نموذج رقم (1)

إقـــــرار

أنا الموقع أدناه مقدم الرسالة التي تحمل العنوان:

تصنيف حجم كبير للنص العربي باستخدام MapReduce

Large-Scale Arabic Text Classification Using MapReduce

أقر بأن ما اشتملت عليه هذه الرسالة إنما هو نتاج جهدي الخاص، باستثناء ما تمت الإشارة إليه حيثما ورد، وإن هذه الرسالة ككل أو أي جزء منها لم يقدم من قبل لنيل درجة أو لقب علمي أو بحثي لدى أي مؤسسة تعليمية أو بحثية أخرى.

DECLARATION

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification.

Student's name: Maher M. A. Abushab	**اسم الطالب:** ماهر محمود على أبوشاب
Signature: [illegible]	**التوقيع:** [illegible]
Date: February 22 , 2015	**التاريخ:** 22 شباط ،2015 .

Islamic University – Gaza

Deanery of Post Graduate Studies

Faculty of Information Technology

Information Technology Program

Large-Scale Arabic Text Classification Using MapReduce

Submitted by:
Maher M. Abushab

Dr. Rebhi S. Baraka
Supervisor

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Information Technology

Jumada Al-Awwal 1436H - February 2015

بسم الله الرحمن الرحيم

الجامعة الإسلامية – غزة
The Islamic University - Gaza

مكتب نائب الرئيس للبحث العلمي والدراسات العليا　　　　هاتف داخلي: 1150

الرقم: ج س غ/35/ Ref

التاريخ: 2015/02/22م Date

نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة شئون البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحث/ **ماهر محمود علي أبو شاب** لنيل درجة الماجستير في **كلية *تكنولوجيا المعلومات*** **برنامج تكنولوجيا المعلومات** وموضوعها:

تصنيف حجم كبير للنص العربي باستخدام MapReduce
Large-Scale Arabic Text Classification Using MapReduce

وبعد المناقشة التي تمت اليوم الأحد 03 جمادى الأولى 1436هـ، الموافق 2015/02/22م الساعة الحادية عشرة صباحاً، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

د. ربحي سليمان بركة	**مشرفاً ورئيساً**	[illegible]
أ.د. علاء مصطفى الهليس	**مناقشاً داخلياً**	[illegible]
أ.د. محمد أمين مكي	**مناقشاً داخلياً**	[illegible]

وبعد المداولة أوصت اللجنة بمنح الباحث درجة الماجستير في **كلية *تكنولوجيا المعلومات*/ برنامج تكنولوجيا المعلومات.**

واللجنة إذ تمنحه هذه الدرجة فإنها توصيه بتقوى الله ولزوم طاعته وأن يسخر علمه في خدمة دينه ووطنه.

والله ولي التوفيق ،،،

مساعد نائب الرئيس للبحث العلمي والدراسات العليا

[illegible]

أ.د. فؤاد علي العاجز

شئون البحث العلمي والدراسات العليا
الجامعة الإسلامية غزة
The Islamic University
Research & Graduate Affairs

بسم الله الرحمن الرحيم

وقل رب زدني علما

صدق الله العظيم

Abstract

Text classification on large-scale real documents has become one of the most core problems in text mining. For English and other languages many text classification works have been done with high performance. However, Arabic language still needs more attention and research since it is highly rich and requires special processing.

Existing Arabic text classification approaches use techniques such as feature selection, data representation, feature extraction and sequential algorithms. Few attempts were done to classify large-scale Arabic text document in a parallel manner.

In our research, we propose a parallel classification approach based on the Naïve Bayes algorithm for large volume Arabic text using MapReduce with enhanced speedup and preserved accuracy.

The experiments show that the parallel classification approach can process large volume of Arabic text efficiently on a MapReduce cluster and significantly improves speedup up to 12 times better than the sequential approach using the same classification algorithm. Also, classification results show that the proposed parallel classifier has preserved accuracy up to 97%.

Keywords: *Text Classification, Naïve Bayes algorithm, Parallel Classifier, MapReduce, and Hadoop.*

الملخص

تصنيف النص العربي واسع النطاق باستخدام نموذج MapReduce

أصبح تصنيف النصوص ذات النطاق الواسع (الحجم الكبير) واحدة من المشاكل الأساسية في مجال التنقيب في البيانات النصية. وهناك العديد من أعمال التصنيف النصية للغة الانجليزية واللغات الأخرى حيث نتجت عن أداء عالي لعملية التصنيف. مع ذلك، فان تصنيف النصوص في اللغة العربية بحاجة الى مزيد من الاهتمام والبحث ويتطلب معالجة خاصة نظرا لأنها لغة غنية في التعبير والمعاني والنحو والصرف. أغلب الطرق الحالية لتصنيف النصوص العربية تستخدم تقنيات مثل: اختيار المزايا (Feature Selection)، تمثيل البيانات (Data Representation)، استخلاص المزايا (Feature Extraction) والخوارزميات المتسلسلة (Sequential Algorithms). القليل من المحاولات تمت لتصنيف النص العربي واسع النطاق بالحوسبة المتوازية.

في هذا البحث قمنا باقتراح طريقة مُصنِّف متوازي للنصوص العربية ذات النطاق الواسع يعتمد على خوارزمية التصنيف (Naïve Bayes) باستخدام نموذج الحوسبة المتوازية MapReduce مع تعزيز التسريع (Speedup) والأداء (Performance) والحفاظ على الدقة (Accuracy). أظهرت النتائج أن المُصنِّف المتوازي المقترح يعالج بكفاءة النصوص العربي ذات الحجم الكبير. حيث أجريت التجارب على نموذج MapReduce وأظهرت النتائج تحسنا كبيرا على التسريع بنسبة تصل الى 12 مرة أفضل من الطريقة التسلسلية لنفس المُصنِّف وأيضاً الاحتفاظ بنتائج دقة تصنيف (Accuracy) عالية وصلت الى أعلى من 97%.

الكلمـــات المفتاحيـــة : تصنيف النصوص العربية، المُصنِّف المتوازي، خوارزمية Naïve Bayes، MapReduce وHadoop.

Dedication

To the memory of my mother …

To my beloved father …

To my wife and children …

To my sisters and brothers…

To my best friends …

Acknowledgements

Thanks to Allah for giving me the power and help to accomplish this research. Without the grace of Allah, I was not able to accomplish this work.

Many thanks and sincere gratefulness goes to my supervisor Eng. Dr. Rebhi S. Baraka, without his help, guidance, and continuous follow-up; this research would never have been.

In addition, I would like to extend my thanks to the academic staff of the Faculty of Information Technology who helped me during my master's study and taught me different courses.

Table of contents

List of Figures

List of Algorithms

List of Table

List of Abbreviations

FN	False Negative
FP	False Positive
HDFS	Hadoop Distribute File System
IR	Information Retrieval
K-NN	K-Nearest Neighbors
MPI	Message Passing Interface
MR	MapReduce
NB	Naïve Bayes
PNB	Parallel Naïve Bayes
TC	Text Classification / Text Categorization
TF	Term Frequency
TF-IDF	Term Frequency - Inverse Document Frequency
TM	Text Mining
TN	True Negative
TP	True Positive
VSM	Vector Space Model

Chapter 1 Introduction

Text classification (TC – also known as text categorization) is the task of assigning text documents automatically into one or more categories predefined (or classes, or topics). This task, that falls at the crossroads of information retrieval (*IR*) and machine learning (*ML*), has witnessed increasing interest in the recent years from researchers and developers alike [1, 2]. Automatic text classification has several useful applications such as classifying text documents in electronic format [3, 4], spam filtering, improving search results of search engines [5], web-page content filtering, and opinion mining [6].

Building a text classification system involves three main phases: compilation of the training dataset, selection of the set of features to represent the defined classes, and training the chosen classification algorithm, followed by testing it using the corpus compiled in the first stage as shown in Figure 1.1 [7].

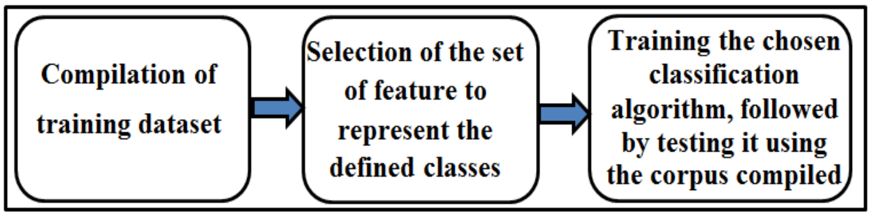

Compilation of training dataset
Selection of the set of feature to represent the defined classes
Training the chosen classification algorithm, followed by testing it using the corpus compiled

Figure 1.1: Building Text Classification System Process

Several methods have been used for text classification [8] such as: Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Artificial Neural Networks, Naïve Bayes (NB) Probabilistic Classifier, Random Forest, and Decision Trees. NB classifier is a statistical method for text classification and is widely applied by many researchers to classify Arabic text documents [9, 10]. It is fast and easy to implement, but it consumes much time when used in classifying large volume of text documents. NB classifier assumes that each feature word is independent from other feature words in a document makes higher efficiency possible but also adversely affects the quality of its results because some of feature words are interrelated [11].

The large amount of text documents with high dimensionality (i.e. the features or attributes are the words that occur in documents) and in Arabic language which has a rich nature and complex morphology requires a large amount of computational power for classification. To be more accurate, we mean by large-scale Arabic text; the large number of text documents that are represented as records (thousands of documents) and the large number of words that are represented as features or attributes in the vector space model after preprocessing the text (thousands of features) [12]. So, in order to preserve accuracy and decrease execution time, we need to resort to parallel programming models such as MapReduce to implement and execute classifications of large volume of Arabic text documents.

MapReduce is a parallel programming model [13] for processing and generating large data sets. It is used to solve many problems, such as data distribution, job scheduling, fault tolerance, machine to machine communication. MapReduce allows developers to write programs that process large-scale of unstructured data in parallel across a distributed cluster of processors or stand-alone computers. It works by breaking the processing into two phases: map phase and reduce phase. Each phase has key-value pairs as input and output, and is specify by two functions: the map function and reduce function [14].

In this research, we build a MapReduce-based parallel classification approach for large scale Arabic text based on Naïve Bayes algorithm that reduces time and preserved accuracy.

To build our approach, we collect a large volume of Arabic corpus and perform several preprocessing steps to prepare the corpus for the classification. Then we design the parallel MapReduce-based classification model. The core of the model is the Naïve Bayes classification algorithm. We designed and conducted several experiments to classify the documents in the collected corpus over the built MapReduce model using the parallelized Naïve Bayes algorithm.

1.1 Problem Statement

Current sequential text classification approaches applied to large-scale Arabic text documents generally require a large number of training inputs to accurately classify large volume of text documents leading to more processing time.

The problem of this research is how to build a MapReduce–based parallel classification approach for large volume Arabic text that reduces the time and preserves the accuracy.

1.2 Objectives

1.2.1 Main Objective

To build a MapReduce–based parallel classification approach for large volume of Arabic text based on Naïve Bayes algorithm that achieves the enhanced level of speedup and preserves the required accuracy.

1.2.2 Specific Objectives

The specific objectives of this research are:

- Determining and collecting an Arabic corpus of text documents with various domains .
- Applying the most suitable text preprocessing techniques such as stemming and term pruning methods and term weighting schemes.
- Designing the suitable MapReduce computing model for parallel classification.
- Implementing the parallel Naïve Bayes algorithm based on the designed MapReduce computing model.
- Conducting the needed experiments on Naïve Bayes algorithm over MapReduce using the collected Arabic corpus.
- Evaluating and comparing the speedup and accuracy of the proposed parallel classifier approach with existing parallel classifier approach using suitable metrics and measures.

1.3 Significance of the Thesis

The rapid increase of online Arabic content in the recent years has raised the need for more efficient Arabic text classification techniques. This work is a contribution in this direction.

- The proposed parallel classifier approach is expected to be applied on multiple domains.
- The approach can be used to efficiently and accurately classify a large volume of Arabic text documents with high dimensionality (i.e. the features or attributes are the words that occur in documents)
- It also overcomes the issue of low speed for the sequential Naïve Bayes algorithm due to the large amount of computational power.

1.4 Scope and Limitations

This research proposes a MapReduce–based parallel Naïve Bayes classifier approach for large volume Arabic text that achieves an enhanced speedup and preserved accuracy. The work is conducted with the following limitations and assumptions:

1. The Arabic corpus will be based on multiple domains.
2. We will apply text preprocessing techniques using RapidMiner and other text document classification tools.
3. Naïve Bayes algorithm will be used for text classification.
4. We will use Apache Hadoop framework to build the cluster where the MapReduce tools will be realized.
5. We will conduct our experiments on a set of processors and their own exclusive memory (multicomputer cluster).
6. We will use 1, 2, 4, 8, 12 and 16 processors to measure the effects on the speedup and the accuracy of proposed approach.

1.5 Research Methodology

In our research, we intend to achieve our specific research objectives using the following methodology shown in Figure 1.2.

- **Research and Survey**: This includes reviewing the recent literature related to MapReduce-based parallel classification and Naïve Bayes classification. Based on the survey, we formulate the general MapReduce and parallel classifier approach.
- **Text Data Collection**: We will collect largest freely public Arabic corpus of text documents with multiple domains having eight classes.
- **Text Preprocessing**: Some preprocessing in the Arabic text corpus will be performed. It includes tokenizing strings to words, normalizing the tokenized words, applying stop words removal, applying the suitable term stemming and pruning methods as a feature reduction technique, and finally applying the suitable term weighting scheme to enhance text document representation as feature vector.
- **Design the Parallel Classifier Approach**: We build the parallel Naïve Bayes classifier for large volume Arabic text based on MapReduce model.
- **Implement the Naïve Bayes Algorithm Using Mahout Library and Hadoop Platform as a realization for the MapReduce model**: We will implement the proposed parallel classifier approach using Naïve Bayes algorithm using Mahout library and Hadoop platform with a multicomputer cluster on the largest freely public Arabic corpus of text documents. We will conduct several experiments to classifying Arabic corpus.
- **Evaluation**: The proposed parallel classifier approach will be evaluated for speedup and accuracy using different performance metrics and classification measures such as precision, recall, and F-measure. In addition, it will be compared with a work implemented in [20] which is a parallel K-NN classifier based on Massage Passing Interface (MPI).
- **Results and Discussion**: In this stage we will analyze the obtained results and justify the effectiveness of the proposed approach.

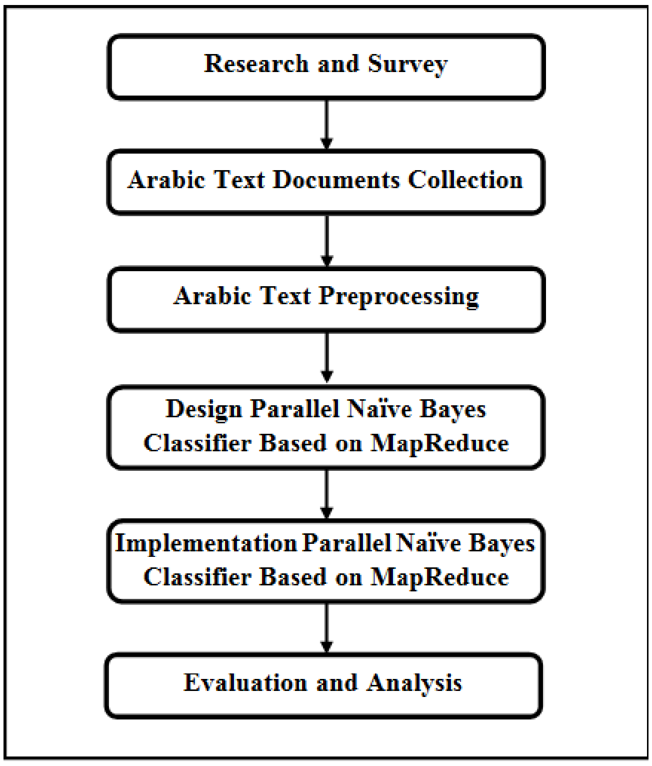

Research and Survey
Arabic Text Documents Collection
Arabic Text Preprocessing
Design Parallel Naïve Bayes Classifier Based on MapReduce
Implementation Parallel Naïve Bayes Classifier Based on MapReduce
Evaluation and Analysis

Figure 1.2: The Research Methodology

1.6 Research Format

The research report is organized as follows: Chapter 2 discusses the state of the art and literature survey. Chapter 3 includes the theoretical foundation of the research. Chapter 4 presents the proposed parallel classification approach. Chapter 5 presents the experimental results and evaluation. Finally, Chapter 6 presents the conclusions and future work.

Chapter 2 Related Works

In this chapter we review related works that address the problem of text classification and identify their limitations, their strengths and the aspects that may be important for our approach.

2.1 Improving the Efficiency of Arabic Text Classification.

Al-Thubaity et al. [15] study the effect of combining five feature selection methods, on Arabic text classification accuracy, two approaches of combination were used, intersection (AND) and union (OR).

They collected a corpus from the website of The Saudi Press Agency (SPA). The SPA consists of 6,300 texts comprising six classes of news, namely culture, economics, general, political, social and sport. The dataset contains more than one million words and the average text length is 172 words. They apply Naïve Bayes (NB) classification algorithm on the SPA dataset to study the effect of feature selection methods combinations on Arabic text classification accuracy. Also, they used feature representation schemas such as namely Boolean and Term Frequency Inverse Document Frequency (TF-IDF) as a weighting scheme for feature selection.

Results show that using Chi-squared (CHI) feature selection method and TF-IDF for feature representation increase the classification accuracy. CHI and Information Gain (IG) feature selection methods produce comparable accuracy and the highest accuracy is achieved when one of them is used, except for on**e** case where relevancy score (RS) achieved the highest accuracy for TF-IDF. In all cases the TF-IDF feature representation performed better than Boolean. Also combining two feature selection methods showed insignificant improvement in classification accuracy, because the complications of using intersection (AND) will cause negative effect on the classification accuracy as the selected features are not enough to train the classifier, and using union (OR) approach cause a problem that is known as the curse of dimensionality. The feature selection methods and weighting schemes can decrease the computation complexity, reduce the dimensionality, and improve the accuracy rate of classification. However, this approach could not do well in the case of reducing computation complexity for text documents with high number of distinct words. Also, this approach reduces the features but does not do well in the case of large volume of text documents with high number of features which increase the computation complexity.

Al-Salemi et at. [16] implement three classifiers based on Bayesian theorem; Simple Naïve Bayes (NB), Multi-variant Bernoulli Naïve Bayes (MBNB) and Multinomial Naïve Bayes (MNB) models on Arabic Text. They applied text reprocessing methods like removing punctuation marks, diacritics and non-Arabic letters, eliminate the words with length less than three and stop word. They applied stemming as feature reduction technique, after that they used several feature selection methods; Mutual Information (MI), Chi-Square statistic (CHI), Odds Ratio (OR) and GSS-coefficient (GSS). They collected 3172 documents belonging to one of four categories (Arts, Economic, Politics and Sport). They split the corpus; 1732 documents for training set and 440 documents for test set. Results show that feature selection and reduction strategies can decrease the computation complexity, reduce the dimensionality of feature space, and improve the performance of classification.

Maybe, the size of the used corpus is small and this approach could not do well in the case of reducing computation complexity for large volume of Arabic text documents with high number of features and in particular in the Arabic language which has a rich nature and very complex morphology.

2.2 Improving the Efficiency of Sequential Classification Algorithms with Parallel Computing.

Ding et al. [17] propose a parallel learning algorithm for text classification. It is based on the combined naïve Bayes text classifier (PC-NB) that relaxes the independence assumption without efficient reduction. They evaluated the parallel implementation on a cluster that consists of six computer, where each node has a 1.6 GHz CPU, 256 MB physical memory and connected by the Ethernet, and MPI library as parallel programming environment. They evaluated the performance on Reuter's dataset with 9603 training documents and 3299 test documents. The experiment results show that the proposed classifier is accurate and powerful while the attributes of an instance are strongly correlated. This approach supports our proposed by using classifier based on Naïve Bayes algorithm, although it is supposed that there will be an impact on the quality of the results because some of feature words are interrelated.

Viegas et al. [18] propose a parallel learning algorithm called GPU-NB. It is based on Naïve Bayes algorithm that uses graphics processing units (GPUs). GPUs are capable of providing a higher parallelism level than what can be obtained with CPUs, with a lower energy consumption. They evaluated GPU-based implementation using Compute Unified Device Architecture (CUDA), the great advantage of this technique is in the simplicity and compactness of the data structures used to represent the document. They evaluated the performance of GPU-NB using six real digital libraries. The collections referred to as Medline, Reuters, ACM, Acl bin, newsgroups20 and Webkb; which have 861,454 documents, 8,184 documents 24,897 documents, 27,677 documents, 18,805 documents, and 8,277 documents respectively. The results show that GPU-NB can speedup the classification process in up 34 x when compared to a sequential CPU-based implementation, also GPU-NB is up to 11 x faster than a CPU-based parallel implementation of Naïve Bayes running with 4 threads.

Moreover, assuming an optimistic behavior of the CPU parallelization, GPU-NB should outperform the CPU-based implementation with up to 32 cores, at a small fraction of the cost. They also show that the efficiency of the GPU-NB parallelization is impacted by features of the document collections, particularly the number of classes, although the density of the collection (average number of occurrences of terms per document) has a significant impact as well.

Kruengkrai et al [19] propose a parallel algorithm for text classification task. The parallel algorithm is based on the Expectation Maximization (EM) algorithm and the NB classifier. One drawback of the NB classifier is that it requires a large set of the labeled training documents for learning accurately. The cost of labeling documents is expensive, while unlabeled documents are commonly available. By applying the EM algorithm, they can use the unlabeled documents to increase the available labeled documents in the training process. They parallelized the algorithm by using the idea of data parallel computation. They evaluated the parallel implementation on a large Linux PC cluster called PIRUN Cluster consists of 72 nodes. They used the 20 Newsgroups data set. It contains approximately 20,000 documents. The experimental results on the efficiency indicate that the parallel algorithm has good speed up characteristics when the problem sizes are scaled up.

Abu Tair and Baraka [20] propose a parallel learning algorithm based on the k-NN algorithm. They evaluated the parallel implementation on a multicomputer cluster that consists of 14 computers, using C++ programming language and the MPI library. They use the proposed parallel classifier to enhance the level of classification speedup, scalability, and accuracy of large-scale Arabic text. They used OSAC Arabic corpus collected from multiple websites, the corpus includes 22,428 text documents. Each text document belongs to 1 of 10 categories (Economics, History, Entertainments, Education and Family, Religious and Fatwas, Sports, Heath, Astronomy, Low, Stories, and Cooking Recipes). The corpus contains about 18,183,511 (18M) words.

They applied suitable term stemming and pruning methods as feature reduction techniques, and finally apply suitable Term Frequency-Inverse Document Frequency (TF-IDF) weighting scheme to enhance text document representation as feature vectors. The experimental results on the performance indicate that the parallel classifier design has very good speedup characteristics when the problem sizes are scaled up. Also, classification results show that the proposed classifier has achieved accuracy, precision, recall, and F-measure with higher than 95%. This work supports our approach in terms of using cluster, but the volume of text documents used in corpus is small-scale compare to large volume of text documents with high number of features. We will compare our approach to this approach in term of classifying large-scale Arabic text classification.

Chu et al. [21] Propose a parallel learning algorithm. The parallel algorithm based on Naïve Bayes using MapReduce model on Shared-memory system. They specify different sets of mappers to calculate them, and then the reducer sums up intermediate result to get the final result for the parameters. Their experiment was on a 16 way Sun Enterprise 6000 running Solaris 10. They evaluated the average speedup on ten datasets from the UCI Machine Learning repository with different size (from 30000 to 2500000), which makes their report more convincing. The results showed that the speedup was [4 nodes, 4x], [8 nodes, 7.8x], [16 nodes, 13x].

Esmaeili et al. [22] use distance detection in vector space model for classifying the News articles, to calculated distances between weighted frequency vectors of each category, and the News vector determine its category by finding minimum distance with weighted frequency vector of categories. They used MapReduce, as a distributed programming model, to implement and execute distributed classification of the news articles, in order to increase performance, calculation accuracy and decrease execution time. They use Hamshahri News dataset that contain the News about 12 years (1996-2007). The dataset include 314106 News files that its volume is about 1.2 GB. The News is categorized in 9 main categories and 26 different sub-categories. They use 80 percent of the dataset for train phase and the remaining 20 percent for test phase.

They implemented proposed distributed classifier, using four machines with 16 cores AMID Opteron 800 MHz processor, 32 GB of RAM and 500 GB of storage volumes. There is LAN network with 100 Mbps that connect the four machines together. Also for building their cluster, they used 1.0.4 version of the Hadoop and Linux CentOS6.2 and Redis 2.6 for storing result. The result show that For train and test dataset with 80-20 ratio, the average values of precision and recall are 29.66% and 53.88% in 35 (number of main categories plus sub-categories count) categories and these metrics are 52.67% and 63.75% for 9 main categories, and also, the processing time with 22 Map function and 100 Reduce function is about 1mins, 51 sec while using those methods with non-distributed manner need some weeks and months according to the volume of data. This work supports our approach in terms of using cluster and MapReduce, a distributed programming model which is a viable and attractive programming model for processing large data sets with a parallel and distributed algorithm on a cluster.

Zhou et al. [23] propose model of parallel classifying algorithm, The parallel algorithm is based on Naïve Bayes algorithm with Map Reduce. They build a small cluster with 3 business machines (1 master and 2 slaves) on Linux, and each machine has two cores with 3.10 GHz, 4GB memory, and 500GB disk. They use the Hadoop version 0.20.2 and java version 1.6.0_26. They test efficiency and scalability of parallel Naïve Bayes algorithm proposed on seven datasets from the UCI Machine Learning repository with different size (from 178 KB to 1 MB).

The Naïve Bayes classifier implemented by the MapReduce trains the training data sets to generate the classification model, and then use the model to classify the removed category samples. The proposed model improved algorithm performance when using with large data set; moreover the parallel algorithms can not only process large datasets, but also enhance the efficiency of the algorithm.

This work supports our approach in terms of using cluster and MapReduce, a distributed programming model which is a viable and attractive programming model for processing large data sets with a parallel and distributed algorithm on a cluster.

2.3 Summary

In this chapter, we presented a review of existing works closely related to our research and identified the drawbacks of existing approaches; we classified the methods of improving the efficiency of sequential classification algorithms into two categories: The first category includes approaches using the a combination of feature selection strategies that decrease the computation complexity, reduce the dimensionality, and improve the accuracy rate of classification. The second category includes approaches using the parallel computing of improving the efficiency of the sequential NB algorithm; their platform comprises a multiprocessor with shared memory that connects multiple processors to a single memory system.

In the next chapter, we present the theoretical foundation underlying our research.

Chapter 3 Theoretical Foundation

In this chapter, the fundamental concepts which represent the basis for understanding our research are presented. First, text classifiers are introduced, followed by providing an overview to Naive Bayes classifier which is used in our proposed parallel classifier, and K-Nearest neighbor classifier which is used in the comparison with our proposed approach. Then Large-scale Arabic text, stemming method, and text representation are explained. Also MapReduce pattern, Apache Hadoop, Hadoop Distributed File System (HDFS), Massage Passing Interface (MPI), and Apache Mahout library are presented in detail. Finally we present an overview of used performance metrics and classification measures.

3.1 Text Classifiers

Many of machine-learning algorithms have been successfully used in text classification such as Support Vector Machine (SVM), K-Nearest Neighbor (K-NN), Artificial Neural Networks, Naïve Bayes (NB) probabilistic Classifier, Random Forest, Distance Detection and Decision Trees. The goal of classification is to build a set of models that can correctly classify the class of the different objects. The input to these methods is a set of objects (i.e., training data), the classes which these objects belong to (i.e., dependent variables), and a set of variables describing different characteristics of the objects (i.e., independent variables). Once such a predictive model is built, it can be used to predict the class of the objects for which class information is not known a priori. The key advantage of supervised learning methods over unsupervised methods is having an explicit knowledge of the classes [1, 24]. Naïve Bayes (NB) algorithm is used as classifier in our work, and K-Nearest Neighbors (K-NN) is used as comparison classifier in our proposed approach. In the next sections, we provide a brief overview of NB and K-NN classifiers.

3.1.1 Naïve Bayes (NB) Classifier

Naïve Bayes classifier is a simple probabilistic classifier which works by applying the Bayes' theorem along with naïve assumptions about feature independence. It assumes that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class conditional independence [1, 24]. It classifies data in two steps [25]:

a. Training Step: Using the training samples, the method estimates the parameters of a probability distribution, assuming features are conditionally independent in the given class.

b. Prediction Step: For an unlabeled test sample, the method computes the posterior probability of that sample belonging to each class. The method then classifies the test sample according the largest posterior probability.

- **Derivation of Naïve Bayes Classifier**

Depending on the precise nature of the probability model, Naïve Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for Naïve Bayes models uses the method of maximum likelihood [23, 24, 26]. Naïve Bayes classification algorithm is described as follows [27]:

- Let *D* be training set of tuples and their associated class labels. Each tuple is represented by a n-dimensional attribute vector, $X= (x_1, x_2, \ldots x_n)$, *n* measurements made on the tuple from *n* attribute, respectively, $A_1, A_2, \ldots, A_n$.
- Suppose that there are *m* classes, $c_1, c_2, \ldots, c_m$. Given a tuple, *X*, the classifier will predict that *X* belongs to the class having the highest probability, conditioned on *X*. That is, the NB classifier predicts that tuple X belongs to the class C_i. If and only if

$$P(C_i|X) > P(C_j|X) \; for \; 1 \le j \le m, \; j \ne i. \quad (3.1)$$

Thus we maximize $P(C_i|X)$. The class C_i for which $P(C_i|X)$is the maximized is called the maximum posteriori hypothesis. By Bayes' theorem (Equation 3.2).

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{p(X)}. \quad (3.2)$$

- As $P(X)$ is constant for all classes, only $P(X|C_i)P(C_i)$ need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equal.

- Based on the assumption is that attributes are conditionally independent (no dependence relation between attributes), $\mathrm{P(X|C_i)}$ is computed using Equation 3.3.

$$P(X|C_i) = \prod_{k=1}^{n} P(X_k|C_i) \qquad (3.3)$$

The probabilities $P(X_1|C_i), P(X_2|C_i), \ldots, P(X_n|C_i)$ can be estimated from the training sample, where:

a. If $\mathrm{A_k}$ is categorical, then $P(X_k|C_i)$ is the number of tuples C_i in D having value X_k for $\mathrm{A_k}$ divided by$|C_i, D|$, (number of tuples of C_i in *D*).

b. if $\mathrm{A_k}$ is continuous-valued, $P(X_k|C_i)$ is usually computed based on a Gaussian distribution with a mean μ and standard deviation σ and, $P(X|C_i)$ is:

$$P(X|C_i) = g(X_k, \mu_{C_i,} \sigma_{C_i}) \qquad (3.4)$$

$$g\left(X_k, \mu_{C_i,} \sigma_{C_i}\right) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \qquad (3.5)$$

Where μ is the mean and σ^2 is the variance. If an attribute value doesn't occur with every class value, the probability will be zero, and a posteriori probability will also be zero.

- In order to classify an unknown sample X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i. Sample X is then assigned to class C_i if and only if

$$P(C_i|X) > P(C_j|X) \text{ for } 1 \leq j \leq m, j \neq i. \quad (3.6)$$

Where

$$P(X|C_i) = \prod_{k=1}^{n} P(X_k|C_i) \qquad (3.7)$$

NB classifier has high computational efficiency as compared to other wrapper methods because it is inexpensive since it is considered linear time O(n) complexity classifier. Informally, this means that for large enough input sizes the running time increases linearly with the size of the input [28]. NB classifier is simple, accurate, fast, low variance due to less searching, handles streaming data well, and easy to implement.

It is based on a simplistic assumption in real world and is only valid to multiply probabilities when the events are independent. Despite this, NB often works much better in many complex real-word situations than one might expect [29]. It exhibits high accuracy and speed when applied to huge amounts of data [23]. Thus chosen to be the proposed classifier in our approach.

3.1.2 K-Nearest Neighbors (K-NN) Classifier

The K-NN algorithm [24] is one of the supervised learning algorithm, the purpose of this algorithm is to classify a new object based on attributes and training samples. It is based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The training tuples are described by n attributes. Each tuple represents a point in an n-dimensional space. In this way, all training tuples are stored in an n-dimensional pattern space. When given an unknown tuple, a K-NN classifier searches the pattern space for the *k* training tuples that are closest to the unknown tuple. These *k* training tuples are the k nearest neighbors of the unknown tuple. Closeness is defined in terms of a distance metric, such as Euclidean distance.

The Euclidean distance between two points or tuples, $X_1=(x_{11},x_{12},\ldots,x_{1n})$ and $X_2=(x_{21},x_{22},\ldots,x_{2n})$, is

$$dis(X_1\,,X_2) = \sqrt{\textstyle\sum_{i=1}^{n}(x_{1i}-x_{2i})^2} \qquad (3.9)$$

K-NN algorithm as described in [24, 27] can be summarized as :

- Determine the parameter *k* i.e., the number of nearest neighbors beforehand.
- Calculate the distance between the query-instance and all the training samples using Euclidean distance as in equation (3.9).
- Distances for all the training samples are sorted and nearest neighbor based on the *k*-th minimum distance is determined.
- Since the K-NN is supervised learning, get all the categories of your training data for the sorted value which fall under *k*.
- The predicted value is measured using the majority of nearest neighbors.

K-NN works well even when there are some missing data. K-NN is good at specifying which predictions have low confidence and has some strong consistent results. K-NN algorithm has several disadvantages such as: the complexity of computation time needed to traverse all the training documents [30], and the difficulty to determine the value of *k* [31], where a series of experiments with different *k* values should be conducted to determine the best value of k.

K-NN algorithm based classifier approach [20] (see Section 2.2) is used in our research as basis for comparison with our proposed approach.

3.2 Large Scale Arabic Text Classification

Most of text classification algorithms have problems with computational complexity of training phase with large scale text documents. The huge amount of text documents with high dimensionality (i.e. the features or attributes and in our case they are the words that occur in documents) and in particular in Arabic language which is a rich and has complex morphology requires a large amount of computations for classification. Large-scale Arabic text means the large number of text documents that are represented as records (thousands of documents) and the large number of words that are represented as features or attributes in the vector space model after preprocessing the text (thousands of features). In order to overcome the complexities of large scale Arabic text classification, researchers developed some techniques such as: feature selection, feature extraction and used distributed computing as platform for computations.

- Feature selection is a basic approach for reducing feature vector size. Different feature selection methods are used in Arabic text classification such as: Term Frequency, Document Frequency, and Information Gain [32, 33, 34].
- Feature extraction is a basic approach for high dimensional feature space to be transformed into low dimensional feature space. For Arabic text classification, words are treated as a feature using their orthographic form, stems which the suffix and prefix were removed from the orthographic form of the word, and the word root, which is the primary lexical unit of a word [35, 36].
- Distributed computing is a basic computing model using different parallelization techniques such as: Massage Passing Interface (MPI), and MapReduce. In [20],

they proposed a K-NN parallel learning algorithm to classify large-scale Arabic text using MPI Model. In our research we will present new approach to classify large scale Arabic text documents using NB algorithm with MapReduce model. MapReduce model is introduce in Section 3.5.

3.3 Stemming Methods

A stem is a natural group of words with equal (or very similar) meaning. Stemming algorithm is a computational process that gathers all words that share the same stem and has some semantic relation [37]. The main objective of the stemming process is to remove all possible affixes and thus reduce the word to its stem. After the stemming process, every word is represented by its stem [38]. Stemming is needed in many applications such as natural language processing, compression of data, and information retrieval systems. Many stemmers have been developed for English and other European languages. These stemmers mostly deal with the removal of suffixes as this is sufficient for most information retrieval purposes. Most Arabic language stemming approaches fall into three classes [39, 37]:

- Root-Based stemmers use morphological analysis to extract the root of a given Arabic word.
- Statistical stemmers attempt to group words variances using clustering techniques.
- Light Stemming reduces Arabic words to their light stems by removing frequently used prefixes and suffixes in Arabic words. Light stemming is chosen because it allows remarkably good information retrieval without providing correct morphological analyses [40].

3.4 Text Representation

Text documents should be represented in some way that enables the classifier to interpret them an indexing method is needed to transform text documents represented by strings of characters to another interpretable representation of the contents of the documents. The most popular approach for data representation is the vector space model.

3.4.1 Vector Space Model

The Vector Space Model (VSM) represents documents as vectors in the space, each vector can be represented by the weights of terms in a document with respect to the dimension of the space. The number of dimensions equals the number of terms or keywords used, we can represent this as a two way matrix where the columns represent terms and rows represent documents in the set, the entries of the matrix are the weights of term i in document j. In the basic two dimensions Cartesian plane, a vector is represented by two points, each consists of the ordered pair *x* and *y*. To represent a vector of *N* terms we need *N* dimensions [41].

Given a collection of documents, its feature vectors are represented by a word-by-document matrix, where each entry represents the weight of a word in a document. The aim of term weighting schemes is to enhance text document representation as feature vector. There are several popular term weighting schemes such as:

- **Binary Term Occurrences (BTO):** which indicates absence or presence of a word with booleans 0 or 1 respectively.
- **Term Occurrences (TO):** the number of occurrences of term t_i in the document d_j.
- **Term Frequency-Inverse Document Frequency (TF-IDF):** is a numerical statistic that is used to evaluate how important a word is to a document in a collection or corpus. It is often used as a weight factor in information retrieval and text mining. The *TF-IDF* value increases proportionally to the number of times a word appears in a document. *TF-IDF* undervalues terms that frequently appears in documents belonging to the same class and gives greater weight to terms that represent the characteristic of the documents in its class [36, 41].
- The calculation of *TF-IDF* is defined as follows [42]:
  a. The term count in the given document is simply the number of times a given term appears in that document. For the term t_i with the particular document d_j, its term frequency (*TF*) is define as follows:

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{3.8}$$

Where $n_{i,j}$ is the number that the term t_i occurred in document d_j, and the denominator is the sum of occurrences of all terms in document d_j.

b. The inverse document frequency (*IDF*) can be calculated from document frequency as follows:

$$IDF_i = \log \frac{|D|}{|\{j: t_i \in d_i\}|} \qquad (3.9)$$

Where |D| is the total number of documents in the corpus, and $\{j: t_i \in d_j\}$ is the number of documents, where the term t_i appears. The *IDF* of a term is low if it occurs in many documents and high if the term occurs in only few documents.

c. The TF-IDF weight is the product of *TF* and IDF. The formula is defined as follows:

$$(TF - IDF)_{i,j} = TF_{i,j} \times IDF_i \qquad (3.10)$$

3.5 MapReduce Overview

MapReduce (MR) is a parallel programming model introduced by Google in 2004, and is used in processing and generating large data sets implementation [43]. It is useful for tasks such as data mining, log file analysis, financial analysis and scientific simulations, filtering documents by tags, counting words in documents, and extracting links to related data [44, 45]. The advantages of MapReduce is simple and easy to use, flexible does not have any dependency on data model and schema, basically independent from underlying storage layer, high scalability, and highly fault-tolerant because each node in the cluster is expected to report back periodically with completed work and status updates [46].

3.5.1 MapReduce Architecture

The basic idea of MapReduce comes from divide and conquer algorithms which are used to partition a large problem into smaller subproblems [47]. Key-value pairs form the basic data structure in MapReduce. MR algorithm involves imposing key-value structure on arbitrary datasets. The programmer defines a mapper and a reducer with the following signature:

$$map : (k_1, v_1) \rightarrow list[(k_2, v_2)]$$
$$reduce : (k_2, list[v_2]) \rightarrow [(k_3, v_3)]$$

The mapper is applied to every input key-value pair (split across an arbitrary number of files) to generate an arbitrary number of intermediate key-value pairs. The reducer is applied to all values associated with the same intermediate key to generate output key-value pairs. Implicit between the map and reduce phase is a distribute "group by" operation on intermediate (shuffle phase). Intermediate data arrive at each reducer in order, sorted by the key. Output key-value pair from each reducer is written in r files on the distributed file system, where r is the number of reducer [48].

Figure 3.1 shows the overall flow of a MapReduce operation. When the user program calls the MapReduce function, the following sequence of actions occur:

1. **1.** MapReduce in user program will divide the input files into N pieces with size varies from 16 MB to 64 MB.
2. **2.** Then it will start many programs on a cluster of different machines. One is master program and the rest are workers, master can assign M map tasks and reduce tasks to an idle workers.
3. **3.** If a worker is assigned a map task, it will parse the input data partition and output key/value pairs, then pass the pair to a user defined Map function. The intermediate key/value pairs are buffered in memory.

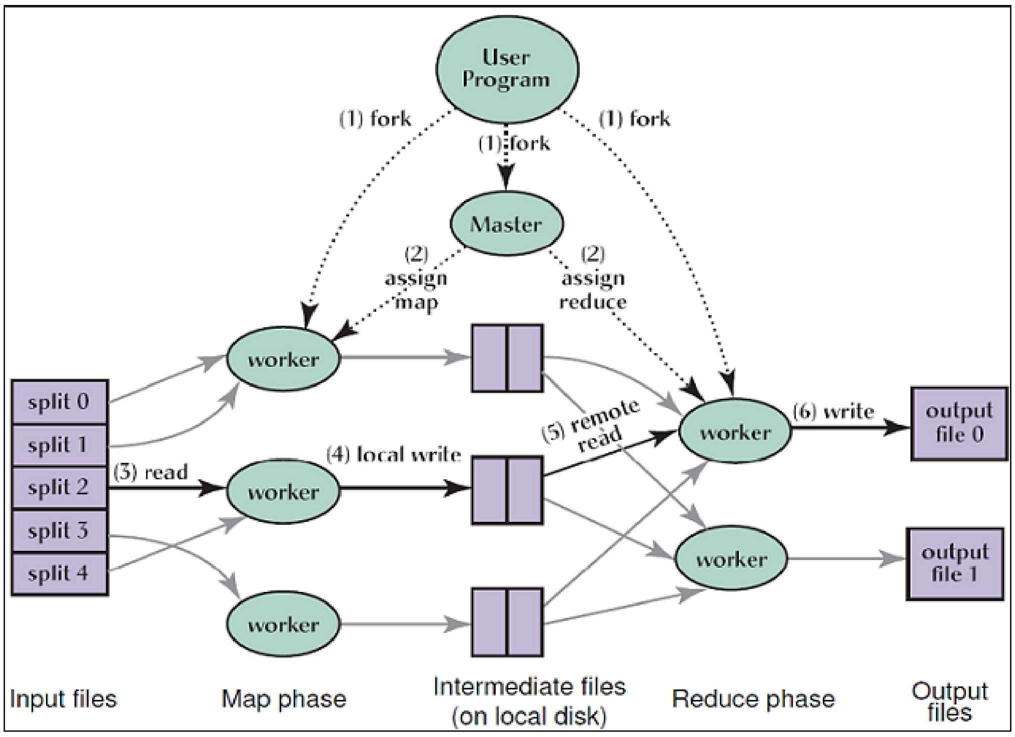


Figure 3.1: MapReduce Operation [13]

4. Periodically, the buffered pairs are written to local disk. After that, the local machine will inform the master of the location of these pairs.
5. If a worker is assigned a reduce task, it will read entire buffer by using remote procedure calls. After that, it will sort the temporary data based on the key.
6. Then the reduce worker will deal with all of intermediate data. For each key and according to set of values, the reducer passes key/value pairs to a user define reduce function. The output is the final output of this partition.
7. After all of the mappers and reducers have finished their work, the master will return the result to user programs. The output is stored in F individual files.

A simple example in [49] that is often used to explain how MapReduce works in practice. It consists in counting the occurrence of single words with in a text. An overview of how MapReduce works is shown in Figure 3.2.

1. Input data (on a distributed file system).
2. Input data are partitioned into smaller chunks of data.

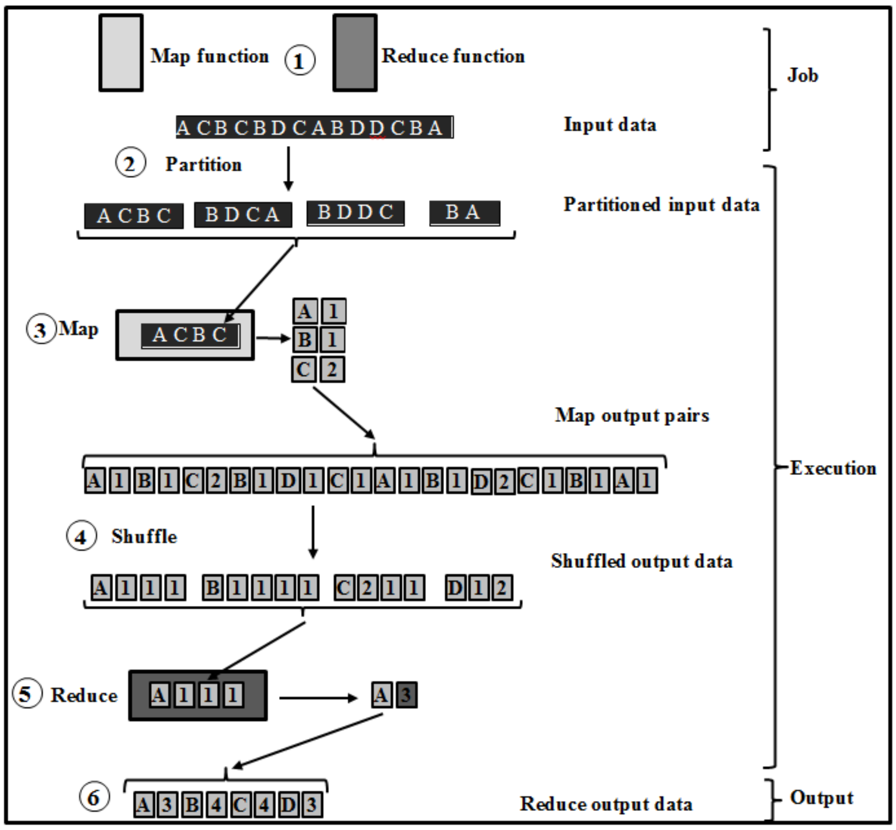


Figure 3.2: MapReduce Programming Model Example [50]

3. For each chunk of input data, a "map task" runs which applies the map function resulting output of each map task is a collection of key-value pairs.
4. The output of all map tasks is shuffled for each distinct key in the map output; a collection is created containing all corresponding values from the map output.
5. For each key-collection resulting from the shuffle phase, a “reduce task” runs which applies the reduce function to the collection of values. The resulting output is a single key-value pair.
6. The collection of all key-value pairs resulting from the reduce step is the output of the MapReduce job.

3.6 Hadoop

Hadoop [50] is an open source framework for writing and running distributed processing of large-scale data sets on high performance cluster. Distributed computing is wide and varied field, but the key distinctions of Hadoop are:

- Accessible*:* Hadoop runs on large clusters of commodity machines or on cloud computing services.
- Robust: Because it is intended to run on commodity hardware, Hadoop is architected with the assumption of frequent hardware malfunctions. It can gracefully handle most such failures.
- Scalable Hadoop scales linearly to handle larger data by adding more nodes to the cluster.
- Simple: Hadoop allows users to quickly write efficient parallel code.

Hadoop accessibility and simplicity give it an edge over writing and running large distributed programs. On the other hand robustness and scalability make it suitable for even the most demanding jobs. Figure 3.3 shows a Hadoop cluster with its distributed computing nodes connected through on Ethernet switch.

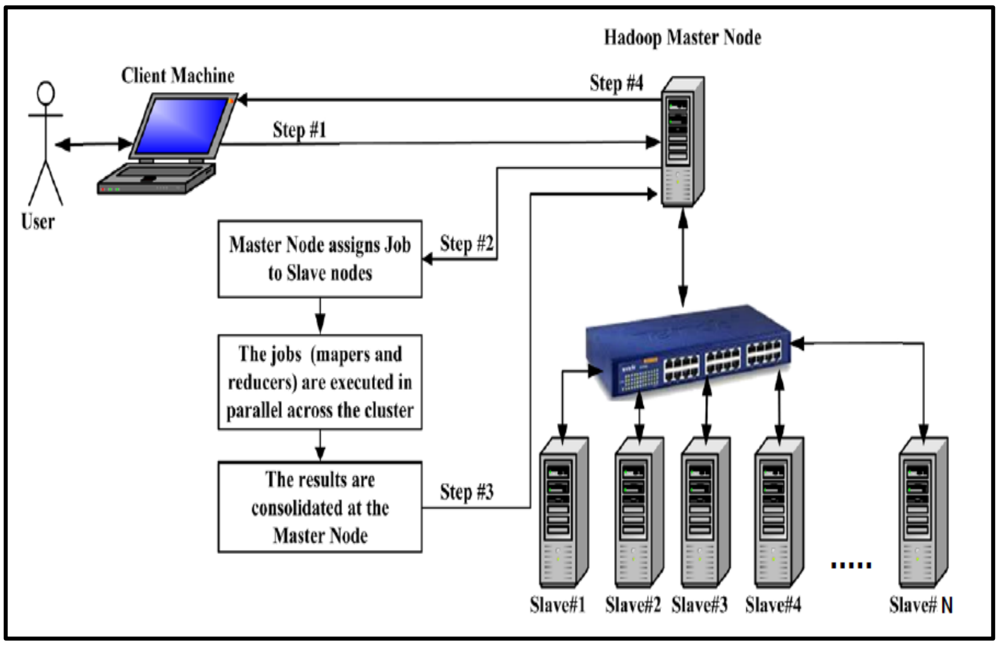

Hadoop Master Node
Client Machine
Step #4
Step #1
User
Master Node assigns Job to Slave nodes
Step #2
The jobs (mapers and reducers) are executed in parallel across the cluster
The results are consolidated at the Master Node
Step #3
Slave#1
Slave#2
Slave#3
Slave#4
Slave# N

Figure 3.3: The Architecture of the Hadoop Cluster [51]

The cluster runs jobs controlled by the master node, which is known as the NameNode and it is responsible for chunking the data, cloning it, sending the data to the distributed computing nodes (DataNode), monitoring the cluster status, and collecting/aggregating the results. Hadoop focuses on moving code to data instead of vice versa. Hadoop is composed into two main subsystems: Hadoop Distributed File System (HDFS) is used for storing the data and MapReduce (MR) used to manipulate the data which is stored on the file system [14].

In the next section we describe HDFS components while MapReduce is explained in Section 3.5. It offers reliable and scalable distributed computing [51]. It is applied in several areas such as text mining, website rating, opinion mining, users' recommendation in some social media, weather forecasting, data analysis, and many problems that require large scale processing [52].

3.6.1 Small Number of Large Files vs Large Number of Small Files in Hadoop

Hadoop [14] is designed to process very large files; "very large" in this context means files that are hundreds of megabytes, gigabytes, or terabytes in size. It works better with a small number of large files than a large number of small files. One reason for this is that HDFS generates splits in such a way that each split is all or part of a single file. If the file is very small ("small" means significantly smaller than an

HDFS block) and there are a lot of them, then each map task will process very little input, and there will be a lot of them (one per file), each of which imposes extra bookkeeping overhead. Compare a 1 GB file broken into sixteen 64 MB blocks, and 10,000 or so 100 KB files. The 10,000 files use one map each, and the job time can be tens or hundreds of times slower than the equivalent one with a single input file and 16 map tasks.

Hadoop becomes a bottleneck when handling massive small files because the name node use more memory to store the metadata of files and the data nodes consume more CPU times to process massive small files. In [54] it is shown that merge massive small files into single large file improve the performance of processing of small files.

3.6.2 Hadoop Distributed File System

Hadoop distributed file system (HDFS) [14, 54, 55] is a distributed file system designed for storing and supporting very large files with streaming data access pattern (write-once and read-many times) running on a cluster of low-cost hardware. HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. It uses replication of data stored on DataNode to provide reliability. Files in HDFS are divided into block size chunks (default size is 64MB), which lead to minimizing the time necessary for seeks, further blocks allow for an easy mechanism to provide fault tolerance and availability [56]. It provides fast, scalable access to the information which is stored in Hadoop [25]. The system architecture of HDFS as shown in Figure 3.4 has two types of nodes; a NameNode as master and a number of DataNode as workers [55].

- **NameNode**

  The master NameNode manages the file system namespace. It keeps the file system tree and metadata for files and directories in the tree, and determines the mapping of data blocks containing the file in data nodes. While storing/writing data to HDFS, NameNode chooses a group of nodes (by default three) to store the block replicas.

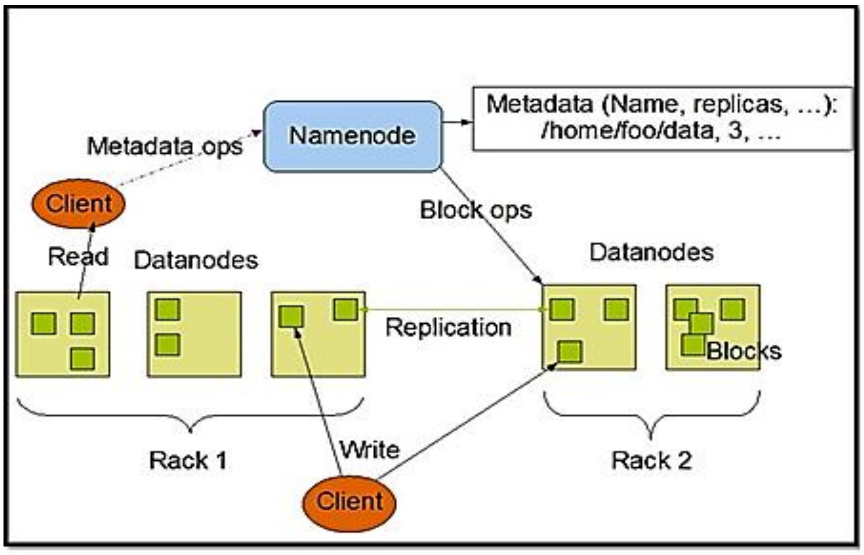


Figure 3.4: Hadoop Distributed File System Architecture [56]

- **DataNode**

  The workers DataNodes are responsible for storing the blocks of files as determined by the NameNode. Also it is responsible for creating, deleting and replicating blocks of files after being instructed by the NameNode.

3.7 Massage Passing Interface (MPI)

MPI [57] is a widely-used message passing standard. Its basic functions are defined by the MPI standard and with implementations targeting distributed memory architectures. One of the key objectives of the MPI standard is to provide portability between different parallel machines. MPI defines its own data types which are used for data transfers and mapped to specific machine-specific data types by the MPI library implementation.

Unlike MapReduce, data in MPI architectures is shared arbitrarily between nodes for synchronization and this is not reliable because the overhead due to the network traffic could dramatically affect performance [58]. Other differences between the two paradigms are stated in Table 3.1.

Table 3.1: MPI and MapReduce Comparison [57]

Items	MPI	MapReduce
What they are	General parallel programming paradigm	A programming paradigm and its associated execution system
Programming model	Massage passing between nodes	Restricted to MapReduce operation
Data organization	No assumptions	Files can be shared
Execution model	Node are independent	Map/Shuffle/Reduce
Usability	Difficult to debug	Simple concept, could be hard to optimize
Key selling point	Flexible to accommodate various applications	Flow through large amount of data with commodity hardware

3.8 Apache Mahout Library

Mahout [59] is a scalable machine learning library running on Apache Hadoop. It provides various machine learning techniques such as recommender engines (collaborative filtering), clustering, and classification. The core of clustering, classification, collaborative filtering algorithms realization is based on Map Reduce paradigm. Its machine learning algorithms are written in java and some portion are built upon Apache Hadoop distributed computation. It is designed to be highly scalable and with the increase of the number of records required to train a model, the time and memory required for training a Mahout algorithm may not increasing linearly, making scalable algorithms in Mahout widely useful [60].

It aims to be the machine learning tool of choice when the collection of data to be processed is very large. All implemented algorithms run in a single machine and some of them are implemented in distributed mode using MapReduce paradigm. It includes a number of classification algorithms such as: Naïve Bayes, Neural Networks, Support Vector Machines, Logistic regression, K-Means, and Canopy Clustering. We choose Mahout’s Naïve Bayes algorithm as the classifier in our research because it is a general framework for MapReduce machine learning algorithms and it can be deployed on top of Apache Hadoop leveraging the full scalability it provides.

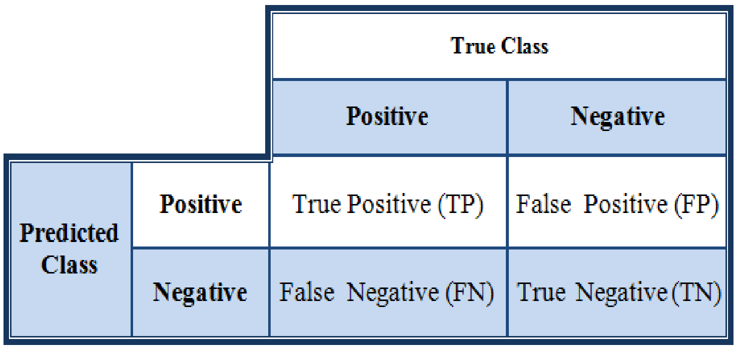
3.9 Performance Metrics and Classification Measures

The performance metrics is a measure of a systems performance. There are several performance metrics such as: speedup, efficiency and scalability, and many classification measures like: accuracy, precision, recall, and F-measure using to evaluate the parallel classifier [61]. They will be used in later to evaluate the effectiveness of our proposed parallel classifier.

3.9.1 Confusion Matrix

The confusion matrix [24] is one of popular tools to evaluate the performance of a model in tasks of classification or prediction. The confusion matrix is represented by a matrix with each row representing the instances in a predicted class, while each column representing in an actual class as shown in Table 3.2.

Table 3.2: Simple Confusion Matrix



		True Class	
		Positive	**Negative**
Predicted Class	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

- **True Positive (TP):** refers to the number of positive instances that are correctly labeled by the classifier.
- **True Negative (TN):** refers to number of negative instances that are correctly labeled by the classifier.
- **False Positive (FP):** refers to the number of positive instances that are incorrectly labeled by the classifier.
- **False Negative (FN):** refers to number of negative instances that are incorrectly labeled by the classifier.

3.9.2 Accuracy

refer the percentage of test set instances that are correctly classified by the classifier.

$$Overall\ Accuracy\ = \frac{(TP + TN)}{(TP + TN + FP + FN)} \qquad (3.1)$$

3.9.3 Precision

refer to the percentage of predicted documents for the given topic that are correctly classified.

$$Precsion\ = \frac{TP}{(TP + FP)} \qquad (3.2)$$

3.9.4 Recall

refers to the percentage of the total documents for the given topic that are correctly classified.

$$Recall\ = \frac{TP}{(TP + FN)} \qquad (3.3)$$

3.9.5 F-measure

it is a standard statistical measure that is used to measure the performance of a classifier system. The F-measure is an average parameter based on precision and recall.

$$\text{F–measure}\ = \frac{2\ \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \qquad (3.4)$$

3.9.6 Speedup

A standard metric to measure the efficiency of a parallel algorithm is the speed up factor [62]. It is defined as the ratio of the time required to solve a specific problem on a single processor to the time required to solve the same problem on a parallel computers with *N* identical processing elements [63].

It is defined as: $S_n = t_s / t_p$, where t_s is the execution time using only one processor and t_p is execution time using n processor. The maximum speed that can be reached is linear speedup.

3.10 Summary

In this chapter, we presented an overview of the basic theoretical foundation related to our research. We present text classifiers, Naïve Bayes classifier, K-Nearest Neighbor classifier, text representation, large scale Arabic text classification, MapReduce, Hadoop platform, Hadoop distributed File System, Massage Passing Interface, and Apache Mahout library. Finally we described performance metrics and classification measures that are used to evaluate the effectiveness of a parallel classifier.

In the next chapter, we provide a detailed description of the proposed parallel classifier approach.

Chapter 4 The Proposed Parallel Classifier Approach

In this chapter we present the proposed parallel classifier approach. We describe all steps of the proposed parallel classifier using algorithms and diagrams. We use MapReduce model to solve the problem of processing a large scale Arabic text. First, we present the steps of collecting Arabic text documents and applying text preprocessing. Second, we describe the steps of splitting and distributing the documents of the collected corpus as MapReduce tasks. Third, we present the steps of calculating the term frequency (TF) and term frequency-inverse document frequency (TF-IDF) using MapReduce model. Finally, we present the Naïve Bayes text classification using MapReduce model.

4.1 The Overall Classification Approach

Figure 4.1 shows the workflow of the parallel classification process. It is roughly divided into four kinds of activities:

1. **Corpus collection and cleaning activities.** The corpus is collected and divided into text documents, then text preprocessing is applied to remove non-Arabic text, perform tokenization, remove Arabic stop word and perform light stemming.
2. **HDFS document uploading, splitting and configuration activities.** An important step in developing a parallel algorithm is to split the problem into tasks that can be executed in parallel by identifying the data on which computations to be performed, and then partitioning this data across various tasks. A task performs the computations with its part of data. In our classifier, the input training data set (corpus) are transferred into a sequence of files then uploaded to HDFS in the MapReduce setting. HDFS splits corpus into 16 MB to 64 MB chunks each presented as a map task and then distribute them among workers with replication 3 times by default. Also, it assigns the parameter configuration such as: the documents number, the classes number and the documents number in each class of corpus.

3. **Term-specific MapReduce-based calculations activities.** Each MapReduce worker node receives its assigned data and calculates parameters such as: word frequency and word counts, then calculates the term frequency- inverse document frequency (TF-IDF) value to generate the vector space model.
4. **Naïve Bayes MapReduce computation activities.** The result of the last step is divided into training set and testing set. The master node assigns workers to calculate probabilities of each class of training set using Naïve Bayes MapReduce classification. Finally, the master node assigns another MapReduce workers nodes to calculate conditional probabilities of each feature value in the testing to predicting the class for each new document.

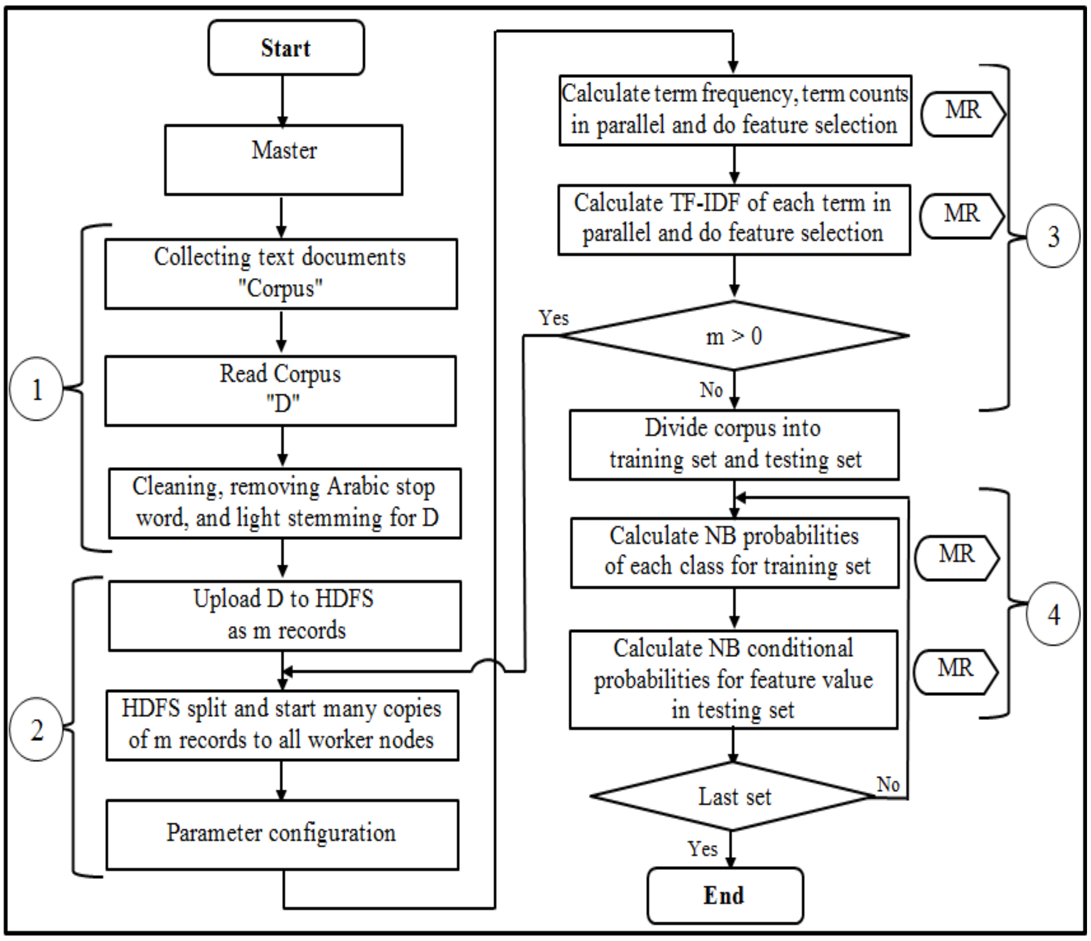

Start
Master
Collecting text documents "Corpus"
Read Corpus "D"
Cleaning, removing Arabic stop word, and light stemming for D
Upload D to HDFS as m records
HDFS split and start many copies of m records to all worker nodes
Parameter configuration
1
2
Calculate term frequency, term counts in parallel and do feature selection
Calculate TF-IDF of each term in parallel and do feature selection
MR
MR
3
m > 0
Yes
No
Divide corpus into training set and testing set
Calculate NB probabilities of each class for training set
Calculate NB conditional probabilities for feature value in testing set
MR
MR
4
Last set
No
Yes
End

Figure 4.1: Workflow of the Proposed Approach

Next we present the details of these classification activities comprising the proposed approach.

4.2 Corpus Collection

One of the difficulties that encountered this work in the field of Arabic language processing is unavailable suitable large volume Arabic corpus for evaluating text categorization algorithms.

Different training data sets are available for text classification in English while few free Arabic training data sets are available to researcher. The most existing popular Arabic text corpus used in text mining field cannot meet our experiments data size for real large-scale Arabic text corpus. Therefore, we choose to collect real large-scale Arabic text corpus from Shamela library [64] which contains a huge collection of data in different Arabic fields.

To build a text dataset which involves compiling and labeling text documents into corpus, we collect the documents from Shamela library using tools available in Shamela program. The process includes converting document files into text format with UTF-8 Encoding using Zilla a word to text converter by software informer as shown in Figure 4.2.

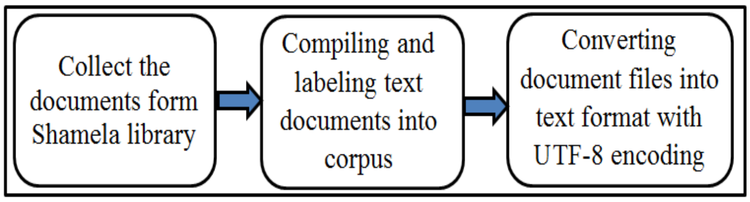

Collect the documents form Shamela library
Compiling and labeling text documents into corpus
Converting document files into text format with UTF-8 encoding

Figure 4.2: Corpus Building Steps

The collected Shamela corpus is categorized into eight main topics; Creed, Fiqh, Al-Hadith, History, Sirah, Tafsir, Trajem and Usual. This collection includes 101,647 text documents that constitute 5,310 MB in size.

4.3 The Parallel Classification as a MapReduce Model

The process of building the parallel Naïve Bayes classifier constitutes the core of our approach. It includes three main phases: text preprocessing phase, training phase, and testing phase. These phases are shown in Figure 4.3.

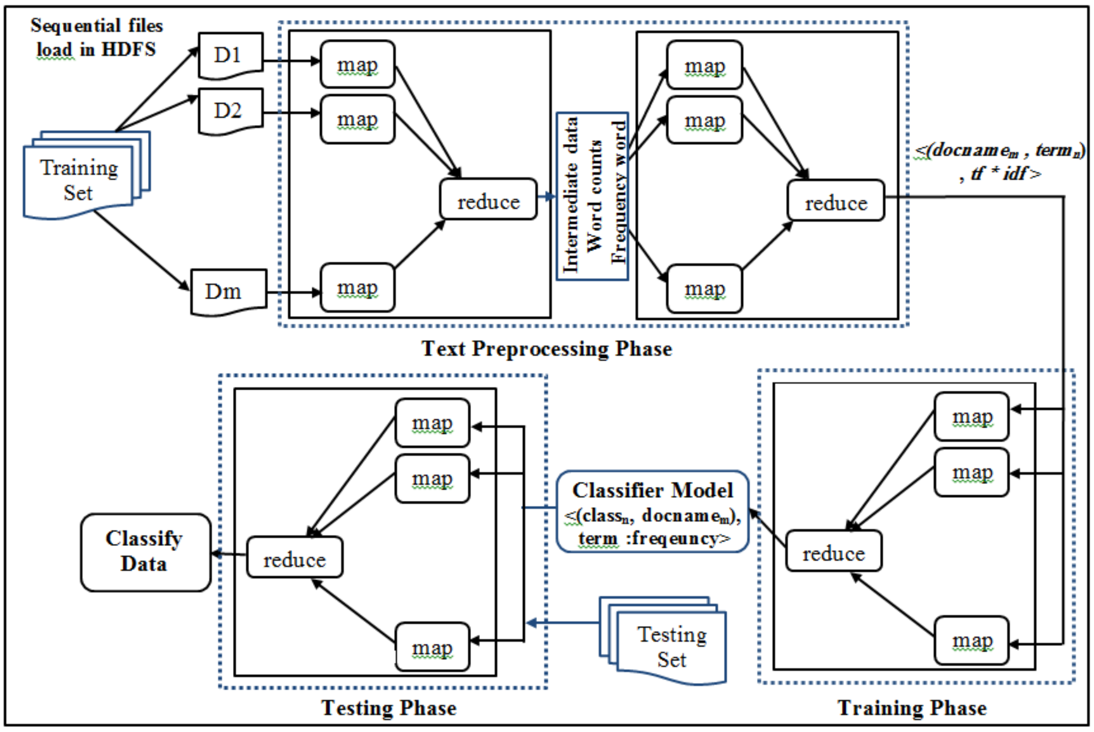

Sequential files load in HDFS
Training Set
D1
D2
Dm
map
reduce
Intermediate data Word counts Frequency word
<(docname$_m$, term$_n$), tf * idf>
Text Preprocessing Phase
Classifier Model <(class$_n$, docname$_m$), term :freqeuncy>
Classify Data
Testing Set
Testing Phase
Training Phase

Figure 4.3: The Proposed Parallel Classifier Approach

As a MapReduce processing model, in the first phase two steps are conducted: (i) the dataset *D* is divided into m subsets $\{D_1, D_2, \ldots, D_m\}$. (ii) the text preprocessing is performed using two MapReduce computations. One MapReduce for calculating the parameters required in the next MapReduce. The outputs of this step are *<(term, docname), n)>* pair, where n is the word count in document and *<(term, docname), (n, N)>* pair, where *N* is the frequency of word in documents which is the input of the next MapReduce. The second MapReduce computes Term Frequency- Inverse Document Frequency (TF-IDF) for each term and extracts terms to generate Vector Space Model (VSM), the output of this step is *<docname, (term, tf*idf)>* pair, where *tf*idf* is the weighting term value.

The second phase (as shown in Figure 4.3) has one MapReduce computation for training Naïve Bayes classifier to build the classifier model, the input of this process is the training set *<(class, docname), (term, tf*idf)>* pair resulted from the first phase and the output is *<(class$_n$, docname$_n$), (term: freqeuncy)>* pair as classifier model.

The third phase has one MapReduce computation for testing Naïve Bayes classifier, the input of this process is testing set and the classifier model resulting from second phase and the output is the classifying classes.

Next we present in details these three phases and their relationships based on the proposed approach as shown in Figure 4.3.

4.3.1 Text preprocessing phase

Text preprocessing is performed through two main steps: the first step includes removing non Arabic text, tokenizing string to words, stop words removal, and term stemming. We have written a specialized Java program to implement these steps in sequential manner because they are performed only once at the beginning of computation. The second step includes: pruning methods and terms weight processing using Mahout library in a distributed manner. Next we elaborate in these steps.

Applying text classification techniques requires usually a preprocessing stage that would remove punctuation marks, function words and might return the remaining words to their stems or roots. Figure 4.4 shows these detailed steps, they include removing non-Arabic text, tokenizing string to words, stop words removal, term stemming and pruning methods as a feature reduction techniques, and finally applying the suitable term weighting scheme to enhance text document representation as feature vector. These steps are details as follows:

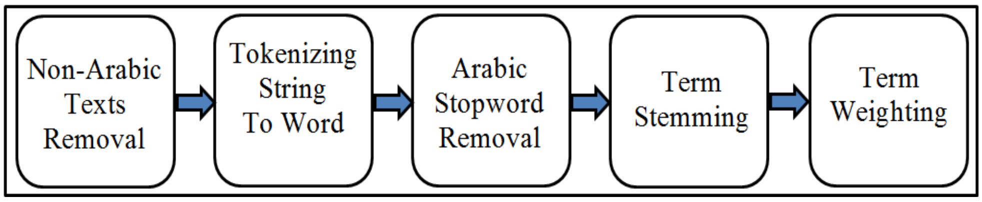


Figure 4.4: Text Processing Details

- All the non-Arabic texts such as the digits and punctuation marks, diacritics, numbers, non-Arabic letters, and removing kashida except in the term Allah were removed.
- Tokenization consists of separating strings by word boundaries, the Arabic Tokenization uses White Space Tokenization because the space is the only way to separate words in Arabic language, i.e. dash and hyphen are not used to separate words in Arabic.
- Arabic Stop word removal deletes tokens that are frequent, but generally not content-bearing.

- Term stemming (Section 3.3) is performed through light stemming because it allows remarkably good information retrieval without providing correct morphological analysis
- Term weighting (Section 3.4.2) is reflect the relative importance of each term in a document. It performed by using TF and TF-IDF terms as feature vectors to generate text representations.

4.3.1.1 Terms Weight Processing

We divide the collected corpus into several directories of text documents (classes). The master node converts them into a sequence files format. A sequence file is a Hadoop class which allows writing a document data in terms binary < key, value> pairs. Each sequence file is represented as a record in the corpus. Then master node uploads the files to Hadoop Distributed File System (HDFS) and in turn, they are distributed to worker nodes.

In order to better distinguish the documents from different categories, weight is assigned to every term (feature) for each document to formulate the terms weighting. We design two parallel MapReduce algorithms, one for calculating the parameter of terms and the other for calculating TF and TF-IDF of each term.

More details about calculating TF and TF-IDF are found in Section 3.4.2.

4.3.1.2 Parameters Computing

The first MapReduce computation involves two jobs; job 1 for calculating word frequency and job 2 for calculating word counts. We define term *t* as a word and *docname* as document name.

a. Word Frequency Calculation

Word frequency is the number of times the word appears globally in all documents. The data flow of the frequency count as a MapReduce is shown in Figure 4.5 and described in Algorithms 4.1 and 4.2.

The input to mapper function Algorithms 4.1 is *docname* as the key and *contents* as the value. The output is (*term, docname*) as the key and 1 as the value.

The output is written to an intermediate files which will be processed by the reducer function. Then we calculate the number of occurrences of word in document directly in the reduce function Algorithms 4.2.

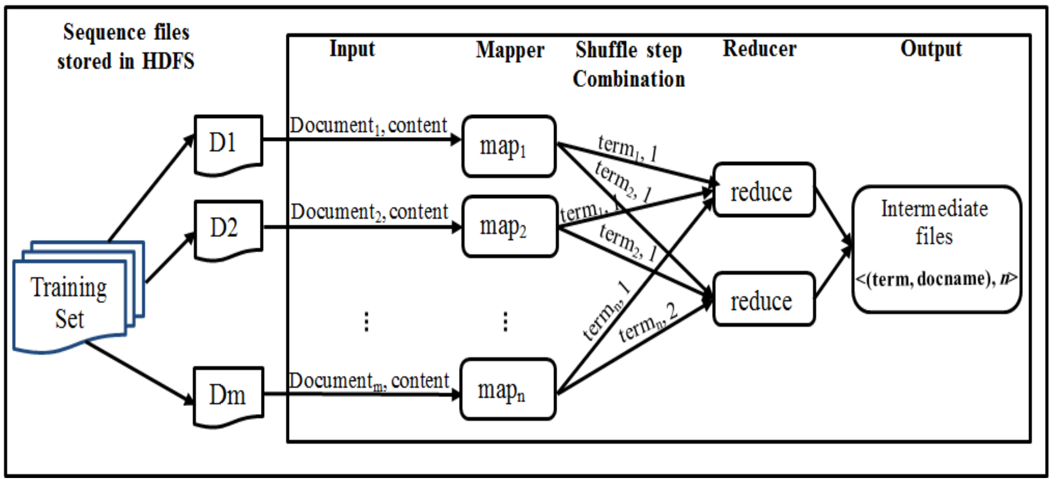


Figure 4.5: Data Flow of Word Frequency of MapReduce Job1

The output of reducer function is (*term*, *docname*) as the key and *n* as the value where *n* is the number of occurrences of the *term* (word) in *docname*.

Algorithm 4.1: Word Frequency Mapper-Job 1

```
input:
  key: docname;                  /* one text document for each map
  value: content;                /* all tokens 'word' of text document
output:
  key': (term, docname)        /* a text for each term;
  value': an integer one.
Begin
  separate all the <term, docname > pairs from the input value;
  for each <term, docname > pair;
     set  key'  as (term, docname);
     set value'  as 1;
     output(key', value');  /* each map data write in intermediate files
 end for.                        /* data exchange between nodes in shuffle process
End.
```

Algorithm 4.2: Word Frequency Reducer-Job 1

```
input:
  key: (term, docname);    /* intermediate data which is sort in same key
  value: 1 ;      /* a vector of integer one with the length that the term occurs.
output:
   key' : (term, docname); /* the same with key
  value': n;                  /* sum of integer one of  each term occurs in document
Begin
  initialize sum as zero;
   for each integer v in value;
       sum + = v ;
  end for;
  n =sum;
  set key'  as (term, docname);
  set  value'  as n;           /*  number of occurrences of the term in document
  output (key', value') ;  /* write the result in intermediate files
End.
```

b. Word Counts calculation

Word counts are the total number of term (word) of each document. The data flow of the frequency count of MapReduce is shown Figure 4.6 and described in Algorithm 4.3 and 4.4.

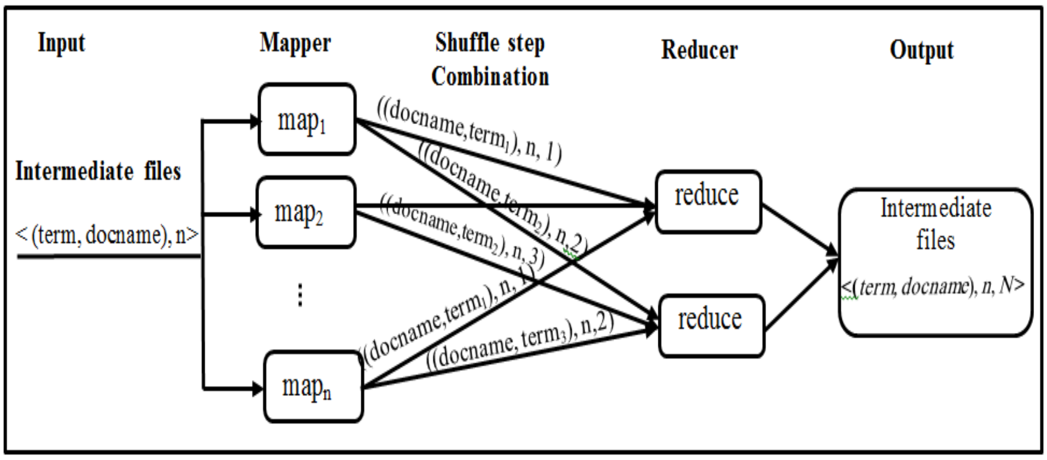


Figure 4.6: Data Flow of Word Counts MapReduce Job 2

The input to this mapper function Algorithm 4.3 is (*term, docname*) as the key and *n* as the value. The output is *docname* as the key and (*term, n*) as the value. The output is written to an intermediate files which is processed by reducer function. The reducer function Algorithm 4.4 computes the total number of frequencies of words in a document. The output of the reducer function is (*term, docname*) as the key and (*n, N)* as the value where *n* is the number of occurrences of the *term* (word) in *docname* and *N* is the total number of term (word) of each document.

Algorithm 4.3: Word Counts Mapper-Job 2

```
input:                          /*  the output of term frequency – reducer task
  key: (term, docname);
  value: n;
output:
  key' :(term, docname);
  value': (n,1);                /*  number of term 'word' of each document
Begin
    for each <(term, docname)> pair;
       set  key'  as (term, docname);
       set value'  as (n,1);
      output (key', value') ; /* each map data write in intermediate files
   end for.                     /* data exchange between nodes in shuffle process
End.
```

Algorithm 4.4: Word Counts Reducer-Job 2

```
input:
  key: (term, docname);    /* intermediate data which is sort in same key
  value: (n,1);
output:
   key':  the same with key ;
   value': sum of one (n,1) in value;
Begin
   initialize sum as zero;
   for each integer v in value;
       sum += v ;
   end for;
   N= sum;
   set key 'as (term, docname);
   set value' as (n, N);       /* the total number of word in each document
   output (key', value'); /* write the result in intermediate files
End.
```

4.3.1.3 Term Frequency Inverse Documents Frequency (TF-IDF) Computing:

The second MapReduce computation involves calculating the TF-IDF of each term. The data flow of TF-IDF MapReduce is shown in Figure 4.7 and described in. Algorithm 4.5 The input to this map function Algorithm 4.5 is the output of the first MapReduce (*term*, *docname*) as the key and (*n*, *N*) as the value. Then we calculate TF which is defined as (*n/N*) and IDF which is defined as (*log D/m*), where *D* is the total number of all documents and *m* is the sum of counts for words in the corpus. At last, we calculate the TF-IDF according the formula TF-IDF= *n/N* * log (*D/m*).

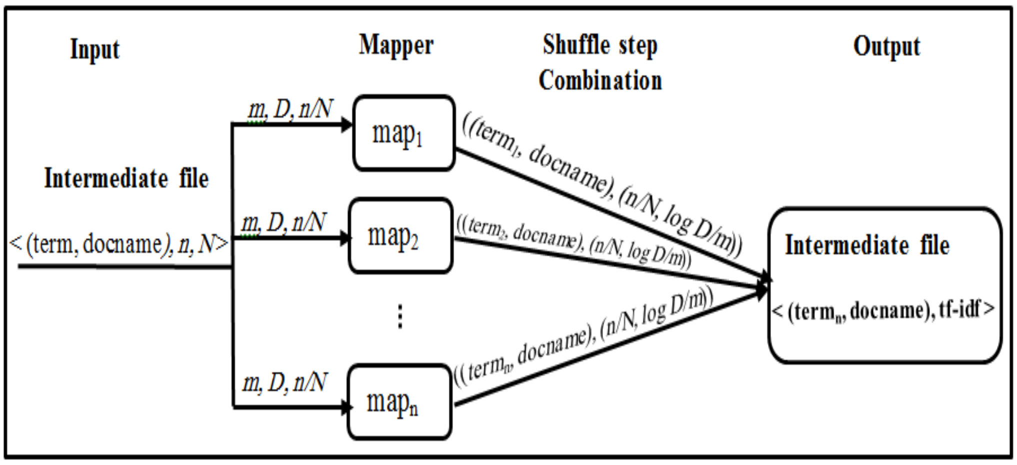


Figure 4.7: Data Flow of TF-IDF MapReduce

The output of the mapper function is the weight vector $w_{t,d}$. There is no reducer function in this MapReduce job. Map output is directly written out.

Algorithm 4.5: TF-IDF Mapper

```
Input:      /* the output of word counts reducer tasks
   Key: (term, docname);
   value: (n, N);
   D;  /* the number of documents
   m;  /* the number of word frequency in the corpus
Output:
   Key': docname;
   Value': (term, tf * idf );
Begin
  for each (term, docname) value
     tf =n/N;                          /* term frequency
     idf=log(D/m)                      /* inverse document frequency
     set key'  as  docname;
     set  value'   as(term, tf*idf);   /* the weight vector $w_{t,d}$.
     Output(key', value');  /* write the result to intermediate files
  end for.
End.
```

4.3.2 Training Phase

This phase has one MapReduce computation for the intensive run of the parallel Naïve Bayes classifier, which calculates the conditional probabilities. The data flow of the training MapReduce computation is shown in Figure 4.8 and described in Algorithm 4.6 and 4.7.

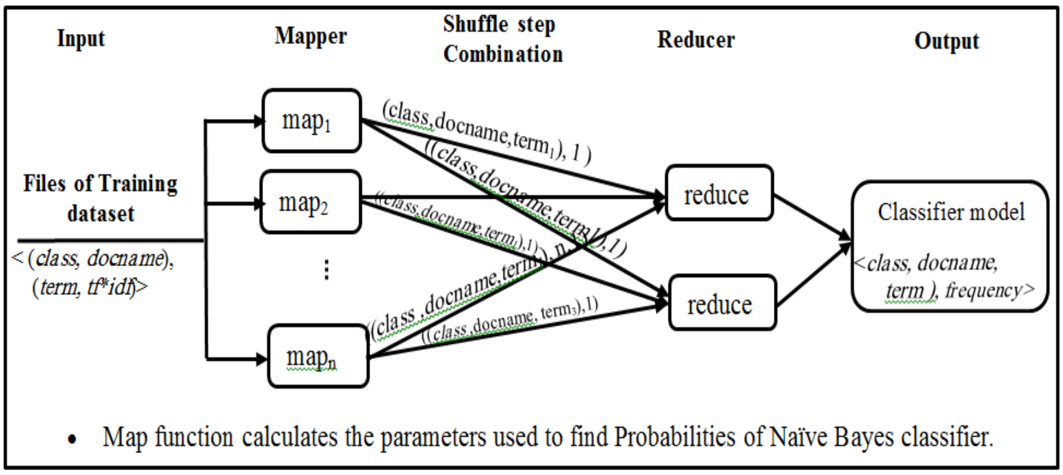

Input
Mapper
Shuffle step Combination
Reducer
Output
Files of Training dataset
<(class, docname), (term, tf*idf>
map1
map2
mapn
(class,docname,term1), 1)
reduce
reduce
Classifier model
<class, docname, term), frequency>
Map function calculates the parameters used to find Probabilities of Naïve Bayes classifier.

Figure 4.8: Data Flow of Naïve Bayes Training MapReduce

The input to the map function is the output file of the TF-IDF MapReduce computation using (*term, docname*) as the key and *tf-idf* as the value. In the training phase, the mapper function Algorithm 4.6 parsers the *class* and the value of each *term (*attribute). The output of the mapper function is a combination of *(class, docname, term, tf*idf)* as the key and *1* as the value. This output is written to intermediate files which is processed by the reducer function.

Algorithm 4.6: Training Naïve Bayes-Mapper

```
input:                    /* training dataset
  key: (class, docname);
  value: (term, tf*idf);
output :
  key': (class, docname, term, tf*idf);
  value': the frequency           /* the frequency of  term value
Begin
  for each  sample
     parse the class and the value of each term
     key': class;
     value': 1;
     output:<key', value'> pair;  /* count the frequency of each term in category
     for each (term, tf*idf)  value do
```

```
            contract a string as  (class, docname , term, tf*idf);
            set key'  as sting;
            set  value'  as 1;
            output:<key', value'> pair;   /* write the result to an intermediate files
      end for.
    end for.
End.
```

The reduce function Algorithm 4.7 counts the frequency of each key. The parameter of the Naïve Bayes classifier is calculated, including $P(c_j)$ and $P(A_i|c_j)$, where c_j denotes the *j*-th category, A_i the *i*-th attribute (term). The reducer function aggregates the number of term and category values, and results in form of ((*class, docname, term: count*$_1$), (*class, docname, term: count*$_2$), … (*class, docname, term*$_n$*: count*$_n$). This output constitutes the training model.

Algorithm 4.7: Training Naïve Bayes-Reducer

```
input :                          /* output by map function, respectively
   key: (class, docname, term, tf*idf);
   value: the frequency;
output:
   key': (class, docname, term, and  tf*idf);
   value': is the result of frequency;
Begin
   initialize a counter sum as 0 to record the current statistical frequency of the key;
   while(value .hasnext ())
       sum+= value. next().get();
       set key as  (class, docname, term, tf*idf);
       set value' as sum;             /* no of document having the term value
       output:<key', value'> pair;   /* write the result to an intermediate files
   end while.
End.
```

4.3.3 Testing Phase

This phase has one MapReduce computation for testing the parallel Naïve Bayes classifier. The data flow of the testing MapReduce is shown in Figure 4.9 and described in Algorithm 4.8 and 4.9

The mapper function Algorithm 4.8 indexes the key in the results produced by training phase and reads the corresponding probabilities. Then it calculates the probability of the test set belonging to each class. So the label can be predicted according to the maximum posterior. The output of the mapper function are *(label, correct)* as the key and *1* as the value and *(label, wrong)* as the key and *1* as the value. This output is written to intermediate files which is processed by the reducer function.

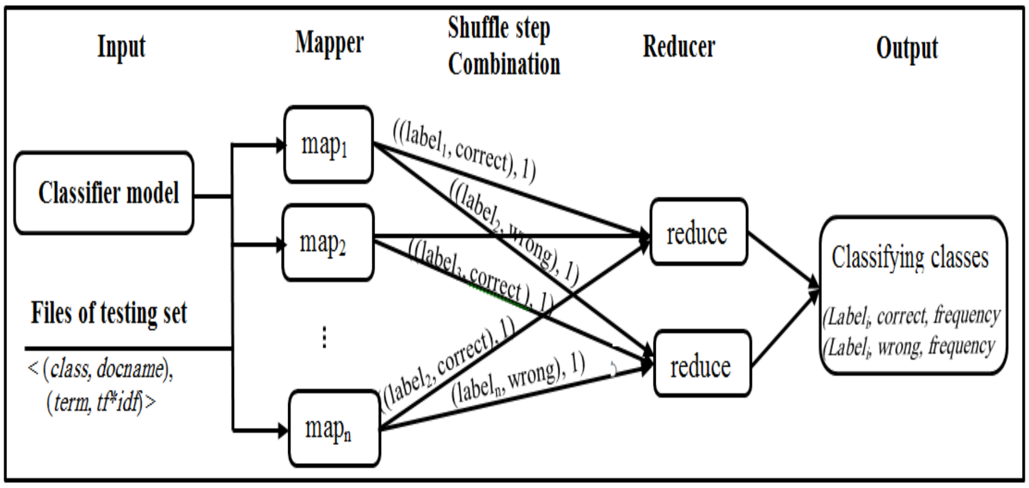


Figure 4.9: Data Flow of Naïve Bayes Testing MapReduce

Algorithm 4.8: Testing Naïve Bayes-Mapper

```
input:   the testing dataset and the output of  naïve Bayes training MapReduce is
         " training model"
output:
   key': label                    /* label is class has set of documents with terms
  value': the frequency
Begin
   parse the label and the value of each term; ;
  initialize an array prob[ ] , the length is set as the size of the testing set;
  for each label  in the testing set
      initialize prob[i] as 1.0;              /*  i is the index of the class in the testing set ;
      for each term  do
         initialize a string as label  with term name and its value ;
        index the string in the keys of the reduce result, record the corresponding value;
        prob[i]*=value ;
      end for
  end for
  index the class with the maximum value of prob;
  if the label  is the same to that docname  take "correct" as key' and 1 as value';
  output :<value', value' > pair;
  else take "wrong" as key' and 1 as value';
  output :<key', value'> pair;
End.
```

The reduce function Algorithm 4.7 states the number of the correct or wrong predicted set. Therefore, the correct rate and error rate can be further calculated. The reducer function aggregates the number of correct label and wrong label of predict set, and results in form of ((*label*, (*correct*, *frequency*)), and ((*label*, (*wrong*, *frequency*)). This output constitutes the classifying classes.

Algorithm 4.9: Testing Naïve Bayes-Reducer

```
input : key, value (key', value' output by map function, respectively)
output:
key': label;
value': the result of frequency for correct key and the result of frequency of wrong key;
Begin
  initialize a counter sum1 as 0 to record current  frequency of the correct key;
  initialize a counter sum2 as 0 to record current  frequency of the wrong key;
  while(value .hasnext ())
      sum1+= value. next().get();
      sum2+= value. next().get();
      set key as  key';
      set value' as sum1 and sum2;
      output:<key', value'> pair;          /* the output of reduce is the  predict classes
  end while.
End.
```

4.4 Summary

In this chapter, we presented the proposed parallel classification approach based on MapReduce model. We used two parallel MapReduce algorithms to calculate the terms weighting; one for calculating the parameter of terms and the other for the term frequency (TF) and term frequency- inverse document frequency (TF-IDF) of each term. Also, we used two parallel MapReduce algorithms, one for training phase and the other for testing phase.

In the next chapter, we present and discuss the experiments carried out to realize and evaluate the proposed classifier.

Chapter 5 Experimental Results and Analysis

In this chapter we present and analyze the experimental results to provide evidence that our parallel classification approach can enhance speedup, performance and preserve the accuracy of the classification. Parallel Naïve Bayes classifier (as described in Section 3.1.1) is used in our experiments which is provided as part of Mahout library (see Section 3.8). First, we present the corpus used in our experiments and give insight into the main characteristics of it. Then we explain the experimental environment and the implementation of the parallel MapReduce Naïve Bayes classifier using Mahout library. We calculate the different measure of speedup and accuracy. Finally we present and discuss the experimental results and make a comparison with the MPI based parallel K-NN classifier for large scale Arabic text [20] (see Section 2.2).

5.1 The Corpus

We used Shamela[1] as the source of our corpus, where we collected as 101,647 text documents that constitute 5,310 MB in size, and 5,100 MB after stop words removal. Each text document belongs to 1 of 8 classes (Creed, Usual, Fiqh, Hadith, History, Seerah, Tafsir, and Trajem) as shown in Table 5.1.

We perform all text preprocessing (Section 4.3.1.1) on the corpus. This includes non-Arabic text removal, tokenizing string to word, Arabic stop word removal, term stemming and term weighting. Specifically, the generated preprocessed corpus undergoes the following representations:

- Light Stemming + TF-IDF
- Light Stemming + TF
- Stemming + TF-IDF
- Stemming + TF

These representations are needed for the classification experiments. More details about text representation are described in Section 3.4.

[1] *http://shamela.ws*

Table 5.1: The Shamela Corpus

Category	Number of Text Document	Size of Text Document(MB)
Creed	6,776	373
Usual	2,245	128
Fiqh	22,405	1180
Hadith	23,530	1200
History	9,232	488
Seerah	4,641	240
Tafsir	18,048	973
Trajem	14,722	784
Total	**101,647**	**5,310 MB**

5.2 Experimental Environment

The experimental environment is built on a MapReduce cluster with 16 machines. One machine acts as NameNode and the other 15 machines act as DataNodes implemented as virtual machines. All the virtual machines have the same configuration; Intel Core2 Quad CPU at 2.5 GHz, 4.00 GB RAM, 320 GB hard disk drive and operating system is Ubuntu 12.4 Linux with Java JDK 1.6.0, and Hadoop version 1.2.0. The number of replicas is set to 3 and the HDFS block size is 64MB. All computers connected through local area network with speed of 10/100 Mbps.

The proposed parallel classifier approach has been implemented on Hadoop cluster with Ubuntu 12.4 operating system and Naïve Bayes classifier available in Mahout framework, which is highly scalable with large scale data.

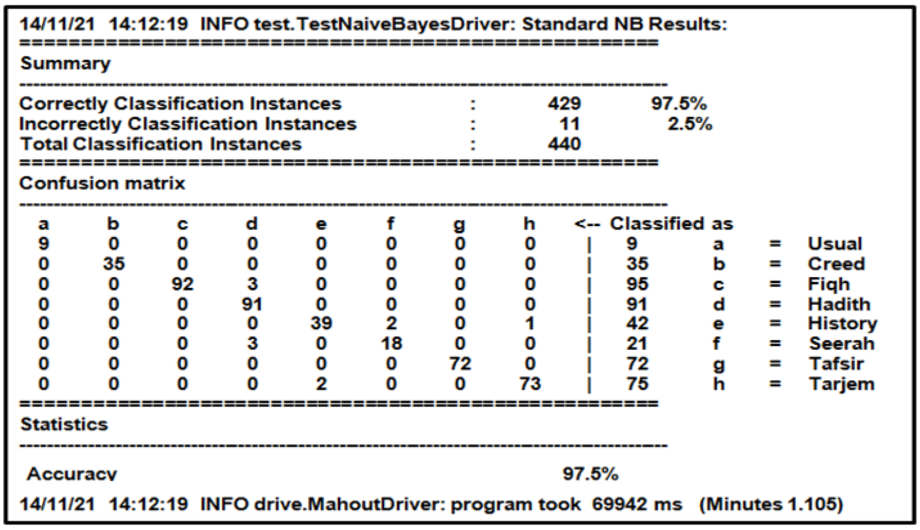
5.3 Implementing the Parallel Naïve Bayes Classifier in Mahout

The proposed parallel MapReduce NB classifier utilizes Hadoop distributed data processing platform, and parallelized NB classification utilizes Mahout library as a MapReduce realization of Arabic documents.

We follow the steps in [65] for building Hadoop cluster with Hadoop version 1.2. For the implementation of the parallel Naïve Bayes classifier using Mahout library, we follow the steps in [66, 67].

The process of realizing the overall classification approach involves the following steps:

- Step 1: All text preprocessing (see Section 4.3.1.1) is performed on Shamela corpus. It is saved as text files directories into *NameNode* then uploaded to HDFS. HDFS divides the input Arabic text files documents into data blocks of size 64 MB (i.e. by default). It stores the metadata of each block in the *NameNode* (Master Node) and all the data blocks in the *DataNodes* (Slave Nodes).
- Step 2: the directories containing the text files are converted into Hadoop sequence files format. The Naïve Bayes algorithm does not work directly with the words and the raw text, but with the weighted vector associated to the original document.
- As the last step in preprocessing phase the terms weight TF and TF-IDF are performed in parallel MapReduce based to form vectors files from the sequence files.
- Step 3: Split the vector files into training set and testing set. In our experiment we randomly selected 50%, 30%, and 20% of vector files from the whole corpus as the testing sets and the remaining percentage of the file vectors as the training sets.
- Step 4: the training phase is conducted a parallel NB classifier on the training set. The output of this step is Naïve Bayes classifier model in the form of binary files.
- Finally step , the testing phase is conducted to test the Naïve Bayes Classifier model on the testing set has a small number of large files. The performance of the model with the testing set by Mahout’s command, which produces the confusion matrix shown in Figure 5.1.



```
14/11/21  14:12:19  INFO test.TestNaiveBayesDriver: Standard NB Results:
=======================================================
Summary
-------------------------------------------------------
Correctly Classification Instances        :        429        97.5%
Incorrectly Classification Instances      :         11         2.5%
Total Classification Instances            :        440
=======================================================
Confusion matrix
-------------------------------------------------------
  a     b     c     d     e     f     g     h    <-- Classified as
  9     0     0     0     0     0     0     0     |   9     a    =   Usual
  0    35     0     0     0     0     0     0     |  35     b    =   Creed
  0     0    92     3     0     0     0     0     |  95     c    =   Fiqh
  0     0     0    91     0     0     0     0     |  91     d    =   Hadith
  0     0     0     0    39     2     0     1     |  42     e    =   History
  0     0     0     3     0    18     0     0     |  21     f    =   Seerah
  0     0     0     0     0     0    72     0     |  72     g    =   Tafsir
  0     0     0     0     2     0     0    73     |  75     h    =   Tarjem
=======================================================
Statistics
-------------------------------------------------------

 Accuracv                                          97.5%
14/11/21  14:12:19  INFO drive.MahoutDriver: program took  69942 ms   (Minutes 1.105)
```

Figure 5.1: The Result of Running Parallel Naive Bayes Classifier Using Mahout

Figure 5.1 shows the classification accuracy value (97.5%) for a small large files, which indicates that the classification is highly accurate. Also it shows the execution time takes for running parallel Naïve Bayes classifier.

5.4 Experimental Results and Discussion

This section presents the results of experiments that have been performed.

5.4.1 The Parallel Classification and its Performance

In the experiments, we use the collected corpus of 101,647 documents that are represented as records and 4096 words that are represented as attributes. We evaluate the performance of the parallel classifier with respect to the execution time and speedup (as described in Section 3.9.2). For evaluation purposes, we follow the steps described in Section 5.3 to split the largest generated text representation for the corpus into the training set and the testing set.

To measure the speedup, we have executed the parallel classifier on a system of nodes varied from 2 to 16. Also we used different number of testing documents to observe the effects of different problem (documents) sizes on the performance. Three sets were used with the number of tested documents 20329, 30459, and 50329 documents.

The parallel algorithm demonstrates essentially linear speedup. When running an algorithm with linear speedup, doubling the number of nodes doubles the speedup.

Table 5.2: The Execution Times (sec.) of One Node and Multip-Node Parallel Classifier

No. of Nodes \ Problems Size		20329 Documents	30459 Documents	50823 Documents
Standalone	**1-Node**	277.56	367.20	629.64
Parallel Classifier	**2- Nodes**	146.88	176.04	259.20
	4- Nodes	136.08	153.36	221.40
	6-Nodes	72.36	89.64	143.64
	8-Nodes	46.44	58.32	88.45
	10-Nodes	44.28	55.08	86.40
	12-Nodes	43.20	52.92	79.92
	14-Nodes	42.12	48.60	70.20
	16-Nodes	38.56	44.60	52.92

In particular, linear speedup is difficult to achieve because the communication cost increases as the number of documents increases. Table 5.2 shows the execution time in seconds for different documents sizes on various numbers of nodes.

Table 5.2, shows the execution time of one node with MapReduce takes more time than the parallel version. In the parallel version, the execution time decreases when the number of processors increases. However, the parallel implementation achieves a good execution time compared to that of one node. In addition, the execution time increases when the number of documents increases. Figure 5.3 shows the curves of the execution time based on Table 5.2

Several observations can be made on these results. First, the sequential NB algorithm is inappropriate for experiment, because the large scale of text document. Second, the parallel NB classifier clearly decreases the classification time than one node it takes 52.92 seconds on 16 nodes. Notes that the time of one node with MapReduce takes about 10.49 minutes.

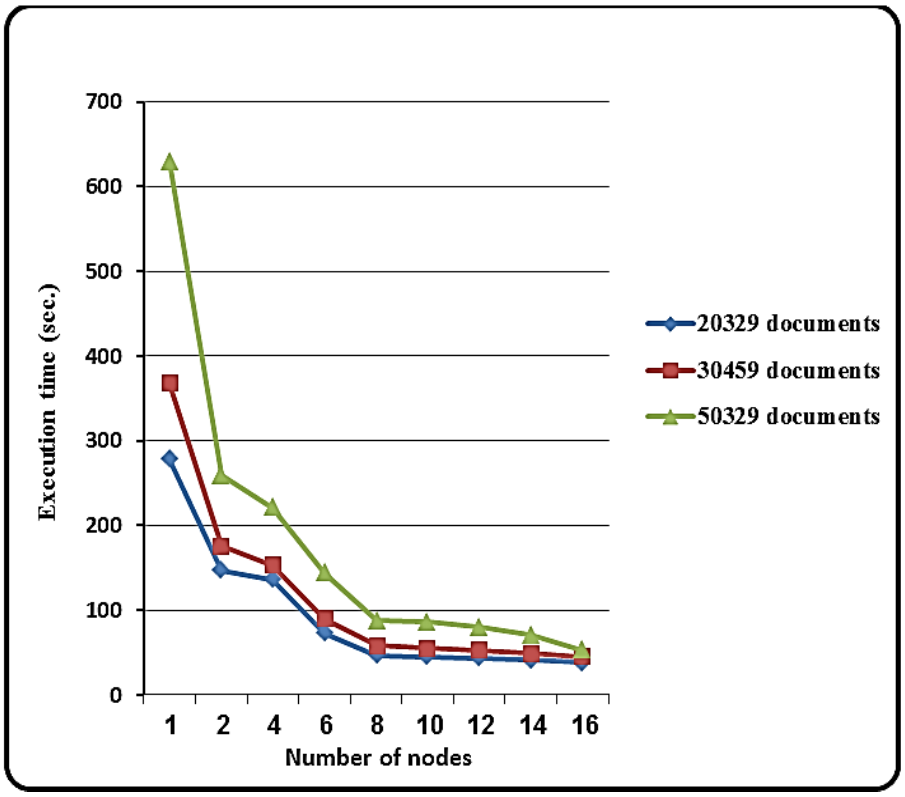


Figure 5.2: Execution Time for the two Classifiers

Moreover, the time that the parallel NB classifier spends does not appear to have a linear relationship with nodes. This is due to the fact that when running Hadoop jobs, starting a cluster first take some time. So when the size of data set is small, the processing time is relatively longer. In addition the execution time of parallel classifier on 8 nodes to 16 nodes has a few changes.

Also, we compute the speedup with the formula $S_n = t_s / t_p$, where t_s is the execution time using only one node and t_p is execution time using n node which is gained from this parallelization as described in Section 3.9.2. The speedup is recorded in Table 5.3 and is illustrated in Figure 5.4.

Table 5.3: The Relative Speedup of the Proposed Parallel Classifier

Problems Size / No. of Nodes	20329 Documents	30459 Documents	50823 Documents
2- Nodes	1.89	2.09	2.43
4- Nodes	2.04	2.39	2.84
6-Nodes	3.84	4.10	4.38
8-Nodes	5.98	6.30	7.12
10-Nodes	6.27	6.67	7.29
12-Nodes	6.43	6.94	7.88
14-Nodes	6.59	7.56	8.97
16-Nodes	7.20	8.23	11.90

These results show that the NB classifier has high speedup. Specifically, as the size of records increases, the speedup improves. Therefore, the parallel NB classifier can treat large scale Arabic text documents efficiently.

The speedup improves in some cases. For example, on the largest tested set (50823 documents), it achieves the relative speedups of 2.43, 2.84, 7.12 and 11.90 on 2, 4, 8, and 16 nodes, respectively. When a small set of tested documents are used, the speedup tend to drop from the linear to sub-linear. The classifier achieves the relative speedups of 1.89, 2.04, 5.98, and 7.20 on 2, 4, 8, and 16 nodes respectively. The smallest tested documents sizes give similar results.

If we increase the number of nodes further, the speedup gains tend to significantly drop. Figure 5.4 shows, the speedups for three documents sets. On 4 nodes the speedup improves from 2.04 to 2.84, on 8 nodes it improve from 5.98 to 7.12, and on 16 nodes it improves from 7.20 to 11.90. It can be shown that our parallel classifier gives better performance with larger volume Arabic text documents than with smaller volume Arabic text documents.

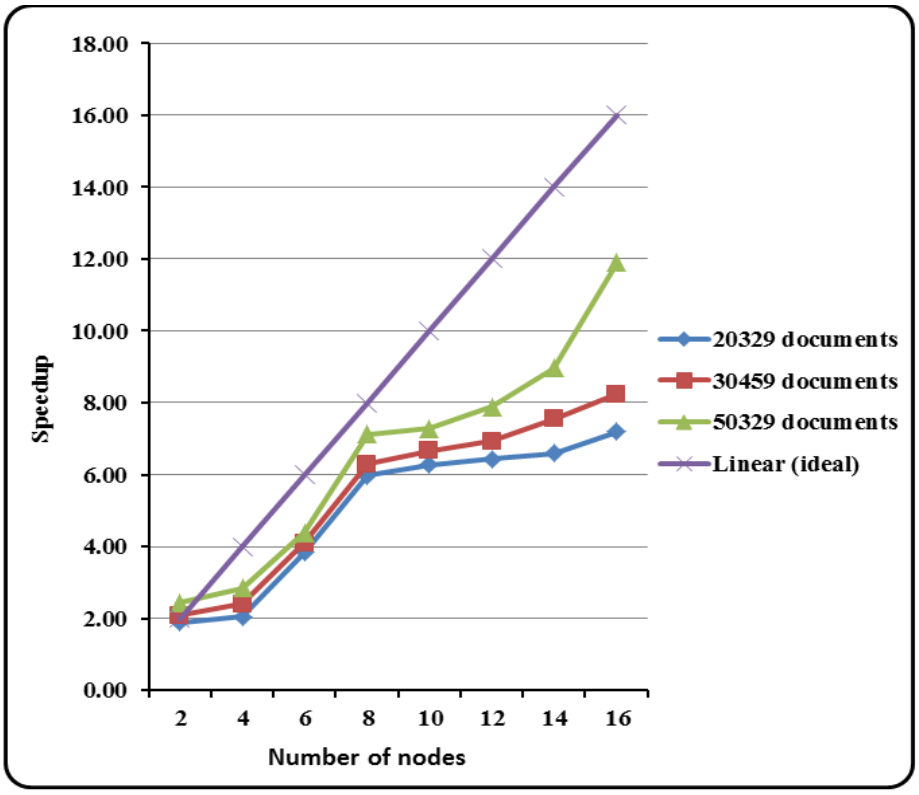


Figure 5.3: The Relative Speedup of the Proposed Parallel Classifier

5.5 Evaluating Quality of the Classification

To ensure that the classifier works well with the tested documents, we tested the quality of the classification. For the purpose of evaluating the classification results, we use confusion matrices (described in Section 3.9.1). We have evaluated the obtained classification results using different classification measures such as accuracy (Eq. 3.1), precision (Eq. 3.2), recall (Eq. 3.3), and F-measure (Eq. 3.4) which are generally common ways of measuring system performance in this field.

We have conducted two experiments, one with a large number of small files and the other with a small number of large files. This is done to overcome the performance problem of HDFS caused by small files size as described in Section 3.6.1

5.5.1 Text Classification Performance with a Large Number of Small Files

In our experiment, we split all generated text representations of Shamela corpus into two parts; 50% of the corpus for training (50833 documents with 4K attributes for each document) and the remaining 50% for testing (50833 documents). We split the corpus in this way to achieve higher classification results. We computed the accuracy, precision, recall, and F-measure for all generated text representations of the corpus and the results are recorded in Table 5.4.

Table 5.4: Classification Results for all Text Representations of Small Files

Performance Measures / Text Representations	Accuracy	Precision	Recall	F-measure
Light Stemming + TF-IDF	84.86	78.8	81.1	79.4
Light Stemming + TF	82.75	74.8	78.9	75.5
Stemming +TF-IDF	83.21	78.5	80.8	79.1
Stemming + TF	81.30	74.5	78.6	75.2

Figure 5.4 illustrates the classification results for all text representations of small files.

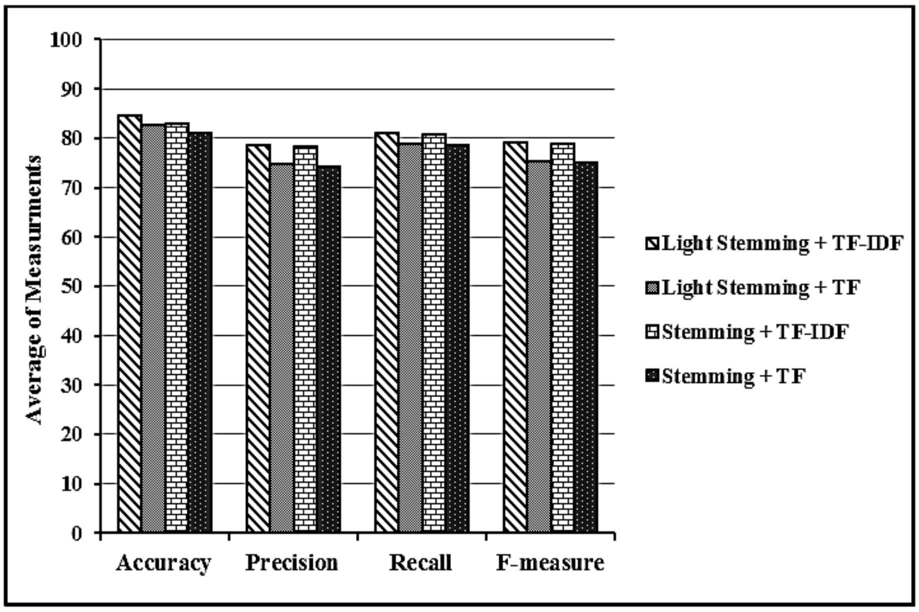


Figure 5.4: Classification Results for all Text Representations of Small Files

To summarize the average classification performance results in NB classifier, the morphological analysis (stemming, light stemming) and term weighting schemes (TF-IDF, TF) have obvious impact on the classifier performance.

The difference in the accuracy and F-measure results is based on the highest and lowest values obtained. Accuracy: 84.86 % - 81.30% = 3.56%, F-measure: 79.4% -

75.2% = 4.2%. This emphasizes that the performance of the classifier greatly depends on the actual representation of the text to be classified. The result shows that, the highest average of accuracy is achieved using light stemming and TF-IDF (84.75%), while using stemming and TF (81.30%) give the lowest average of accuracy.

5.5.2 Text Classification Performance with a Small Number of Large Files

In this experiment, we split all generated text representations of the corpus into two parts; 50% of the corpus for training (440 documents with 512K attributes for each document) and the remaining 50% for testing (440 documents). We computed the accuracy, precision, recall, and F-measure for all generated text representations of the corpus and the results are recorded in Table 5.5.

Table 5.5 shows that, the highest accuracy result (97.50%) is when using light stemming and TF-IDF text representations and the lowest accuracy result (89.86%) is when using stemming and TF text representations.

Table 5.5: Classification Results for all Text Representations of Large Files

Performance Measures / Text Representations	Accuracy	Precision	Recall	F-measure
Light Stemming + TF-IDF	97.50	97.20	96.59	96.87
Light Stemming + TF	96.87	96.38	97.91	97.10
Stemming +TF-IDF	92.27	85.77	88.25	84.07
Stemming + TF	89.86	86.71	83.59	83.54

Figure 5.5 illustrates that light stemming and TF-IDF representation has the best classification results and increases the classification accuracy.

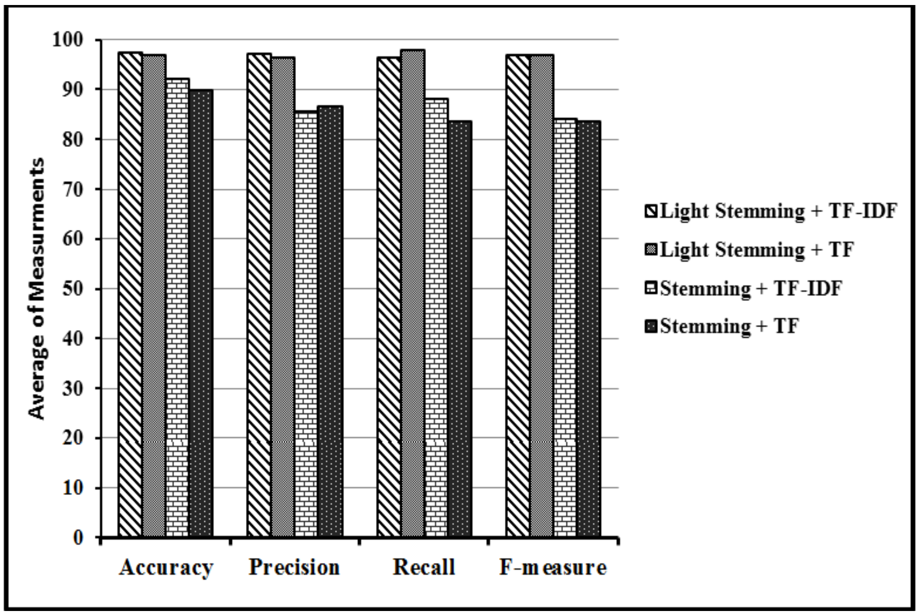


Figure 5.5: Classification Results for all Text Representations of Large Files

After conducting the two experiments, we can note that, there is a great difference in improvement of accuracy and F-measure results conducted on the two experiments. The accuracy result on large numbers of small files is 84.86%, while on small numbers of large files was improved to 97.50% as shown in Figure 5.6.

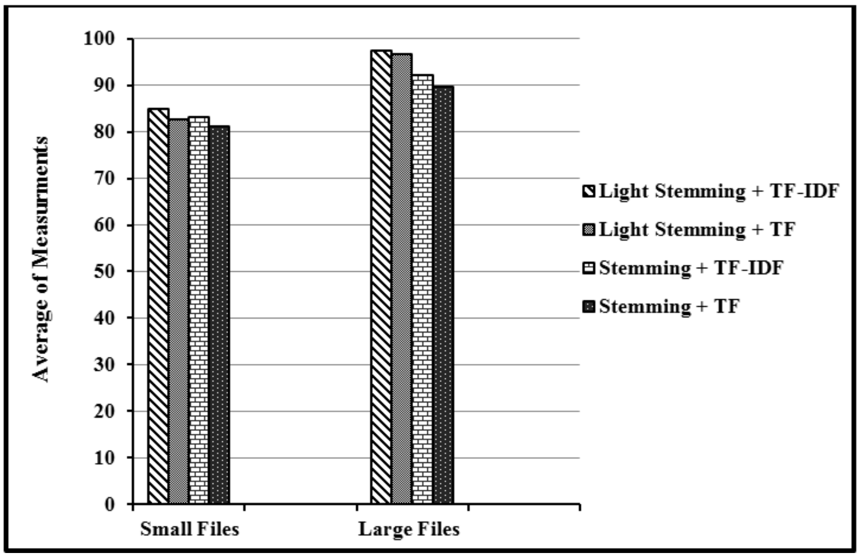


Figure 5.6: Accuracy of Small Files and Large Files Classification

In addition, the execution times for the parallel NB classification in the two experiments are decreased with a small number of large files as shown in Table 5.6 and Figure 5.7.

As the file size increases, the CPU time and MapReduce time are decrease. Also it confirms that, there is improvement in the performance of 48.4% and 71.2% of execution times of CPU and MapReduce respectively.

Table 5.6: Execution Times of a Small Number of Large Files and a Large Number of Small Files

Technique / File Size	CPU Time in Millisecond	MapReduce Time in Millisecond
38KB	132400	162396
4MB	46155	27270

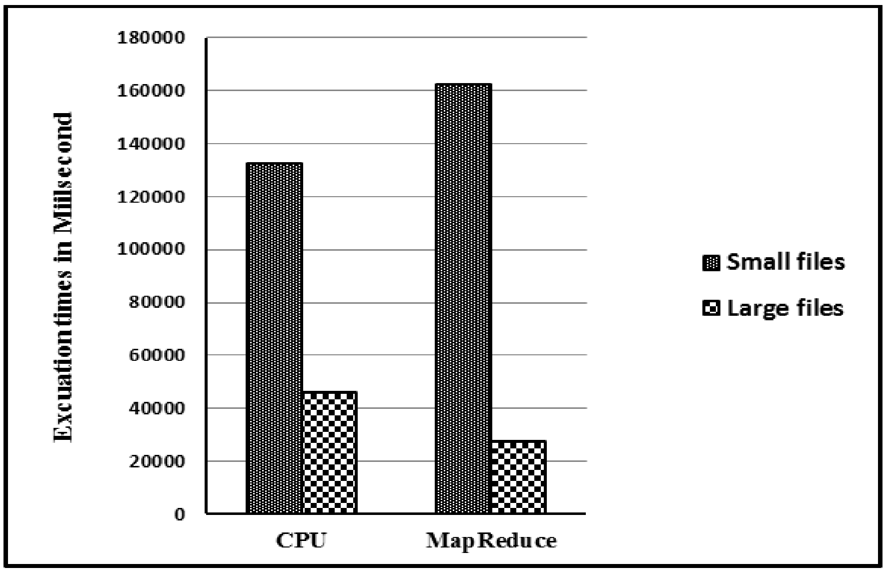


Figure 5.7: Execution Times of CPU and MapReudce with Small and Large Files

The performance for each class of the corpus for the best text representation (light stemming + TF-IDF) that achieved the best classification results is shown in Table 5.7.

Table 5.7: Classification Results for Light Stemming and TF-IDF

Category \ Performance	Precision	Recall	F-measure
Creed	91.67	94.3	93.0
Feiqh	97.80	93.7	95.7
Hadith	93.81	100.0	96.8
History	95.12	92.9	94.0
Seerah	90.00	85.7	87.8
Tafsir	100.00	97.2	98.6
Tarajm	98.65	97.3	98.0
Usual	81.82	100.0	90.0

Figure 5.8, depicts the performance for the domains: Tafsir domain has the highest performance F-measures (98.6%), because Tafsir has a small size of words that are limited and are clearly compared to other domains. Also, it shows that Seerah domain has lowest performance F-measure (87.8%) and this is because Sirah has a large space domain.

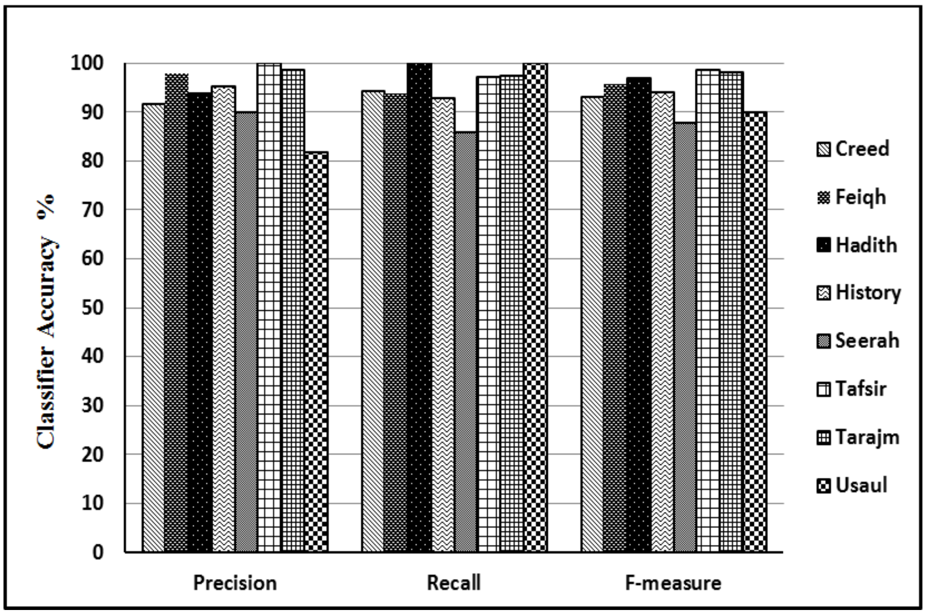


Figure 5.8: Classification Results for Light Stemming and TF-IDF

5.6 Comparison with Related Approaches

To complete the evaluation of our Parallel Naïve Bayes classifier, we compare it with the MPI-based parallel approach [20] along nine criteria which are the most common criteria show in Table 5.8. The most important criteria in the comparison are the size of data, the parallel platform, the programming model, and the speed up which obviously is affected by these criteria. We mention this comparison to show that using the MapReduce model regardless the kind of classification algorithm can improve speedup significantly.

M. AbuTair and R. Baraka in [20] (see Section 2.2), proposed a parallel classifier for large scale Arabic text documents. The parallel algorithm is based on the K-NN algorithm. They evaluated the parallel implementation on a multiprocessor cluster that consists of 14 computers with shared memory. They experimented with a 214 MB dataset. The speedup results were relative up to 14 processors.

The comparison between our approach and the MPI-based approach is summarized in Table 5.8.

Table 5.8: The Comparison Between MapReduce Model Parallel Approach and MPI-Based Parallel Approach

Criteria	MapReduce Model Parallel Approach	MPI-based Parallel Approach
Language	Mahout java project	C++
Size of dataset	5138 MB	241 MB
Type of dataset	Shamela corpus	OSAC Arabic corpus
Number of nodes	2, 4, 8, 12, 14, 16 nodes	2, 4, 8, 12, 14 nodes
Execution times (sec)	259, 221.4, 88.4, and 52.9 on 2, 4, 8, and 16 nodes	1914, 997.9, 566, and 398.6 on 2, 4, 8, and 14 processors
Speedup	2.43, 2.84, 7.12, and 11.90 on 2, 4, 8, and 16 nodes	1.87, 3.59, 6.33, and 9.00 on 2, 4, 8, and 14 processors
Parallel platform	Hadoop Cluster	A multicomputer cluster
programming model	HDFS	MPI
The processor speed	2.5 GHZ	3.30 GHz
The memory size	4GB	4GB

The comparison is done along ten criteria: the programming language, the size of the corpus, the type of corpus, the number of nodes, the execution times, the speedup, the parallel platform, the programming model, the processor speed, and the memory size.

Our work is significantly different, because the corpus is 21.3x times larger, and the tested set (50833 documents *4096 attributes), is 6.4x times larger. Moreover the time spent for classification is 7.5x times smaller, and the parallel classifier achieved the relative speedup of 11.90 on 16 processors. Our approach is a scalable parallel system because the efficiency can be kept constant as the number of processing elements is increased provided that the problem size is increased (from 20329 documents to 50833 documents). Our dataset contains 101,647 * 4069 values, the size of the dataset is 5138 MB.

5.7 Summary

This chapter presented and analyzed the experimental results. It presented the corpus characteristics, explained the machine environment, and implementation of the parallel NB classifier using Mahout Library. Also, it presented experimental results of parallel classification and its performance. The evaluation of the quality of the classification model during sets of experiments. Finally, we compared our parallel Naïve Bayes classifier approach with an MPI-based parallel approach.

Chapter 6 Conclusion and Future Work

Text classification of large-scale text documents is an important research in text mining. Sequential Naïve Bayes classifier is a popular machine learning for text classification, widely applied, fast and easy to classify Arabic text documents. However, it takes more time when used in classifying large scale of text documents.

We proposed a parallel Naïve Bayes classifier for large-scale Arabic text document based on MapReduce. It involves Arabic text documents collection, Arabic text preprocessing, design the suitable MapReduce computing model for parallel classification as a Hadoop platform, implementation the parallel Naïve Bayes algorithm using Mahout library over the designed MapReduce computing model.

We tested the parallel classifier using a large scale Shamela-sourced corpus which is the largest Arabic corpus of text documents. The test is performed on Hadoop cluster consisting of 16 nodes as a MapReduce model.

For evaluation purposes, we use accuracy, precision, recall, and F-measure to evaluate the classification of our approach and speedup to evaluate its performance.

The results show that the proposed parallel NB classifier approach can significantly improves speedup up to 12x times better than the sequential approach using the same classification algorithm and preserve accuracy up to 97%.

Also we compare our approach with MPI-based approach [20]. The result shows that our proposed parallel NB classifier approach is 7.5x times faster, and processes large scale of Arabic text documents is 21.3x times larger on commodity hardware effectively.

The proposed approach can be used efficiently and accurately to classify a large scale of Arabic text with high dimensionality and solved the problem of low speed, and preserve high accuracy for the sequential NB algorithm.

There are many directions for improvements and future investigations. Our work can be extended to cover larger computer clusters with larger volume of Arabic text documents that constitute more than one terabytes in size. Also, other parallel classification algorithms can be applied with our approach to investigate their effectiveness and performance with large scale Arabic text. Additionally, our approach can be applied to other domains such as medical information, weather data, and social media among others to check its generalization. It can also be used as online classification approach with web data. Finally, the work can be applied with other cloud-based technologies such as big data analytics, where data mining algorithms can be used with big data techniques over MapReduce model to speed up the process and give accurate results.

References

[1] R. Feldman and J. Sanger, 'The Text Mining Handbook Advanced Approaches in Analyzing Unstructured Data'. Cambridge; New York: Cambridge University Press, 2007.

[2] J. Han and M. Kamber, 'Data Mining Concepts and Techniques'. Amsterdam; Boston; San Francisco, CA: Elsevier ; Morgan Kaufmann, 2006.

[3] S. Kim, K. Han, H. Rim, and S. Myaeng, 'Some Effective Techniques for Naïve Bayes Text Classification', IEEE Trans. Knowl. Data Eng., vol. 18, no. 11, pp. 1457–1466, Nov. 2006.

[4] T. Liu, Z. Chen, B. Zhang, and G. Wu, 'Improving Text Classification Using Local Latent Semantic Indexing', in Fourth IEEE International Conference on Data Mining, 2004. ICDM ’04, pp. 162–169, 2004.

[5] M. Missen and M. Boughanem, 'Using WordNet’s Semantic Relations for Opinion Detection in Blogs', in Advances in Information Retrieval, Soule-Dupuy, Eds. Springer Berlin Heidelberg, pp. 729–733, 2009.

[6] A. Balahur and A. Montoyo, 'A Feature Dependent Method for Opinion Mining And Classification', in International Conference on Natural Language Processing and Knowledge Engineering, NLP-KE ’08, pp. 1–7, 2008.

[7] M. S. Khorsheed and A. Al-Thubaity, 'Comparative Evaluation of Text Classification Techniques Using A large Diverse Arabic Dataset', Lang. Resour. Eval., vol. 47, no. 2, pp. 513–538, Mar. 2013.

[8] F. Sebastiani, 'Machine Learning in Automated Text Categorization', ACM Comput Surv, vol. 34, no. 1, pp. 1–47, Mar. 2002.

[9] S. Alsaleem, 'Automated Arabic Text Categorization Using SVM and NB', Int Arab J E-Technol, vol. 2, no. 2, pp. 124–128, 2011.

[10] M. Elkourdi, A. Bensaid, and T. Rachidi, 'Automatic Arabic Document Categorization Based on the Naïve Bayes Algorithm', in Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages, pp. 51–58, 2004.

[11] B. Wang and S. Zhang, 'A Novel Text Classification Algorithm Based on Naïve Bayes and KL-Divergence', in Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005, pp. 913–915, 2005.

[12] S. Liang, Y. Liu, C. Wang, and L. Jian, 'A CUDA-Based Parallel Implementation of K-Nearest Neighbor Algorithm', in Cyber-Enabled Distributed Computing and Knowledge Discovery,2009. CyberC’09. International Conference on, pp. 291–296, 2009.

[13] J. Dean and S. Ghemawat, 'MapReduce: Simplified Data Processing on Large Clusters', Commun ACM, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[14] T. White, 'Hadoop: The Definitive Guide', 3rd Edition, Storage and Analysis at Internet Scale. O'Reilly Media /Yahoo Press, 2012.

[15] A. Al-Thubaity, N. Abanumay, S. Al-Jerayyed, and Z. Mannaa, 'The Effect of Combining Different Feature Selection Methods on Arabic Text Classification', in (SPND), 2013 14th ACSI International Conference pp. 211–216, 2013.

[16] B. Al-Salemi and M. Ab Aziz, 'Statistical Bayesian Learning for Automatic Arabic Text Categorization', J. Comput. Sci., vol. 7, no. 1, 2011.

[17] W. Ding, Q. Wang, and Q. Guo, 'A Novel Naive Bayesian Text Classifier', in 2008 International Symposiums on Information Processing (ISIP), pp. 78–82, 2008.

[18] F. Viegas, G. Andrade, J. Almeida, and L. Rocha, 'GPU-NB: A Fast CUDA-Based Implementation of Naïve Bayes', in 2013 25th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp. 168–175, 2013.

[19] C. Kruengkrai and C. Jaruskulchai, 'A Parallel Learning Algorithm for Text Classification', in Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 201–206, 2002.

[20] M. AbuTair and R. Baraka, 'Design and Evaluation of a Parallel Classifier for Large-Scale Arabic Text', Int. J. Comput. Appl., vol. 75, 2013.

[21] C. Chu, S. Kim, G. Bradski, and K. Olukotun, 'Map-Reduce for Machine Learning on Multicore', in NIPS, vol. 6, pp. 281–288, 2006.

[22] L. Esmaeili, M. Akbari, V. Amiry, and S. Sharifian, 'Distributed Classification of Persian News (Case Study: Hamshahri News Dataset)', in 2013 3th International Econference on Computer and Knowledge Engineering (ICCKE), pp. 46–51, 2013.

[23] L. Zhou, H. Wang, and W. Wang, 'Parallel Implementation of Classification Algorithms Based on Cloud Computing Environment', Itkomnika Indones. J. Electr. Eng., vol. 10, no. 5, pp. 1087–1092, 2012.

[24] J. Han and M. Kamber, 'Data Mining Concepts and Techniques'. Amsterdam; Boston; San Francisco, CA: Elsevier ; Morgan Kaufmann, 2006.

[25] L. Zhou, H. Wang, and W. Wang, 'Research on Parallel Classification Algorithms for Large-scale Data', J. Converge. Inf. Technol., vol. 7, no. 21, pp. 333–340, Nov. 2012.

[26] I. Witten, E. Frank, and M. Hall, 'Data Mining: Practical Machine Learning Tools and Techniques'. Burlington, MA: Morgan Kaufmann, 2011.

[27] R. Duba, P. Hart, and D. Strok, 'Pattern Classification'. Burlington, MA: Johen Wiley & Sons, Inc., 2001.

[28] Y. Yang and G. Webb, 'Discretization for Naive-Bayes Learning: Managing Discretization Bias and Variance', Mach. Learn., vol. 74, no. 1, pp. 39–74, Jan. 2009.

[29] M. Martinez-Arroyo and L. Sucar, 'Learning an Optimal Naive Bayes Classifier', in 18th International Conference on Pattern Recognition, 2006. ICPR 2006, vol. 3, pp. 1236–1239, 2006.

[30] G. Guo, H. Wang, D. Bell, and K. Greer, 'An K-NN Model-Based Approach and its Application in Text Categorization', in Computational Linguistics and Intelligent Text Processing, Springer, pp. 559–570, 2004.

[31] A. He, A. Tan, and C. Tan, 'A Comparative Study on Chinese Text Categorization Methods,' in Proceedings of PRICAI', 2000 International Workshop on Text and Web Mining, pp. 24–35, 2000.

[32] M. Syiam, Z. Fayed, and M. Habib, 'An Intelligent System for Arabic Text Categorization', Int. J. Intell. Comput. Inf. Sci., vol. 6, no. 1, pp. 1–19, 2006.

[33] M. Aghdam, N. Ghasem-Aghaee, and M. Basiri, 'Text Feature Selection Using Ant Colony Optimization', Expert Syst. Appl., vol. 36, no. 3, Part 2, pp. 6843–6853, Apr. 2009.

[34] A. El-Halees, 'A Comparative Study on Arabic Text Classification'. Egypt. Comput. Sci. J., vol. 30, no. 2, 2008.

[35] F. Thabtah, W. Hadi, G. Al-shammare, 'VSMs with K-Nearest Neighbour to Categories Arabic Text Data'. Hong Kong: IAENG International Association of Engineers, 2008.

[36] D. Said, N. Wanas, N. Darwish, and N. Hegazy, 'A Study of Text Preprocessing Tools for Arabic Text Categorization'. in The Second International Conference on Arabic Language, pp. 230–236, 2009.

[37] C. Paice, 'An Evaluation Method for Stemming Algorithms', in Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New York, NY, USA, pp. 42–50, 1994

[38] A. Hotho, A. Nürnberger, and G. Paaß, 'A Brief Survey of Text Mining', LDV Forum - GLDV J. Comput. Linguist. Lang. Technol., 2005.

[39] J. Lovins, 'Development of a Stemming Algorithm', MIT Information Processing Group, Electronic Systems Laboratory, 1968.

[40] M. Aljlayl and O. Frieder, 'On Arabic Search: Improving the Retrieval Effectiveness Via a Light Stemming Approach', presented at the Proceedings of the eleventh International Conference on Information and Knowledge Management, pp. 340–347, 2002.

[41] R. Cummins and C. O’Riordan, 'Determining General Term Weighting Schemes for the Vector Space Model of Information Retrieval Using Genetic Programming', in 15th Artificial Intelligence and Cognitive Science Conference (AICS 2004), 2004.

[42] L. Jing, H. Huang, H. Shi, 'Improved Feature Selection Approach TFIDF in Text Mining', Proc 1st Int Conf Mach. Learn. Cybern. Beijing, 2002.

[43] J. Dean and S. Ghemawat, 'MapReduce: Simplified Data Processing on Large Clusters', Commun ACM, vol. 51, no. 1, pp. 107–113, 2008.

[44] 'What is MapReduce?' - Definition from WhatIs.com, [Online], Available: http://searchcloudcomputing.techtarget.com/definition/MapReduce. [21-Sep-2014].

[45] M. Zaharia and D. Borthakur, 'Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling', in Proceedings of the 5th European Conference on Computer systems, pp. 265–278, 2010.

[46] K. Lee, Y. Lee, H. Choi, Y. Chung, and B. Moon, 'Parallel Data Processing with MapReduce: A Survey', AcM SIGMoD Rec., vol. 40, no. 4, pp. 11–20, 2012.

[47] J. Lin and C. Dyer, 'Data-Intensive Text Processing with MapReduce', Synth. Lect. Hum. Lang. Technol., vol. 3, no. 1, pp. 1–177, 2010.

[48] P. Zhou, J. Lei, and W. Ye, 'Large-Scale Data Sets Clustering Based on MapReduce and Hadoop', J. Comput. Inf. Syst., vol. 7, no. 16, pp. 5956–5963, 2011.

[49] T. Ruiter, 'A Workload Model for MapReduce', Thesis Comput. Sci. Parallel Distrib. Syst. Group Fac. Electr. Eng. Math. Comput. Sci., vol. Delft University of Technology, Jun. 2012.

[50] C. Lam, 'Hadoop in Action', 1st edition. Greenwich, Conn: Manning Publications, 2010.

[51] 'Apache Hadoop' - Wikipedia, the free encyclopedia. [Online], Available: http://en.wikipedia.org/w/index.php?title=Apache_Hadoop&oldid=625239666. [14-Sep-2014].

[52] A. Alam and J. Ahmed, 'Hadoop Architecture and its Issues', in 2014 International Conference on Computational Science and Computational Intelligence (CSCI), vol. 2, pp. 288–291, 2014.

[53] G. Prasad, H. R. Nagesh, and M. Deepthi, 'Improving the Performance of Processing for Small Files in Hadoop: A Case Study of Weather Data Analytics', Int. J. Comput. Sci. Inf. Technol., vol. 5, no. 5, 2014.

[54] 'Apache Hadoop. Welcome to Apache Hadoop'. [Online], Available: http://hadoop.apache.org/. [23-Sep-2014].

[55] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, 'The Hadoop Distributed File System', in Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, pp. 1–10, 2010.

[56] O. Joldzic, 'Applying MapReduce Algorithm to Performance Testing in Lexical Analysis on HDFS', in Telecommunications Forum (TELFOR), 2013 21st, pp. 841–844, 2013.

[57] 'Fault tolerance for parallel MPI jobs'. [Online].Available: http://www.open-mpi.org/faq/?category=ft. [10-Nov-2014].

[58] 'Hadoop Distributed File System Architecture'. [Online], Available: http://hadoop.apache.org/docs/stable1/hdfs_design.html. [23-Sep-2014].

[59] 'Apache Mahout'. [Online], Available: http://en.wikipedia.org/wiki/Apache_Mahout. [12-Nov-2014].

[60] S. Owen, R. Anil, T. Dunning, and E. Friedman, 'Mahout in Action'. Greenwich, CT, USA: Manning Publications Co., 2011.

[61] A. Borisenko, 'Performance Evaluation in Parallel Systems', ACM Sigplan Notes, vol. 17, no. 6, pp. 150–155, 2010.

[62] M. Abd-El-Barr and H. El-Rewini, 'Fundamentals of Computer Organization and Architecture'. Hoboken, N.J.: Wiley, 2005.

[63] A. Grama, 'Introduction to Parallel Computing'. Harlow, England; New York: Addison-Wesley, 2003.

[64] 'Shamela Library', http://shamela.ws.

[65] 'Running Hadoop on Ubuntu Linux (Single-Node Cluster) - Michael G. Noll'. [Online]. Available:http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/#installation. [01-Nov-2014].

[66] 'Install Mahout in Ubuntu for Beginners | Chameera wijebandara’s Blog'. [Online]. Available:http://chameerawijebandara.wordpress.com/2014/01/03/install-mahout-in-ubuntu-for-beginners/. [01-Nov-2014].

[67] S. Perera and T. Gunarathne, 'Hadoop MapReduce Cookbook Recipes for Analyzing Large and Complex Datasets with Hadoop MapReduce'. Birmingham: Packt Pub., 2013.