of Hadoop, this is much slower as the data has to be accessed directly from the disk, whereas Spark temporarily stores the necessary data in memory as RDDs. Spark has four core components: machine learning (MLlib), graph processing (GraphX), relational database (Spark SQL), and stream processing (Spark Streaming). We will look at literature which compares Spark's machine learning, graph processing and relational database performance with Hadoop's equivalent tools to gain an insight in to how these behave differently for various use cases. We will not review streaming as this does not apply for Hadoop. It is used as a batch processing application.

3.1 Relational Database Querying Compared

There are a number of tools that implement a relational database in Spark including Shark and Spark SQL (Armbrust et al.; 2015), therefore a one on one comparison to its equivalent in Hadoop (i.e. Hive) may not be possible. Spark SQL is the newer improved tool of the two technologies and Shark is currently being dissolved to allow for full adoption of Spark SQL.

[1]Mahout Algorithms Available(subject to change): `http://mahout.apache.org/users/basics/algorithms.html`

Unfortunately, there does not appear to be a study that includes Shark, Spark SQL and Hive together. However, we can see from (Shenker et al.; 2013) that there is a very large difference between the Hadoop based, Hive, and Spark based, Shark, relational databases with Shark's queries returning up to 100 times quicker than Hadoop's Hive. These queries are not described in detail, so it is difficult to distinguish the extent to which this difference is as a result of sped up read/write access or computing speed. We can also see this comparison made in (Armbrust et al.; 2015) where a simple filtered select query runs almost 100 times quicker than Hive. There is no suggestion that this query is recursive and so would not likely benefit from the advantages of cached data, as data would only have to be called once. This leaves room for inquiry as to what other mechanisms are leading to this large variation in computing time, which may or may not be specific to relational database querying within Hadoop and Spark.

3.2 Graph Processing

The PageRank algorithm, which is described in detail in (Gu and Li; 2013), is commonly used as a means of comparison between Hadoop and Spark. In the papers (Liang and Lu; 2015), (Lin et al.; 2013), (Gu and Li; 2013), (Jiang et al.; 2014) we can see a consistent stark difference between the total processing time of PageRank, an iterative algorithm. The paper (Gu and Li; 2013) solely analyses the total processing time and does not prove exactly at what stage of the process the main differences lie between the two programs, which would be a useful addition.

3.3 Machine Learning Algorithms

In (Kupisz and Unold; 2015) we see that the main algorithm used is a similarity measure (Tanimoto coefficient) between particular items. This goes contrary to typical collaborative algorithms where information or similarity between items is typically unknown and recommendation is based purely on preference of users, who have expressed interest into the same items as yourself. This study is identified as limited in experimental input volume. Many papers (Zaharia et al.; 2012) (logistic regression) (Lin et al.; 2013) (Kmeans) (Jiang et al.; 2014) (Naive Bayes) demonstrate a decided improvement in execution time performance for algorithms on Spark from Hadoop.

An interesting occurrence is shown in the paper (Gu and Li; 2013) which compares algorithms on Hadoop and Spark, that for lower volumes of data Spark in most cases outperforms Hadoop, but when data becomes too large to fit in memory Spark's performance per byte of data slows down and Hadoop shows more efficiency in this regard. This can be somewhat managed by tuning Spark and optimizing memory management. This is something we will have to consider in our implementation. Ensuring sufficient memory is available to Spark will be needed for this experiment as will be described later in the paper.

4 Project Specification

4.1 Dataset

Text classification is a component of many use cases, such as in recommender systems (Mooney and Roy; 2000), spam detection (Ntoulas et al.; 2006), search engine precision

(Chekuri et al.; 1997), etc. In this research we will explore the process of developing a model to classify news items into given categories based solely on the text within. This requires a process of first performing an algorithm TF-IDF and then Naive Bayes on the TF-IDF output. This dataset specifically is 20newsgroup which was collected for use in the study (Lang; 1995) and is found at [2].

The twenty categories of news into which the data is split, the size of the dataset, the combined size of documents within each category, along with the volume of documents for each category and overall volumes are shown in Table 1. No document belongs to more than one category, eradicating any need for training multi-label items (Zhang et al.; 2009). This data is split into training and test directories. Each directory has a subdirectory with the twenty news categories and within each of these is a number of files, each containing just one piece of text, referred to as a document.

For our experiment, we will be combining the test and training data as we will only be running the training of the model, therefore there is no need for separate testing data and we will be able to run the model on a larger dataset as a result.

4.2 Tools Comparisons: Hadoop 2.6, 2.7 and Spark 2.0

This experiment will assess the performance enhancements that were made in Hadoop by comparing Hadoop's latest versions on the 2.6 and 2.7 release lines. A component update was introduced with the 2.7 line: "MAPREDUCE-4815 - Speed up FileOutputCommitter for very large jobs with many output files". [3] This has an impact when running iterative algorithms as mentioned above and so we will assess the effect this improvement makes to the overall runtime of the iterative algorithms, TDIDF and Naive Bayes. Further to this, we will add into our comparisons the latest Spark release, 2.0.0, and analyse whether the update to Hadoop's I/O writing makes any real difference to the variation in performance to Spark's RDD framework.

4.3 Algorithms Chosen

Unsurprisingly, due to limitations of iterative processing on Hadoop as discussed in previous sections of this paper, there is a limited number of classification algorithms, and machine learning algorithms in general, that are readily available in Mahout. They do not prove to be scalable due to their iterative nature and also issues due to inefficient read/writes. In fact, there are only three classification algorithms listed in Mahout's webpage.[4] This is not the case with Spark. There are many classification algorithms built in MLlib.[5] We have therefore chosen the following algorithms for comparison of the tools:

→TFIDF
→Naive Bayes

[2]Dataset: `http://qwone.com/~jason/20Newsgroups/`

[3]Hadoop Release Notes: `http://hadoop.apache.org/releases.html`

[4]Mahout Algorithms Available(subject to change): `http://mahout.apache.org/users/basics/algorithms.html`

[5]MLlib Algorithms Available(subject to change): `http://spark.apache.org/docs/latest/mllib-guide.html`

Table 1: 20NewsGroups Dataset

Category	**No. of Documents**	**Size of Collection / Sub Collection(KBs)**
comp.graphics	973	4648
comp.os.ms-windows.misc	985	5260
comp.sys.ibm.pc.hardware	982	4116
comp.sys.mac.hardware	963	4004
comp.windows.x	988	4828
rec.autos	990	4132
rec.motorcycles	996	4100
rec.sport.baseball	994	4212
rec.sport.hockey	999	4472
sci.crypt	991	4672
sci.electronics	984	4100
sci.med	990	4636
sci.space	987	4556
talk.politics.misc	775	4028
talk.politics.guns	910	4252
talk.politics.mideast	940	5168
talk.religion.misc	628	3036
alt.atheism	799	3752
soc.religion.christian	997	4796
misc.forsale	975	4040
Total	18846	86808

4.4 TF-IDF

TF-IDF (Term Frequency Inverse Document Frequency) is a typical text classification algorithm which measures the importance of terms or words within a document relative to those in a collection of documents. It does so by assigning each word a weight between zero and one. In this case, it is being used as a means of feature selection and not actual classification as is the use case in (Joachims; 1996) (i.e. features are selected by choosing frequent words in a text which demonstrate a distinction from words which sit within the wider collection). These features are extracted from the original dataset and are used to feed into the Naive Bayes classification algorithm. The algorithm is described clearly in (Ramos et al.; 2003) as:

"TF-IDF calculates values for each word in a document through an inverse proportion of the frequency of the word in a particular document to the percentage of documents the word appears in."

This goes a long way to giving some form of structure and deriving some useful information from the data. Firstly, it narrows down our input by allowing us to filter out less relevant words. However, it is very dependent on the larger collection of documents to which it sits. Therefore, we cannot necessarily find the most relevant text for documents on an independent basis without a larger context from other documents. Also, the most relevant text derived as a result of this may vary largely depending on the collection it sits within.

4.5 Naive Bayes

Naive Bayes is a classification algorithm that works with probabilities of independent factors, so it may not be as iteratively intensive as other classification algorithms including Decision Trees and Artificial Neural Networks. It is also an algorithm that has not yet been explored extensively in literature when comparing Spark and Hadoop, apart from comparisons made in (Jiang et al.; 2014).

Although the features (i.e. text/words within each document) in our dataset cannot be assumed to be independent of each other, not strictly fitting the "naive Bayes assumption" (McCallum and Nigam; 1998), it still provides relatively strong classification prediction. Because it is a relatively simple algorithm, in terms of calculation intensity, the positives of simplicity are likely to outweigh the negatives in a number of Naive Bayes Text Classification instances. (McCallum and Nigam; 1998) states:

"Because of the independence assumption, the parameters for each attribute can be learned separately, and this greatly simplifies learning, especially when the number of attributes is large."

It also offers an alternative to heavily iterative algorithms, many of which have been used as a benchmark for comparison of Spark and Hadoop in literature.

4.5.1 Naive Bayes Versions

Mahout has two types of Naive Bayes algorithms, Complement Naive Bayes and the standard Multinomial Naive Bayes. Spark's MLlib also has two types of Naive Bayes: Bernoulli and Multinomial. Therefore we will be using multinomial in our testing as we want to ensure consistency, and it is the one type which is common in both applications. A quick look through each of the Naive Bayes variations gives us an insight into how they are implemented and which would be most appropriate to use if readily available in either Spark or Hadoop.

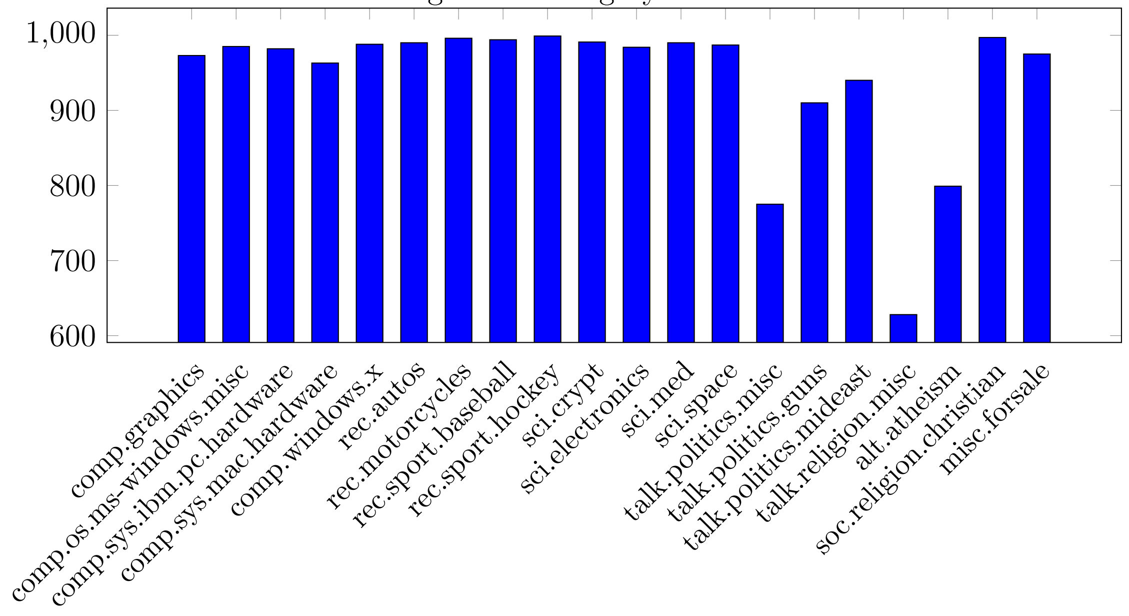
> **Bernoulli** is the original implementation of Naive Bayes. Briefly explained, its basic algorithm operates on features that are considered to be binary. In terms of text classification, this would mean the algorithm would depend only on whether a word appears or not and disregards the frequency or any other weighting associated with a word that presents as a feature. This means that all words in a text would typically have be considered as input and would likely lead to very intensive computing requirements.
>
> **Multinomial** extends the Bernoulli Naive Bayes algorithm by taking into account a weighting factor for each of the features. In text classification this is achieved by calculating term frequencies or more complex transformations such as TFIDF (as is used in this research) as well as other feature selection methods.
>
> **Complement Naive Bayes** introduced in (Rennie et al.; 2003) extends Multinomial Naive Bayes further by tackling issues caused by bias which manifests from imbalances in class volumes. It does so by altering the way in which weights are estimated. Instead of using data from one class as is the case in Multinomial Naive Bayes it uses all data but that one class for weight estimation. This means weights are estimated using a higher volume of data in most cases and alleviates bias. This also suggests a more time-intensive method than multinomial as a result.

In this case multinomial is in fact the most appropriate because it gives us better accuracy than Bernoulli, as is shown in (McCallum and Nigam; 1998), for a larger set of vocabulary which is the case with our 20newsgroups dataset along with the fact that our output from the TF-IDF algorithm provides an appropriate input to the multinomial model. Opposite to this is the Bernoulli model which just takes into account whether a word appears or not, and its relative importance in a document is not considered. On the other side, there is no need for Complement Naive Bayes as our dataset has evenly balanced categories/classes as demonstrated in Figure 1.

4.6 Methodology

For our experiment we will run each algorithm three consecutive times and then record the performance metrics to obtain the average processing time on each software tool. For Spark, this will require an extra step as the initial algorithm test will have to go through more steps than any consecutive runs due to the caching mechanism. Therefore, for comparison purposes, I will initially run the algorithm and then record three iterations after the initial one, so there will be four in total. The initial Spark algorithms steps will be recorded for review but simply not used in its comparison against Hadoop. When running our tests it has been ensured that Spark and Hadoop are the only applications running on either the virtual machine or the host machine for each test to ensure no competition for resources is occurring in the background.

Figure 1: Category Volumes



5 Implementation

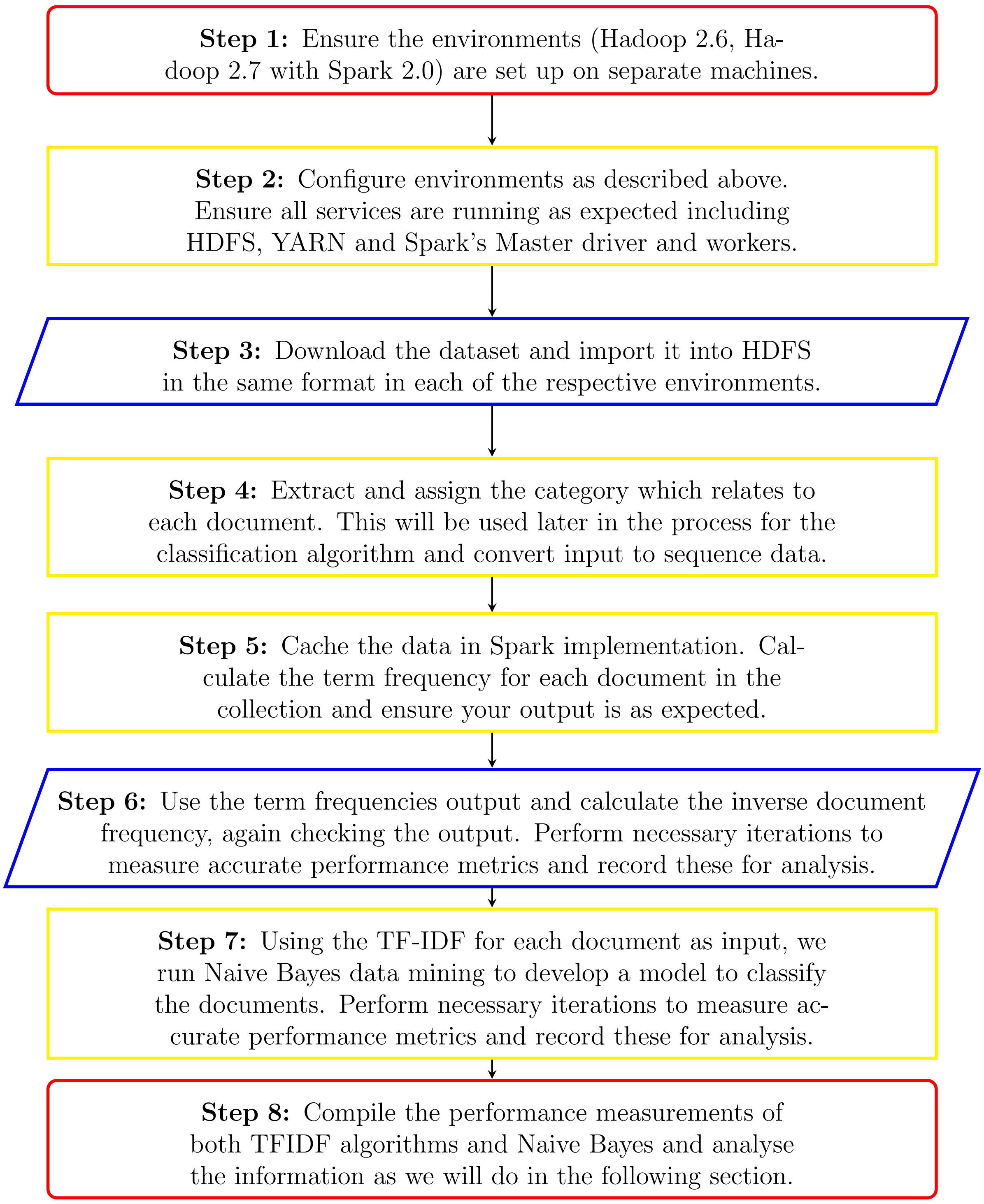
5.1 Environment and Configuration

These experiments were set up using two virtual machines, specifically two Oracle VM VirutalBoxes [6], each given 4GB of memory on the same 6GB windows 64 bit host machine. Each virtual machine was allocated 20GB of disk space. Both machines' operating systems were set up using Ubuntu 14.04.1 and Java version 7 was installed. Hadoop 2.6.4 was set up on one virtual machine whereas Hadoop 2.7.3 and Spark 2.0.0 were installed together on a separate identical virtual machine as Spark 2.0.0 sits on top of Hadoop 2.7.3.

Hadoop's set up included resource manager yarn in pseudo distributed mode with one data node in operation. The additional machine learning application Apache Mahout, which works in conjunction with Hadoop, was utilized for running algorithms. Mahout Version 0.12.2 was used with both versions of Hadoop. On the Spark implementation, the tests were run using spark shell with master set to Hadoop's yarn and executor memory increased from default of 1GB to 3GB as I ran into out of memory exceptions on Naive Bayes with the default executor memory settings. The algorithms were processed using Spark's in-built machine learning library known as MLlib, whilst implementing this using Scala version 2.11.8.

[6]Oracle VM VirtualBox Download: `https://www.virtualbox.org/wiki/Downloads`

5.2 Process Flow



Step 1: Ensure the environments (Hadoop 2.6, Hadoop 2.7 with Spark 2.0) are set up on separate machines.

Step 2: Configure environments as described above. Ensure all services are running as expected including HDFS, YARN and Spark's Master driver and workers.

Step 3: Download the dataset and import it into HDFS in the same format in each of the respective environments.

Step 4: Extract and assign the category which relates to each document. This will be used later in the process for the classification algorithm and convert input to sequence data.

Step 5: Cache the data in Spark implementation. Calculate the term frequency for each document in the collection and ensure your output is as expected.

Step 6: Use the term frequencies output and calculate the inverse document frequency, again checking the output. Perform necessary iterations to measure accurate performance metrics and record these for analysis.

Step 7: Using the TF-IDF for each document as input, we run Naive Bayes data mining to develop a model to classify the documents. Perform necessary iterations to measure accurate performance metrics and record these for analysis.

Step 8: Compile the performance measurements of both TFIDF algorithms and Naive Bayes and analyse the information as we will do in the following section.

5.3 Process Notes

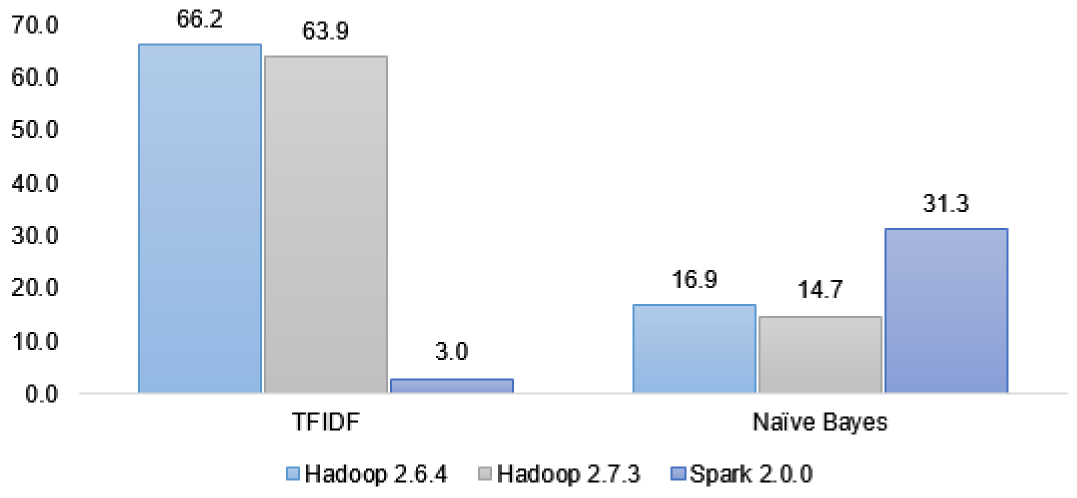
All of these steps from two onwards follow the same process on each tool although there is a slightly different configuration of environment and actual code implementation. These are detailed clearly in the configuration manual.

The data for each environment was transferred into HDFS and operated on from there.

6 Evaluation

The processing time results of the algorithms we ran on the two versions of Hadoop, 2.6.4 and 2.7.3 along with those of Spark are visible from the following chart. As you can see we have grouped each algorithm together, as we are using these as a method/context for comparison of the specified tools.

Figure 2: Processing Time (Seconds)



6.1 Evaluating Hadoop 2.6.4 Vs Hadoop 2.7.3

As expected, due to an absence of fundamental programming alterations, Hadoop 2.6 and 2.7 follow the same processing time trends. However, there is indeed a justified increase in speed for Hadoop 2.7, performing 3.5 percent quicker than Hadoop 2.6 on TFIDF and 13 percent quicker on Naive Bayes. This is likely to be as a result of the update for FileOutputCommitter as discussed in Section 4.2. This would require an experiment that tracks the exact metric before conclusion, which is not covered in this paper.

6.2 Evaluating Hadoop Vs Spark

Interestingly, we can see two contrasting results when we compare Hadoop to Spark. For TFIDF Spark completes the task at a fraction of time of Hadoop's computation time, with Spark spending less than 5 percent of the time it takes either version of Hadoop. Figures here remain consistent across all iterations. Naive Bayes tells a different story with Spark, on average, taking almost twice as long as either version of Hadoop. This varies quite considerably across iterations however, as is visible in Table 2, so more extensive testing may be required before a steady view on the results are achieved.

6.3 Discussion

Drawing from the results above there is no conclusive evidence to suggest that either Hadoop or Spark is more efficient than the other. Memory optimization in Spark is a very important aspect to consider as can cause significant improvements in performance.

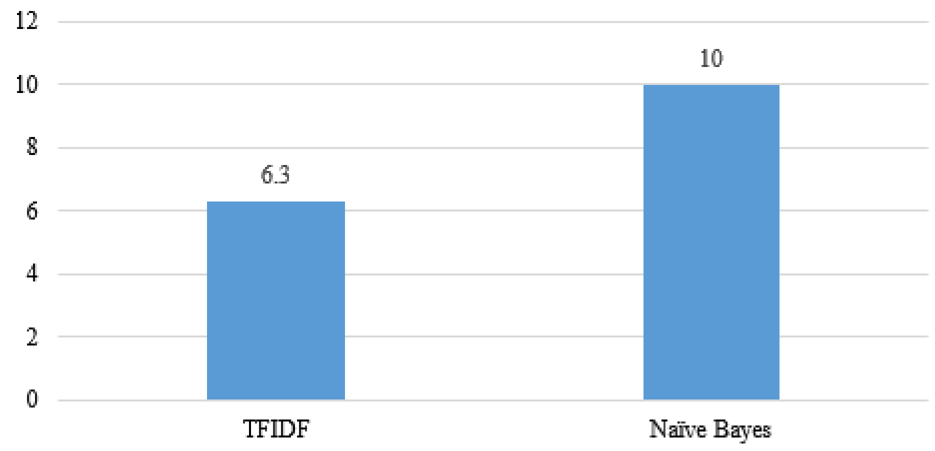
Table 2: Detailed Experiment Results

Algorithm	**Tool**	**Iteration 1**	**Iteration 2**	**Iteration 3**	**Average**
TF-IDF	Hadoop 2.6.4	68.2	65.2	65.4	66.2
TF-IDF	Hadoop 2.7.3	62.1	69.0	60.7	63.9
TF-IDF	Spark 2.0.0	3.0	3.0	3.0	3.0
Naive Bayes	Hadoop 2.6.4	18.1	18.0	14.8	16.9
Naive Bayes	Hadoop 2.7.3	14.8	14.6	14.9	14.7
Naive Bayes	Spark 2.0.0	55.0	25.0	14.0	31.3

We know this from literature that proves once the allocated RAM is reaching its maximum usage, performance slows down considerably. Also, there are limitations to this experiment in that it was performed on a single computer and did not benefit from the advantages of true storage and computing distribution/parallelization and the additional disk memory and RAM that typically comes with a fully distributed system.

When running this experiment there is an overhead of processing time spent collecting data and caching it in memory that is not included in the comparative analysis, see Chart in Section 6.3 This is a factor that a user may also need to consider, which may slow down the overall speed of a process if the data is not consistently cached in memory.

Figure 3: First Run Processing Time on Spark 2.0.0 (mins)



7 Conclusion and Future Work

7.1 Conclusion

In this paper we have tested a text classification process, using TFIDF for feature extraction and Naive Bayes for classification on two of the most widely adopted open source "Big Data" technologies. We have found that Spark does not always perform more efficiently than Hadoop, as Hadoop Naive Bayes returns quicker on the given dataset as per this research's experimentation, whereas TFIDF is much quicker on Spark on the exact same dataset with the same size of input data.

Pseudo distributed clusters are not the intended set-up for tools such as Spark and Hadoop. So, although experimenting on them with this configuration gives us a degree of insight into their performance results, it does not accurately reflect most real life use cases. Thus, conclusions drawn are limited by this.

We conclude that there are many factors for choosing one system over another. For irregular or more ad-hoc explorations of data and model building, Hadoop may prove beneficial due to insignificantly slower times in one off instances and a lack of overhead processing time used for collecting the dataset in memory, as is the case in this paper. However, if one were to choose Spark there are many possibilities for optimization of memory utilization and thus reducing overall processing time. Caching is one of these methods and it was used within this research.

7.2 Future Work

Extensive research included in papers such as (Kibriya, A. M.; Frank, E.; Pfahringer, B; Holmes; 2005), (Informatik and Joachims; 1998), (Dumais et al.; 1998) and (Huang et al.; 2003) has been carried out on the comparative analysis on Classification algorithms from Naive Bayes, Support Vector Machines, Decision Trees, etc. I believe it would be useful to reiterate some of these text classification algorithm experiments but with the perspective of efficiency on Spark only, as many of these are not available to implement on Hadoop.

Future Work will be to look at ways of optimizing both Spark and Hadoop to analyse what kind of difference this can make, as opposed to simply choosing the higher performing basic implementation of each environment. There have been many proposals in research papers relating to optimizing both Hadoop and Spark, one of which relates to optimizing shuffle performance as proposed by (Davidson and Or; 2013).

Ideally, we would like to run these experiments on a fully distributed framework, perhaps using AWS, Bluemix or some other similar infrastructure, rather than a pseudo-distributed mode. This would allow us to analyse larger datasets which can run more efficiently due to an increased number of cores, be that data nodes or worker memory. It would also more accurately reflect implementations which occur in reality.

References

Armbrust, M., Ghodsi, A., Zaharia, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T. and Franklin, M. J. (2015). Spark SQL: Relational Data Processing in Spark, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15* pp. 1383–1394.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R. E. (2006). Bigtable: A distributed storage system for structured data, *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA* pp. 205–218.

Chekuri, C., Goldwasser, M. H., Raghavan, P. and Upfal, E. (1997). Web Search Using Automatic Classification, (June 1999).

Davidson, A. and Or, A. (2013). Optimizing Shuffle Performance in Spark.

Dumais, S., Platt, J., Heckerman, D. and Sahami, M. (1998). Inductive learning algorithms and representations for text categorization, *Proceedings of the seventh international conference on Information and knowledge management - CIKM '98* pp. 148–155.

Gu, L. and Li, H. (2013). Memory or time: Performance evaluation for Iterative Operation on Hadoop and Spark, *Proceedings - 2013 IEEE International Conference on High Performance Computing and Communications, HPCC 2013 and 2013 IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2013* pp. 721–727.

Huang, J., Lu, J. and Ling, C. (2003). Comparing naive Bayes, decision trees, and SVM with AUC and accuracy, *Third IEEE International Conference on Data Mining* pp. 11–14.

Informatik, F. and Joachims, T. (1998). Text Categorization with Suport Vector Machines: Learning with Many Relevant Features, *Proceedings of the 10th European Conference on Machine Learning ECML '98* pp. 137–142.

Jiang, T., Zhang, Q., Hou, R., Chai, L., McKee, S. A., Jia, Z. and Sun, N. (2014). Understanding the behavior of in-memory computing workloads, *IISWC 2014 - IEEE International Symposium on Workload Characterization* pp. 22–30.

Joachims, T. (1996). TFIDF A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.pdf.

Kibriya, A. M.; Frank, E.; Pfahringer, B; Holmes, G. (2005). Multinomial naive Bayes for text categorization revisited, *Proceedings of 17th Australian Joint Conference on Artificial Intelligence, Cairns, Australia* pp. 488–499.

Kupisz, B. and Unold, O. (2015). Collaborative filtering recommendation algorithm based on Hadoop and Spark, *2015 IEEE International Conference on Industrial Technology (ICIT)* pp. 1510–1514.

Lang, K. (1995). NewsWeeder: Learning to Filter Netnews, *Proceedings of the 12th International Machine Learning Conference (ML95)* pp. 331–339.

Liang, F. and Lu, X. (2015). Accelerating Iterative Big Data Computing Through MPI, *Journal of Computer Science and Technology* **30**(2): 283–294.

Lin, X., Wang, P. and Wu, B. (2013). Log analysis in cloud computing environment with Hadoop and Spark, *2013 5th IEEE International Conference on Broadband Network & Multimedia Technology* pp. 273–276.

McCallum, A. and Nigam, K. (1998). A Comparison of Event Models for Naive Bayes Text Classification, *AAAI/ICML-98 Workshop on Learning for Text Categorization* pp. 41–48.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M. J., Zadeh, R., Zaharia, M. and Talwalkar, A. (2015). MLlib: Machine Learning in Apache Spark, *CoRR* **17**: 1–7.

Mooney, R. J. and Roy, L. (2000). Content-Based Book Recommending Using Learning for Text Categorization, *Pro- ceedings of the Fifth ACMConference on Digital Libraries* pp. 195–204.

Ntoulas, A., Najork, M. and Manasse, M. (2006). Detecting spam web pages through content analysis, *Proceedings of the 15th* .

Ramos, J., Eden, J. and Edu, R. (2003). Using TF-IDF to Determine Word Relevance in Document Queries, *Processing* .

Rennie, J. D. M., Shih, L., Teevan, J. and Karger, D. R. (2003). Tackling the Poor Assumptions of Naive Bayes Text Classifiers, *Proceedings of the Twentieth International Conference on Machine Learning (ICML)-2003)* **20**(1973): 616–623.

Russom, P. (2011). Big Data Analytics.

Shenker, S., Stoica, I., Zaharia, M. and Xin, R. (2013). Shark: SQL and Rich Analytics at Scale, *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD2013)* pp. 13–24.

Zaharia, M., Chowdhury, M., Das, T. and Dave, A. (2012). Fast and Interactive Analytics over Hadoop Data with Spark, *Usenix* **37**(4): 45–51.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S. and Stoica, I. (2010). Spark : Cluster Computing with Working Sets, *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* p. 10.

Zhang, M. L., Peña, J. M. and Robles, V. (2009). Feature selection for multi-label naive Bayes classification, *Information Sciences* **179**(19): 3218–3229.