The Islamic University of Gaza
Deanery of Post Graduate Studies
Faculty of Information Technology
Information Technology Department

A High Performance Parallel Classifier for Large-Scale Arabic Text

Submitted by:

Mohammed M. Abu Tair

Supervised by:

Eng. Dr. Rebhi S. Baraka
(Assistant Professor of Computer Science)

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

Palestine, Gaza
March 2013 – 1434 H

بسم الله الرحمن الرحيم

الجامعة الإسلامية – غزة
The Islamic University - Gaza

عمادة الدراسات العليا
هاتف داخلي: 1150

الرقم Ref ج س غ/35/
2013/03/24م
التاريخ Date

نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة عمادة الدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحث/ **محمد محمد أحمد أبو وطير** لنيل درجة الماجستير في **كلية *تكنولوجيا المعلومات*** **برنامج تكنولوجيا المعلومات** وموضوعها:

مُصنف متوازي عالي الأداء للنصوص العربية واسعة النطاق

A High Performance Parallel Classifier for Large-Scale Arabic Text

وبعد المناقشة التي تمت اليوم الأحد 12 جمادى الأولى 1434هـ، الموافق 2013/03/24م الساعة الثانية مساءً، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

د. ربحي سليمان بركة	مشرفاً ورئيساً	.....
د. إياد محمد الأغا	مناقشاً داخلياً	.....
أ.د. محمد أمين مكي	مناقشاً داخلياً	.....

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،،،

عميد الدراسات العليا

أ.د. فؤاد علي العاجز

P.O. Box 108, Rimal, Gaza, Palestine fax: +970 (8) 286 0800 فاكس Tel: +970 (8) 286 0700 هاتف ص.ب. 108 الرمال، غزة، فلسطين
public@iugaza.edu.ps www.iugaza.edu.ps

Abstract

Text classification has become one of the most important techniques in text mining. It is the process of classifying documents into predefined categories or classes based on their content. A number of machine learning algorithms have been introduced to deal with automatic text classification. One of the common classification algorithms is the k-Nearest Neighbor (k-NN) which is known to be one of the best classifiers applied for different languages including Arabic language and it is included in numerous experiments as a basis for comparison. Furthermore, it is a simple classification algorithm and very easy to implement since it does not require a training phase that most classification algorithms must have. However, the k-NN algorithm is of low efficiency because it requires a large amount of computational power for evaluating a measure of the similarity between a test document and every training document and for sorting the similarities. Such a drawback makes it unsuitable to handle a large volume of text documents with high dimensionality and in particular in the Arabic language.

In our research, we propose to develop a parallel classifier for large-scale Arabic text that achieves the enhanced level of speedup, scalability, and accuracy. The proposed parallel classifier is based on the sequential k-NN algorithm. We test the parallel classifier using the Open Source Arabic Corpus (OSAC) which is the largest freely public Arabic corpus of text documents. We study the performance of the parallel classifier on a multicomputer cluster that consists of 14 computers. We report both timing and classification results. These results indicate that the proposed parallel classifier has very good speedup and scalability and is capable of handling large documents collections. Also, classification results show that the proposed classifier has achieved accuracy, precision, recall, and F-measure with higher than 95%.

Keywords: *Arabic text classification, k-NN algorithm, Parallel classifier, Parallel and distributed computing, Multicomputer cluster.*

الملخص

عنوان البحث: مُصنف متوازي عالي الأداء للنصوص العربية واسعة النطاق

أصبح تصنيف النصوص أحد أهم التقنيات المستخدمة في مجال التنقيب في البيانات النصية، ويُعرَّف تصنيف النصوص على أنه عملية تصنيف المستندات إلى أصناف محددة مسبقاً بالاعتماد على محتواها، وقد قُدِّمت العديد من خوارزميات تعليم الآلة للتعامل مع تصنيف النصوص، وأحد هذه الخوارزميات هي خوارزمية (k-NN) والتي تُعرَف كأحد أفضل المُصنِّفات لمختلف اللغات بما في ذلك اللغة العربية و يتم تضمينها في تجارب عديدة كأساس للمقارنة. علاوة على ذلك، فهي تعتبر خوارزمية تصنيف بسيطة وسهلة جداً للتطبيق لكونها لا تحتاج مرحلة التعلم التي تحتاجها معظم خوارزميات التصنيف الأخرى. مع ذلك، فخوارزمية (k-NN) ذات كفاءة منخفضة لأنها تتطلب كمية كبيرة من الطاقة الحسابية لحساب التشابه بين المستند الذي سيتم تصنيفه والمستندات في عينة التدريب وكذلك في ترتيب هذه التشابهات. مثل هذا العائق يجعلها غير مناسبة لمعالجة وتصنيف الحجم الكبير من الوثائق النصية والتي تحتوي على حجم كبير من الكلمات وخصوصاً في اللغة العربية.

في هذا البحث قمنا باقتراح تطوير مُصنِّف متوازي عالي الأداء للنصوص العربية واسعة النطاق والذي يحقق المستوى المُحسَّن من الأداء والدقة. المُصنِّف المتوازي المقترَح يعتمد على خوارزمية (k-NN)، وقد تم اختبار المُصنِّف المقترَح باستخدام مجموعة البيانات النصية العربية (OSAC) والتي تعتبر أكبر مجموعة عربية من الوثائق النصية المتوفرة بشكل متاح، وتم دراسة الأداء للمُصنِّف المتوازي على أربعةَ عشرَ جهاز كمبيوتر، وقمنا بتوثيق نتائج الأداء والتصنيف. أظهرت النتائج امتلاك المُصنِّف المتوازي لأداء عالي وقدرته على معالجة وتصنيف مجموعة كبيرة من الوثائق النصية. من ناحية أخرى، أظهرت نتائج التصنيف بأن المُصنِّف حقق نتائج تصنيف عالية من حيث (accuracy, precision, recall, and F-measure) تصل إلى أعلى من ٩٥%.

الكلمات المفتاحية: تصنيف النصوص العربية، خوارزمية k-NN، المُصنِّف المتوازي، الحوسبة المتوازية والموزعة، أجهزة الحاسوب المتعددة.

Dedication

To the memory of my father

To my beloved mother

To whom I love

Acknowledgments

It is my pleasure to express my gratitude to all the people who contributed, in whatever manner, to the success of this work.

First of all, I thank Allah for giving me the strength and ability to complete this thesis.

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.

Also I would like to extend my thanks to the academic staff of the Faculty of Information Technology at the Islamic University-Gaza who helped me during my Master's study and taught me different courses.

Special thanks for Prof. Mohammad A. Mikki and Dr. Iyad M. Alagha for their valuable guide and comments.

Last but not least, I am greatly indebted to my family for their love and support.

Mohammed M. Abu Tair
March, 2013

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

API	Application Program Interface
BTO	Binary Term Occurrences
BOT	Bag Of Tokens
CA	Classical Arabic
CMP	Chip Multiprocessor
CUDA	Compute Unified Device Architecture
DA	Dialectal Arabic
DSP	Digital Signal Processing
EM	Expectation Maximization
FN	False Negative
FP	False Positive
GPU	Graphics Processing Unit
k-NN	k-Nearest Neighbor
MPI	Message Passing Interface
MSA	Modern Standard Arabic
NB	Naïve Bayes
NLP	Natural Language Processing
OSAC	Open Source Arabic Corpus
SIMD	Single Instruction Stream, Multiple Data Stream

SMP	Symmetric Multiprocessing
SVM	Support Vector Machines
TF-IDF	Term Frequency - Inverse Document Frequency
TF	Term Frequency
TN	True Negative
TO	Term Occurrences
TP	True Positive
VSM	Vector Space Model

Chapter 1 Introduction

1.1 Overview

Text mining, sometimes alternately referred to as text data mining, roughly equivalent to text analytics, refers to the process of deriving high-quality information from text. High-quality information is typically derived through the devising of patterns and trends through means such as statistical pattern learning. Text mining usually involves the process of structuring the input text (usually parsing, along with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database), deriving patterns within the structured data, and finally evaluation and interpretation of the output as shown in Figure 1.1. Text mining is well motivated, due to the fact that much of the world's data can be found in text form (newspaper articles, emails, literature, web pages, etc.) Typical text mining tasks include text classification, text clustering, concept/entity extraction, sentiment analysis, and document summarization [16, 22].

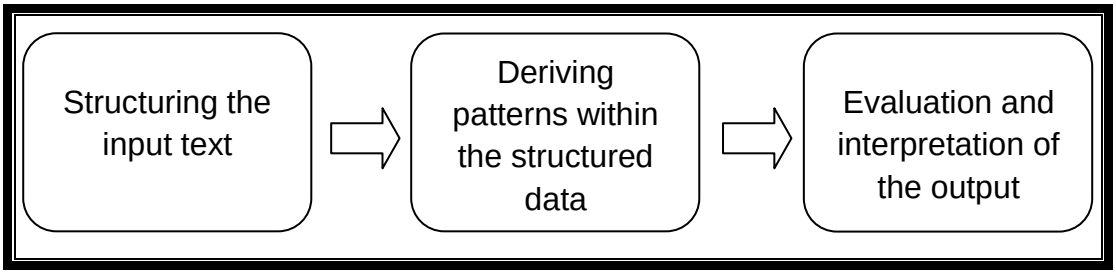

Figure 1.1: Text Mining Process.

Automatic text classification (also known as text categorization or topic spotting) is the task of assigning documents to one or more predefined categories based on their content. This task, which falls at the crossroads of information retrieval and machine learning, has witnessed a booming interest in the last years from researchers and developers alike [16, 22]. Automatic text classification has been used in many applications such as real time sorting of files into folder hierarchies, topic

identifications, automatic meta-data organization, documents' organization for databases and web pages [44, 45, 53].

The main consecutive phases of building a text classification system involve compiling and labeling text documents in corpus, selecting a set of features to represent text documents in a defined set classes or categories (structuring text data), and finally choosing a suitable classifier to be trained and tested using the compiled corpus (Figure 1.2).

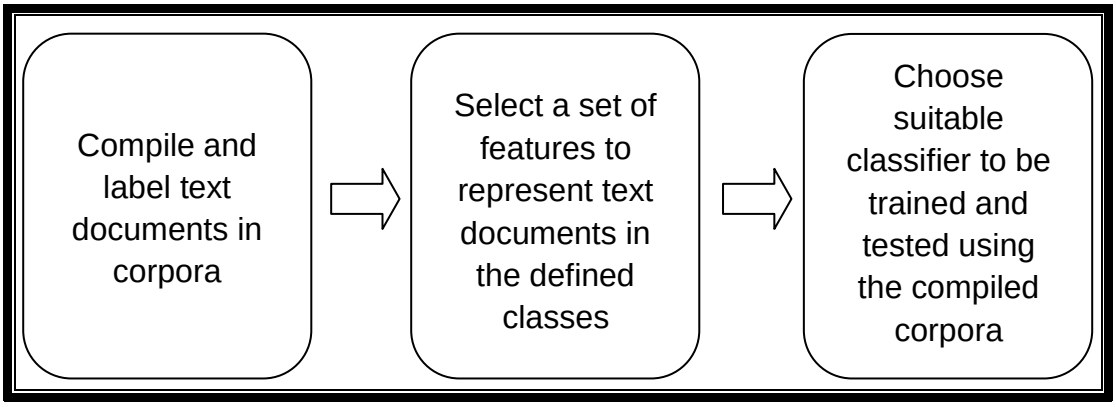

Figure 1.2: Building Text Classification System Process.

The constructed classifier system then can be used to classify new (unlabeled) text documents. It is shown in Figure 1.3.

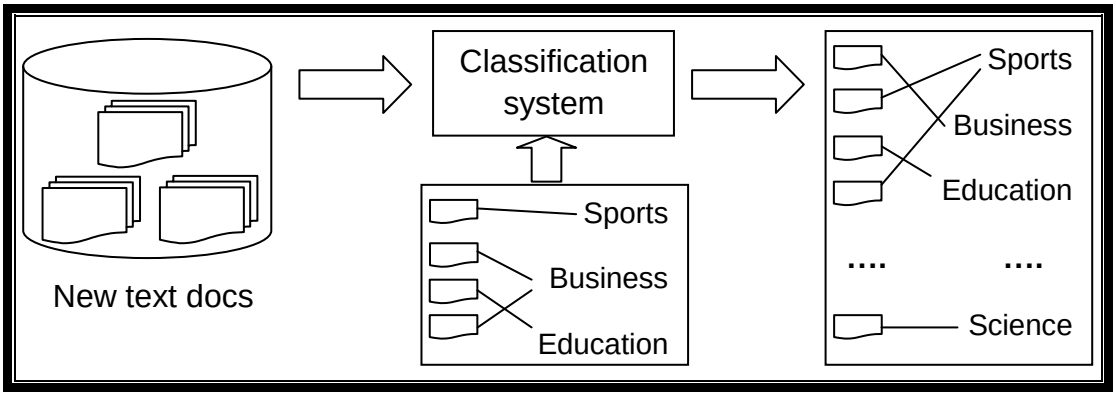

Figure 1.3: Classifying New Text Documents Using Text Classification System.

Many algorithms have been used for text classification for different languages including Arabic language such as k-NN [3, 13, 52], Naïve Bayes (NB) [13, 14, 29], Support Vector Machines (SVM) [13, 24], and Decision Tree [4, 13, 39].

Most serial text classification algorithms, like the k-NN algorithm, take a large amount of running times especially when the volume of text documents available for analysis is big. The huge amount of text documents with high dimensionality (i.e. the features or attributes and in this case they are the words that occur in documents) and in particular in the Arabic language which has a rich nature and very complex morphology requires a large amount of computational power for classification.

To be more precise, 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) [30].

The k-NN algorithm becomes a standard within the field of text classification for different languages and is included in numerous experiments as a basis for comparison. It has been in use since the early stages of text classification research, and is one of the best classifiers within the field [32, 45]. Furthermore, it is a simple classification algorithm and very easy to implement since it does not require a training phase that most classification algorithms must have. However, the k-NN algorithm is of low efficiency because it requires a large amount of computational power for evaluating a measure of the similarity between a test document and every training document and for sorting the similarities. Such a drawback makes it unsuitable to handle a large volume of text documents with high dimensionality and in particular in the Arabic language which has a rich nature and very complex morphology and for some applications where classification efficiency is crucial such as online text classification, in which the classifier has to respond to a lot of documents arriving simultaneously in stream format. Since text data rapidly increase on the Internet, the scalability of the algorithm is required to handle such massive data.

Parallel and distributed computing is an interesting technique for scaling up the algorithms. It presents a natural and promising method to deal with the problem of efficient classification in large-scale Arabic text collection.

The current trend in parallel and distributed computing is clustering. In clustering, powerful low cost workstations are linked through fast communication interfaces to achieve high performance computing. Recent increases in communication speeds, microprocessor clock speeds, and availability of message passing libraries make cluster based computing appealing in terms of both high performance computing and cost effectiveness. Parallel and distributed computing on clustered systems is a viable and attractive proposition due to the high communication speeds of modern networks [19].

The Message Passing Interface (MPI) approach is considered to be one of the most mature methods currently used in parallel programming mainly due to the relative simplicity of using the method by writing a set of library functions or an Application Program Interface (API) callable from C, or C++ Programs. MPI was designed for high performance on both massively parallel machines and clusters [31].

1.2 Problem Statement

Most serial text classification algorithms, like the k-NN algorithm, take a large amount of running times especially when the volume of text documents available for analysis is big. The huge amount of text documents with high dimensionality and in particular in the Arabic language which has a rich nature and very complex morphology require a large amount of computational power for classification.

The problem of this research is how to develop a parallel classifier for large-scale Arabic text that achieves the enhanced level of speedup, scalability, and accuracy.

1.3 Objectives

1.3.1 Main Objective

The main objective of this research is to develop a parallel classifier for large-scale Arabic text that achieves the enhanced level of speedup, scalability, and accuracy. The proposed parallel classifier is based on the sequential k-NN algorithm.

1.3.2 Specific Objectives

The specific objectives of this research are:

- Determining the largest freely public Arabic corpus of text documents with various domains.
- Investigating the most suitable text preprocessing techniques such as stemming and term pruning methods and term weighting schemes.
- Determining the most suitable data decomposition and task mapping techniques for the proposed parallel classifier.
- Designing the parallel classifier model.
- Implementing the sequential k-NN algorithm as well as the proposed parallel classifier.
- Applying the implemented sequential k-NN algorithm as well as the proposed parallel classifier on the largest freely public Arabic corpus of text documents.
- Evaluating the proposed parallel classifier using different performance metrics for parallel systems such as execution time, speedup, efficiency, and parallel overhead.
- Evaluating the obtained classification results using different classification measures such as accuracy, precision, recall, and F-measure.

1.4 Importance of the Thesis

- The proposed parallel classifier can be applied to various domains.
- This classifier is suitable for applications where the classification efficiency is crucial such as online text classification, in which the classifier has to respond to a lot of documents arriving simultaneously in stream format.

- Solve the problem of low efficiency for the sequential k-NN algorithm due to the large amount of computational power.
- The proposed parallel classifier can be used to efficiently and accurately categorize a large volume of Arabic text with high dimensionality.
- More support for the Arabic language in the technology area as our Islam encourages us to support it.

1.5 Scope and Limitations

The outcome of this research will be a parallel classifier for large-scale Arabic text that achieves the enhanced level of speedup, scalability, and accuracy. The work is applied with some limitations and assumptions such as:

- We will use a freely public Arabic corpus for text documents collection and the Non-free Arabic corpora are not considered.
- We will apply the text preprocessing techniques using the open source machine learning tool RapidMiner and the text preprocessing step will not be covered by the parallel classifier.
- The proposed parallel classifier is depend on the sequential k-NN algorithm.
- We will conduct our experiments on a set of processors and their own exclusive memory (multicomputer cluster). This platform is programmed using *send* and *receive* primitives. Libraries such as MPI provide such primitives.
- The maximum number of the used processors will be subject to the experiment.

1.6 Research Methodology

We follow a research methodology that consists of the following steps:

- **Conducting a Survey:** This include reviewing the recent literature closely related to the thesis problem statement and the research question. After analyzing the existing methods, identifying the drawbacks or the lack of existing approaches, we formulate the strategies and solutions and how to overcome the drawbacks.

- **Text Data Collection:** We will determine the largest freely public Arabic corpus of text documents with various domains.

- **Text Preprocessing:** Some preprocessing in the Arabic text corpus will be performed. It includes tokenizing strings to words, normalizing the tokenized words, applying stopwords removal, applying the suitable 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. We use the open source machine learning tool RapidMiner for text preprocessing.

- **Design the Parallel Classifier Model:** The model is a way of structuring a parallel classifier by selecting the most suitable decomposition and mapping techniques and applying the appropriate strategy to minimize interactions [19].

- **Implement the Sequential k-NN Algorithm as well as the Proposed Parallel Classifier:** We will implement the sequential k-NN algorithm using C++ programming language to serve as a baseline when we compare it with the proposed parallel classifier. We will implement the proposed parallel classifier using C++ programming language and the MPI library on a multicomputer cluster. We will apply the implemented sequential k-NN algorithm as well as the proposed parallel classifier on the largest freely public Arabic corpus of text documents.

- **Analysis and Discussion:** The proposed parallel classifier will be evaluated using different performance metrics for parallel systems such as execution time, parallel overhead, speedup, and efficiency which determines the scalability. Also, the obtained classification results will be evaluated using different classification measures such as accuracy, precision, recall, and F-measure which are generally accepted ways of measuring systems' success in this field.

1.7 Outline of the Thesis

The rest of the thesis is organized as follows: Chapter 2 is an overview of parallel computing. Chapter 3 reviews related works. Chapter 4 describes the proposed parallel Arabic text classifier. Chapter 5 presents the experiments and the results. Finally, Chapter 6 presents the conclusions and future directions.

Chapter 2 Overview of Parallel Computing

This chapter presents an overview of parallel computing; the motivation for parallel computing, types of parallel computers, methods of parallelism, interconnection schemes of parallel computing systems, software environments for parallel programming, the master-slave programming paradigm, the problems in developing parallel algorithms for distributed environment, and finally the performance metrics for parallel systems.

2.1 Definition: Parallel Computing

Parallel computing is the simultaneous execution of the same task on multiple processors in order to obtain faster results. It puts the emphasis on generating large computing power by employing multiple processing entities simultaneously for a single computation task. These multiple processing entities can be a multiprocessor system, which consists of multiple processors in a single machine connected by bus, or a multicomputer system, which consists of several independent computers interconnected by telecommunication networks or computer networks [54].

2.2 Motivation of Parallel Computing

The main purpose of doing parallel computing is to solve problems faster or to solve larger problems.

Parallel computing is widely used to reduce the computation time for complex tasks. Many industrial and scientific research and practice involve complex large-scale computation, which without parallel computers would take years and even tens of years to compute. It is more than desirable to have the results available as soon as possible, and for many applications, late results often imply useless results [54].

As predicted by Moore's Law [34], the computing capability of single processor has experienced exponential increase. This has been shown in incredible advancement in microcomputers in the last few decades. Performance of a today desktop PC costing a few hundred dollars can easily surpass that of million-dollar parallel supercomputer built in the 1960s. It might be argued that parallel computer will phase out with this increase of single chip processing capability. However, three main factors have been pushing parallel computing technology into further development.

First, although some commentators have speculated that sooner or later serial computers will meet or exceed any conceivable need for computation, this is only true for some problems. There are others where exponential increases in processing power are matched or exceeded by exponential increases in complexity as the problem size increases. There are also new problems arising to challenge the extreme computing capacity. Parallel computers are still the widely used and often only solutions to tackle these problems [54].

Second, at least with current technologies, the exponential increase in serial computer performance cannot continue for ever, because of physical limitations to the integration density of chips. In fact, the foreseeable physical limitations will be reached soon and there is already a sign of slow down in pace of single chip performance growth. Further improvement in performance will rely more on architecture innovation, including parallel processing. Intel and AMD have already incorporated multicore architectures in their latest offering [47].

Finally, generating the same computing power, single-processor machine will always be much more expensive than parallel computer. The cost of single CPU grows faster than linearly with speed. With recent technology, hardware of parallel computers are easy to build with off-the-shelf components and processors, reducing the development time and cost. It is also much easier to scale the processing power with parallel computers. Most recent technology even supports to use old computers and shared components to be part of parallel machine and further reduces the cost.

With the further decrease in development cost of parallel computing software, the only impediment to fast adoption of parallel computing will be eliminated [54].

2.3 Types of Parallel Computers

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism.

2.3.1 A Multi-Core Processor

A multi-core processor is a single computing component with two or more independent actual central processing units, called cores, which are the units that read and execute program instructions. The instructions are ordinary CPU instructions such as add, move data, and branch, but the multiple cores can run multiple instructions at the same time, increasing overall speed for programs amenable to parallel computing. Manufacturers typically integrate the cores onto a single integrated circuit die known as a Chip Multiprocessor (CMP), or onto multiple dies in a single chip package [15]. A generic dual-core processor is shown in Figure 2.1.

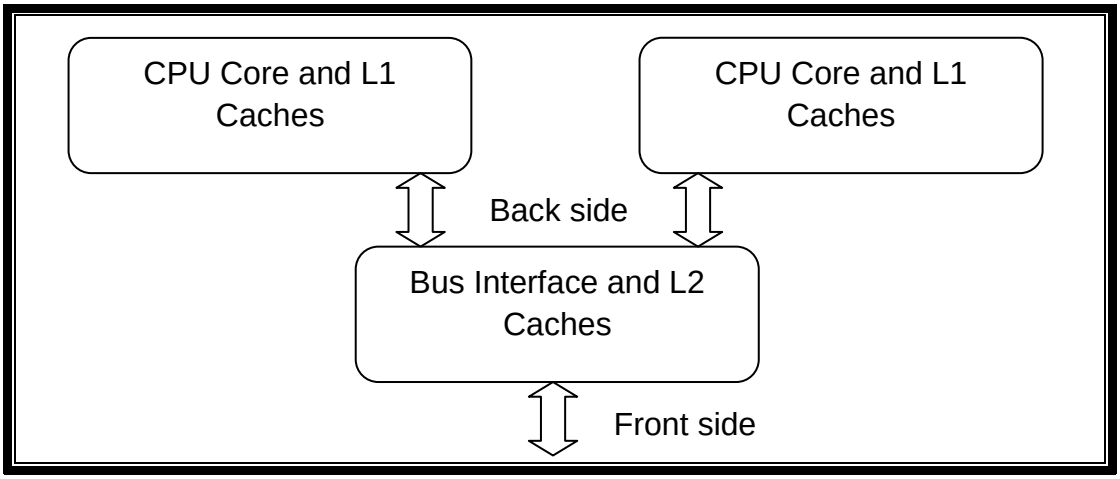

Figure 2.1: A Generic Dual-Core Processor [15].

Processors were originally developed with only one core. A dual-core processor has two cores, a quad-core processor contains four cores, a hexa-core processor contains six cores, an octa-core processor contains eight cores. A multi-core processor implements multiprocessing in a single physical package.

Multi-core processors are widely used across many application domains including general-purpose, embedded, network, Digital Signal Processing (DSP), and graphics [15].

2.3.2 A Shared Memory Multiprocessor

A shared memory multiprocessor is a computer system with multiple identical processors that share memory and connect via a bus. It involves a multiprocessor computer hardware architecture where two or more identical processors are connected to a single shared main memory and are controlled by a single OS instance. Most common multiprocessor systems today use an Symmetric Multiprocessing (SMP) architecture [8]. A shared memory multiprocessor system is shown in Figure 2.2.

A shared memory multiprocessor systems are tightly coupled systems with a pool of homogeneous processors running independently, each processor executing different programs and working on different data and with capability of sharing common resources (memory, I/O device and so on) and connected using a system bus [8].

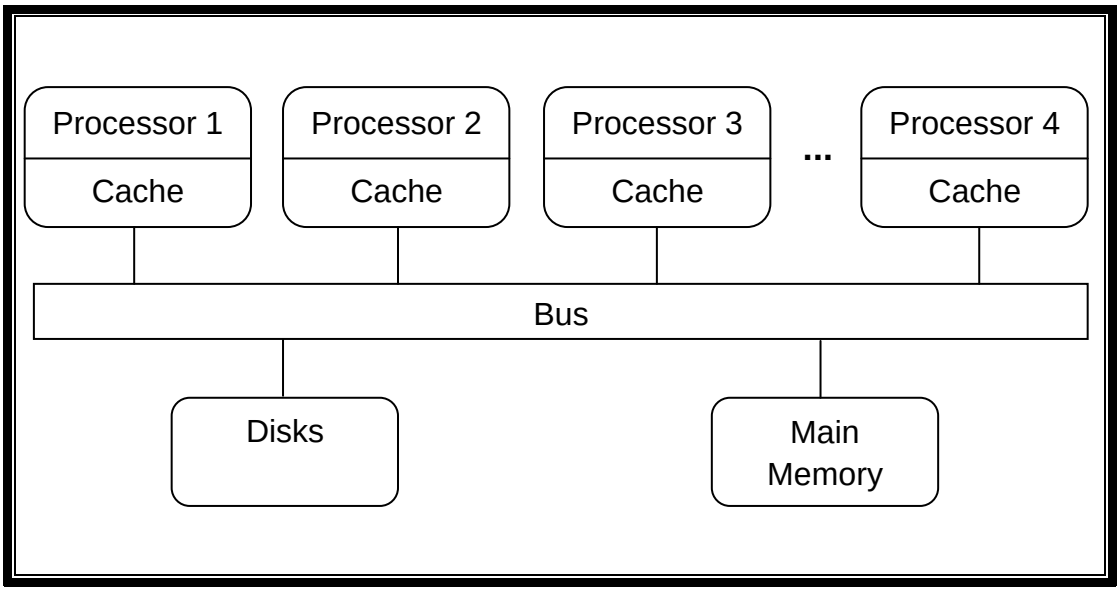

Figure 2.2: A Shared Memory Multiprocessor System [8].

2.3.3 Cluster Computing

The current trend in parallel and distributed computing is clustering. In clustering, powerful low cost workstations are linked through fast communication interfaces to achieve high performance parallel computing. Recent increases in communication speeds, microprocessor clock speeds, and availability of message passing libraries make cluster based computing appealing in terms of both high performance computing and cost effectiveness. Parallel and distributed computing on clustered systems is a viable and attractive proposition due to the high communication speeds of modern networks. Computer cluster is now the mainstream architecture of modern parallel machines [19].

The components of a cluster are usually connected to each other through fast local area networks, each computer running its own instance of an operating system. Computer clusters emerged as a result of convergence of a number of computing trends including the availability of low cost microprocessors, high speed networks, and software for high performance distributed computing [51]. The typical architecture of a cluster is shown in Figure 2.3.

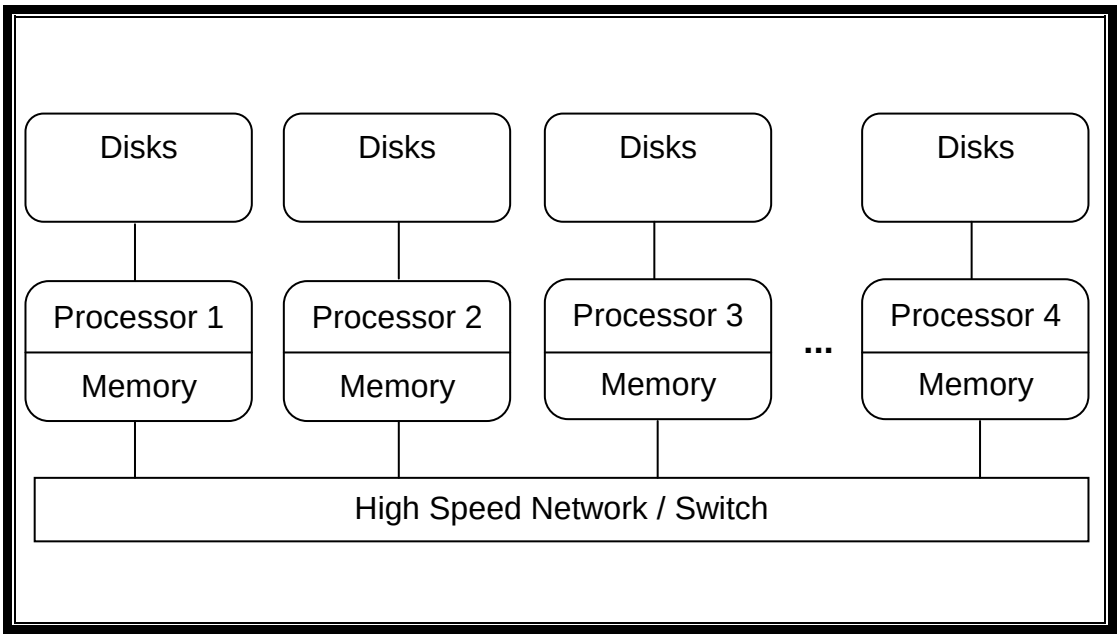

Figure 2.3: The Typical Cluster Computer Architecture [7].

Attributes of Clusters

- Computer clusters may be configured for different purposes ranging from general purpose business needs such as web-service support, to computation-intensive scientific calculations [10].
- Load-balancing clusters are configurations in which cluster-nodes share computational workload to provide better overall performance [46].
- High-availability clusters improve the availability of the cluster approach. They operate by having redundant nodes, which are then used to provide service when system components fail. High-availability cluster implementations attempt to use redundancy of cluster components to eliminate single points of failure [46].

Our research mainly rely on a computer cluster for computing power.

2.4 Methods of Parallelism

For compute-intensive applications, parallelization is an obvious means for improving performance and achieving scalability. A variety of techniques may be used to distribute the workload involved in data mining over multiple processors. Two major classes of parallel implementations are distinguished; task parallelism and data parallelism.

With task parallelism each processor has or needs access to the entire database and multiple operations are executed concurrently. With data parallelism the database is portioned among the processors and the same operation is executed in multiple partitions at the same time. From a data mining viewpoint, data parallelism has several main advantages over task parallelism. A lot of previously written serial code can be reused in a data parallel fashion. This simplifies programming and leads to a development time significantly smaller than one associated with task parallel programming. In most applications, the amount of data can increase arbitrarily fast, while the number of lines of code typically increases at a much slower rate. To put it in simple terms, the more the data is available, the more opportunity to exploit data

parallelism [19]. Figure 2.4 shows that the dataset itself can be shared (in shared memory architecture), partially or totally replicated, or portioned among the available nodes (in distributed memory architecture).

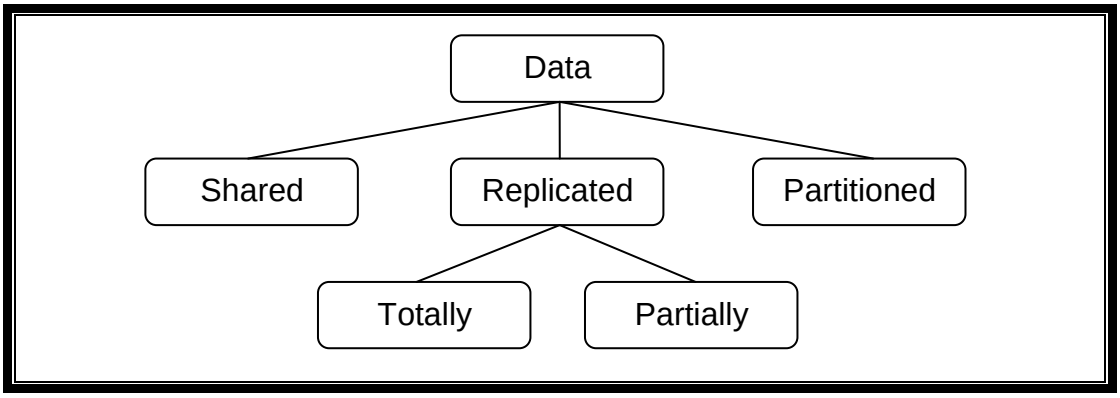

Figure 2.4: Data Partitioning Methods.

Data partitioning comes in two flavors. A partitioning based on records will assign non-overlapping sets of records to each of the processors. Alternatively, a partitioning of attributes will assign sets of attributes to each of the processors [19].

2.5 Interconnection Schemes of Parallel Computing Systems

High-performance parallel computers, especially those able to scale to thousands of processors, have been using sophisticated interconnection schemes. Here we cover the major interconnection schemes listed in Figure 2.5 in brief.

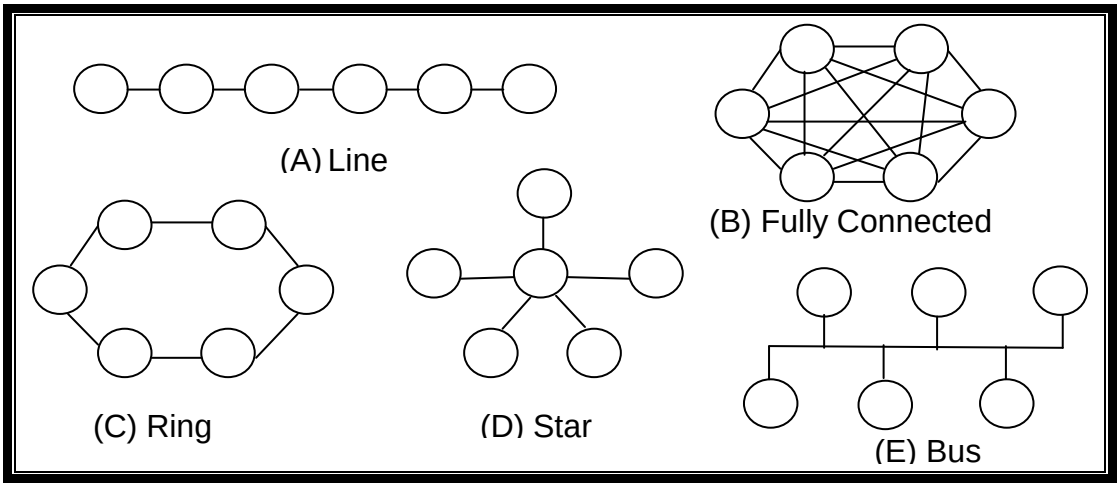

Figure 2.5: Illustrations of Simple Interconnection Schemes [54].

Figure 2.5(A) illustrates the line scheme, which is the simplest connection scheme. In this illustration, circle represents a computing node and line represents direct communication channel between nodes. Computing nodes are arranged on and connected with a single line. However, communication between any two non-neighbor nodes needs the help of other nodes; the fault at any node will make the whole system break. This scheme is simple and low-cost, but will not be able to generate high performance or reliability; and as system scales, the performance degrades rapidly [19]. Figure 2.5(C) illustrates the ring scheme, which is an enhanced line topology, with an extra connection between the two ends of the line. However, basic characteristics are still the same [19].

The other extreme is probably the fully-connected topology, in which there is a direct connection between any two computing nodes. Fully-connected topology is shown in Figure 2.5(B). The corresponding graph representation has an edge between any two vertices, and distance between any two vertices is 1 , and it generates the minimal communication latency. This scheme will generate the highest performance possible, but due to the complexity and thus cost, it can hardly be scalable with larger scale, although performance will not degrade at all [19].

Similar to fully-connected network, bus network, illustrated in Figure 2.5(E), has direct connection between any two nodes. In fact, bus topology shares the same logical graph representation with fully-connected topology. The connection between any pair of nodes is not dedicated but shared: interconnection is implemented via a shared bus. This reduces the complexity significantly. This single shared bus prevents more than one pair of nodes to carry out point-to-point communication. As a result, the system does not scale very well [19].

An intuitive improvement on bus network is to change the bus to eliminate the constraint that only two nodes can communicate at any time. The result is the star network, where a communication switch node is added to replace the shared bus, as shown in Figure 2.5(D) [19].

For computer clusters, most are built with a star structured interconnection network around a central switch.

There are other types of more sophisticated topology schemes, such as tree, mesh, and hypercube, which are widely used in parallel computers with thousands of processors or more. These schemes often scale better to larger scale network with good performance. Readers are advised to [9] for more information about this.

2.6 Software Environments for Parallel Programming

Parallel programming is a complex task. In order to reduce this complexity, different programming models are abstracted, with each providing tools such as special-purpose compilers, libraries and frameworks to simplify programming task. These tools hide many details about parallel execution, such as message transfer and routing, task allocation and migration, and platform differences. Higher-level programming model will even have commonly-used algorithms pre-implemented in the bounded libraries [35]. Our research is based on message passing programming model and specifically on MPI standards and MPICH library.

2.6.1 Message Passing Interface and MPICH

MPI, is the most widely-used message passing standard. The basic functions are defined by the MPI standard [33], and with many implementations targeting almost all distributed memory architectures, it is the de facto industrial standard for message passing programming. There are two main standards that make up MPI, MPI-1 and MPI-2. Most basic functionality is provided in the MPI-1 standard, with more advanced features defined in the MPI-2 standard. One of the key objectives of the MPI standard is to provide portability between different parallel machines. Therefore, MPI defines its own data types which are used for data transfers which are then mapped to specific machine data types by the MPI library implementation, which should ensure that programs do not have to be rewritten to use different computing hardware [31].

Basically, MPI provides two types of communication operations. Point-to-point operations which allow any two processes to exchange information via *MPI_Send* (for sending), *MPI_Recv* (for receiving) and their variants. Collective operations are

provided so that a set of processes, known as a communicator, can share and dispatch data through broadcast and reduction operations [31].

When an MPI program runs, the user will explicitly specify the number of parallel processes and how the processes are mapped to physical processors. On startup, each processor starts one or more processes to execute the same program body. Each parallel process will be assigned a rank, which serves as the identity of the process, and which will also cause processes to carry out different computation despite their common program body. During the execution, processes carry their own computation, without synchronization with other processes unless they encounter an explicit synchronization command. Processes communicate with each other using point-to-point or collective communication primitives, using process rank to address the recipient or sender if it is required. The whole parallel program exits when all the parallel processes have finished. Although there is no requirement on how the computation result is generated, in many cases a head process, usually the one with rank 0, will collect the results from participating processes and assemble the final outcome [31].

The two major implementations of MPI standards are MPICH [20] developed by Argonne National Laboratory and LAM by Ohio Supercomputing Center and Indiana University. Our research is heavily based on MPICH2. MPICH2 is a new implementation of MPI by the MPICH team. In addition to features of its predecessor, including the portability advantage, MPICH2 includes partial implementation of MPI-2 functions, including one-side communication, dynamic process creation, and expand MPI-IO functionality [31].

2.7 Master-Slave Programming Paradigm

The master-slave paradigm is the main programming paradigm used in parallel programs. Master-slave approach is used for task that can be partitioned into several independent subtasks, which can be carried out separately and probably (but not necessarily) in parallel without any inter-subtask communication [19].

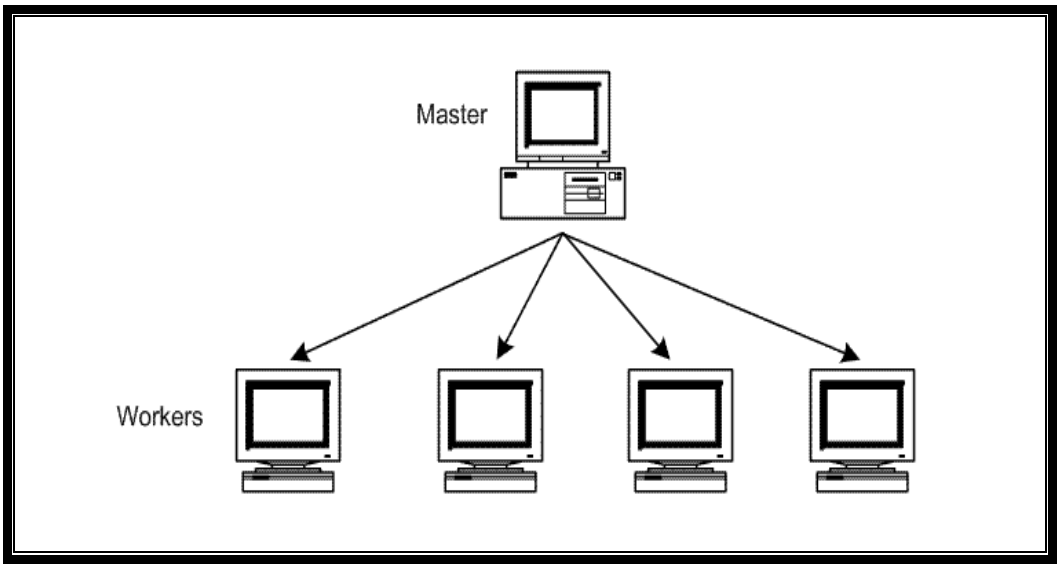

Figure 2.6: Master-Slave Paradigm [54].

The master-slave paradigm is depicted in Figure 2.6. The master node, usually denoted as node 0, is in charge of farming out work load to workers. Several workers work on workloads assigned by the master node. When a worker finished its current work load, it reports the result back to the master if necessary and triggers the master to send additional work load to the worker. As long as the task can be partitioned into sufficiently small segments, this approach will produce small amounts of idle time for the worker nodes [19].

The master-slave paradigm is very robust to program. All tasks control is done by one processor, the master. The user should not be burdened with the difficult issue of how to distribute algorithm control information to the various processors. Moreover, the typical parallel programming hurdles of load balancing circumvented. Having a central point of control facilitates the collection of a job's statistics. Furthermore, a surprising number of sequential approaches to large-scale problems can be mapped naturally to the master-worker paradigm [17].

Programs with centralized control are easily able to adapt to a dynamic and heterogeneous computing environment. If additional processors become available during the course of the computation, they simply become workers and are given portions of the computation to perform. Having centralized control also

eases the burden of adapting to a heterogeneous environment, since only the master need be concerned with the matchmaking process of assigning tasks to resources making the best use of the resource characteristics [17].

Our research is based on the master-slave programming paradigm in which the master processor generates the work and allocates it to the worker processors.

2.8 Problems in Developing Parallel Algorithms for Distributed Environment

There are several problems in developing parallel algorithms for a distributed environment with data mining which is being considered in this research work. These are [19]:

- **Data Distribution:** One of the benefits of parallel and distributed data mining is that each node can potentially work with a reduced-size subset of the total database. A parallel algorithm in distributed environment must effectively distribute data to allow each node to make independent progress with its incomplete view of the entire database.

- **I/O Minimization:** Even with good data distribution, parallel data mining algorithms must strive to minimize the amount of I/O they perform to the database.

- **Load Balancing:** To maximize the efficiency of parallelism, each workstation must have approximately the same amount of work to do. Although a good initial data distribution can help provide load-balancing.

- **Avoiding Duplication:** Ideally, no workstation should do redundant work (work already performed by another node).

- **Minimizing Communication:** An ideal parallel data mining algorithm allows all workstations to operate asynchronously, without having to stall frequently for global barriers or for communication delays.

- **Maximizing Locality:** As in all performance programming, high-performance parallel data mining algorithms must be designed to reap the full performance potential of hardware. This involves maximizing locality for good cache behavior, utilizing as much of the machine's memory bandwidth as possible, etc.

Achieving all of the above goals in one algorithm is nearly impossible, as there are tradeoffs between several of the above points. Existing algorithms for parallel data mining attempt to achieve an optimal balance between these factors.

2.9 Performance Metrics for Parallel Systems

In order to demonstrate the effectiveness of parallel processing for a problem on some platform, several concepts have been defined. These concepts will be used in later chapters to evaluate the effectiveness of parallel programs. These include serial runtime, parallel runtime, parallel overhead, speedup, and efficiency.

2.9.1 Serial Runtime

The serial runtime of a program is the time elapsed between the beginning and the end of its execution on a sequential computer. The serial runtime is denoted by T_S [19].

2.9.2 Parallel Runtime

The parallel runtime is the time that elapses from the moment the first processor starts to the moment the last processor finishes execution. The parallel runtime is denoted by T_P [19].

2.9.3 Total Parallel Overhead

The parallel overhead is the total time spent by all processors combined in non useful work [19]. The overhead function (T_o) is given by:

$$T_o = (p\ T_P - T_S) / T_S \qquad (2.1)$$

where p is the number of processors, T_S is the serial runtime, and T_P is the parallel runtime.

2.9.4 Speedup

The speedup is the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with p identical processing elements [19]. This is shown as:

$$S = T_S / T_P \qquad (2.2)$$

where S is the speedup achieved with p processors, T_S is the serial runtime, and T_P is the parallel runtime.

A typical speedup curve for a fixed size problem is shown in Figure 2.7. As the number of processors increases, speedup also increases until a saturation point is reached. Beyond this point, adding more processors will not bring further performance gain. This is the combined result of reduced computation on participating node, and increased duplicate computation and synchronization and communication overhead [19].

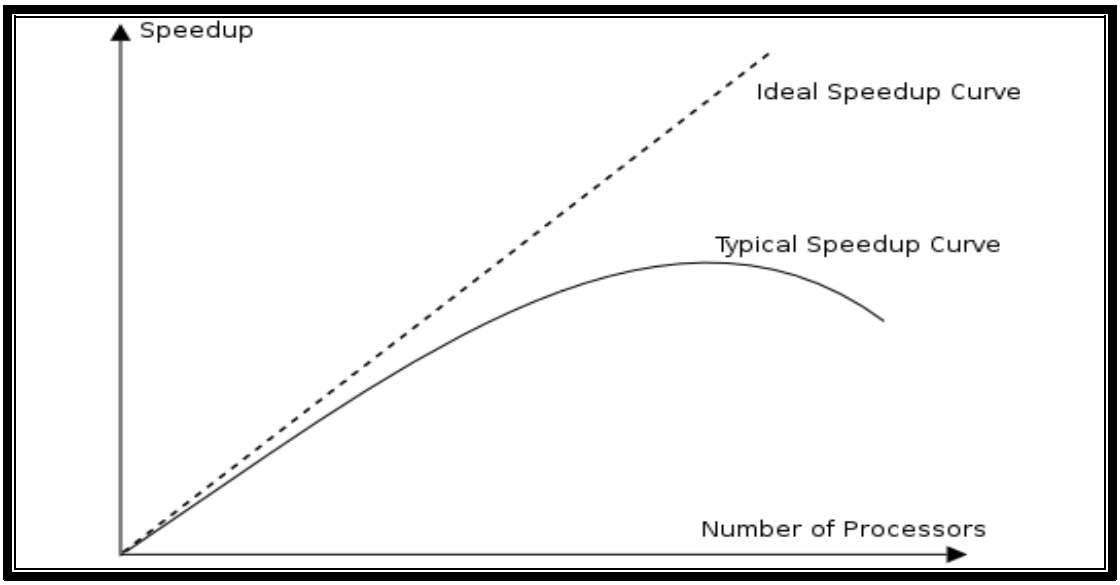

Figure 2.7: Typical speedup curve [54].

2.9.5 Efficiency

The efficiency is a measure of the fraction of time for which a processing element is usefully employed [19]. It is given by:

$$E = S / p \qquad (2.3)$$

where *E is* the efficiency, *S* is the speedup achieved with *p* processors, and *p* is the number of processors.

It measures how much speedup is brought per additional processor. Based on the typical speedup curve shown in Figure 2.7, it is evident that typically efficiency will be decreased upon increase in the number of processors. Efficiency can be as low as 0 and as high as 1 [19].

2.9.6 Scalability

The concept of scalability cannot be computed but evaluated. A parallel system is said to be scalable when the efficiency can be kept constant as the number of processing elements is increased, provided that the problem size is increased [19].

2.10 Summary

In this chapter, we presented an overview of parallel computing; the main purpose of doing parallel computing, the classification of parallel computers according to the level at which the hardware supports parallelism, the techniques that used to distribute the workload in parallel programs, the major interconnection schemes of parallel computing systems, the message passing programming model for parallel programs, the master-slave paradigm which is the main programming paradigm used in parallel programs, the problems in developing parallel algorithms for distributed environment, and finally we described the performance metrics for parallel systems that evaluate the effectiveness of parallel programs.

A review of existing works closely related to the thesis will be discussed in the next chapter.

Chapter 3 Related Works

This chapter presents a review of existing works closely related to the thesis and identifies the drawbacks of existing approaches.

In order to improve the efficiency of sequential classification algorithms for text classification, some researches have been conducted in this area and they can be classified into three categories:

3.1 Enhancing the Efficiency of Sequential Classification Algorithms with Feature Selection, Reduction and Pruning Strategies

Al-Shalabi et. al [3], applied k-NN on Arabic text, they used Term Frequency Inverse Document Frequency (TF-IDF) as a weighting scheme for feature selection and got accuracy of 95%. They also applied stemming as feature reduction technique. They collected a corpus from newspapers (Al-Jazeera, An-Nahar, Al-Hayat, Al-Ahram, and Ad-Dostor) and from Arabic Agriculture Organization website. The corpus consists of 627 documents belonging to one of six categories (politics 111, economic 179, sport 96, health and medicine 114, health and cancer 27, and agriculture 100). They preprocessed the corpus by applying stop words removal and light stemming. The feature selection and reduction strategies can decrease the computation complexity, reduce the dimensionality, and improve the accuracy rate of classification. However, 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 text documents with high number of features and in particular in the Arabic language which has a rich nature and very complex morphology.

Duwairi et. al [11], compared three dimensionality reduction techniques; stemming, light stemming, and word cluster. Stemming reduces words to their stems. Light stemming removes common affixes from words without reducing them to their stems. Word clusters group synonymous words into clusters and each cluster is represented by a single word. The purpose of employing the previous methods is to reduce the size of documents vectors without affecting the accuracy of the classifiers. They used k-NN to perform the comparison. The comparison metric includes size of documents vectors, classification time, and accuracy (in terms of precision and recall). They used Term Frequency (TF) as a weighting scheme for feature selection. They collected 15,000 documents belonging to one of three categories (sport, economic, education). Each category has 5,000 documents. They split the corpus; 9,000 documents for training and 6,000 documents for testing. Several experiments were carried out using four different representations of the same corpus: the first version uses stem-vectors, the second uses light stem-vectors, the third uses word clusters, and the fourth uses the original words (without any transformation) as representatives of documents. In terms of vector sizes and classification time, the stemmed vectors consumed the smallest size and the least time necessary to classify a testing dataset that consists of 6,000 documents. The light stemmed vectors superseded the other three representations in terms of classification accuracy. The feature selection and reduction strategies 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 and in particular in the Arabic language which has a rich nature and very complex morphology. Also, this approach reduces the features but what is the solution in the case of large volume of text documents which increase the computation complexity.

Guan and Zhou [18], proposed a training-corpus pruning based approach to speedup the k-NN algorithm. It depends on the removal of the noisy and superfluous documents in training corpuses, which leads to substantial classification efficiency improvement. They used clustering-based feature selection method that treating each training class as a distinctive cluster, then using a genetic algorithm to select a subset

of documents features. They used Apte corpus; the number of documents sample is 5773 in ten categories, 2447 documents prepared for testing. The pruning strategy can reduce the size of training corpus significantly, decrease the computation complexity, but it can damage the classification quality of k-NN for text classification, any removal of training documents may aggravate the sparseness of the text corpus, which leads to a degradation of the k-NN classifier.

3.2 Enhancing the Efficiency of Sequential Classification Algorithms by Combination with Other Algorithms

Buana et. al [6], proposed a method that combines traditional k-NN algorithm and k-Means clustering algorithm. They used TF-IDF as the weighting scheme for feature selection. They group all the training samples of each category by k-Means algorithm, and take all the cluster centers as the new training samples, the modified training samples are used for classification with the k-NN algorithm. The results show that the combination of the proposed algorithm in this study has a percentage accuracy reached 87%, an average value of f-measure evaluation= 0.8029 with the best k-values= 5 and the computation takes 55 second for one document. Buana collected corpus from news website www.detik.com and www.kompas.com. The number of documents sample is 802 with 5915 terms and 6 categories that are, General News, Business Economics, Education and Science, Health, Sports, and Technology. 60 documents prepared for testing, each category of 10 documents.

Tan [48], proposed a binary k-NN for text classification. He employed the information gain as the feature selection method. They combine the centroid-based classifier with the k-NN classifier. He compute a centroid vectors to represent the documents of each class. For each test document, he first select some neighboring classes as candidate categories by calculating the similarity between the test document and centroid vectors; he then use the k-NN decision rule to find the most similar category among the candidate categories. The results show that the binary k-NN takes much less CPU time, without loss of classification accuracy. Tan used two English corpora: the 20Newsgroup and the OHMUSED. The 20Newsgroup

dataset contains approximately 20,000 documents and The OHMUSED dataset contains approximately 11,162 documents in ten categories.

The combination of traditional k-NN algorithm and clustering algorithm can reduce the time complexity of traditional k-NN algorithm. However, The clustering algorithm can take a large amount of time for clustering the training samples especially in the case of the large volume of text documents.

3.3 Enhancing the Efficiency of Sequential Classification Algorithms with Parallel Computing

Ruoming et. al [37], proposed a parallel learning algorithm. The parallel algorithm is based on the k-NN algorithm. They evaluated the parallel implementation on a multiprocessor with shared memory that connect multiple processors to a single memory system. Each training sample is processed by one processor. After processing the sample, the processor determines if the list of k current nearest neighbors should be updated to include this sample. They used a full-replication scheme to avoid the race conditions. They experimented with a 800 MB main memory resident dataset. The reduction object in this algorithm's parallel implementation is the list of k-nearest neighbors, the value of k used in their experiments was 2000. The speedup results were suitable up to four processors. However, sharing memory in this way can easily lead to a performance bottleneck and the scalability of the processors is limited.

Lianga et. al [30], proposed a parallel learning algorithm. The parallel algorithm is based on the k-NN algorithm. They evaluated the parallel implementation on Compute Unified Device Architecture (CUDA) enabled Graphics Processing Unit (GPU). The advantage of this method is the highly parallelizable architecture of the GPU. Recent development in GPUs has enabled inexpensive high performance computing for general-purpose applications. Due to GPU's tremendous computing capability, it has emerged as the co-processor of the Central Processing Unit (CPU) to achieve a high overall throughput. CUDA programming model provides the programmers adequate C language like APIs to better exploit the parallel power of the GPU and manipulate it. At the hardware level, CUDA-enabled GPU is a set of

Single Instruction Stream, Multiple Data Stream (SIMD) processors with 8 stream processors. They used synthetic data generated by MATLAB for the purpose of evaluation where the number of data objects is 262144 records. Their experiment showed good scalability on data objects. CUK-NN presented up to 15.2 speedup. The result shows that CUk-NN is suitable for large scale dataset. However, since SIMD processors are specially designed, they tend to be expensive and have long design cycles and the scalability of the processors is limited.

Zufrin [55], proposed a parallel decision tree, it is a distributed-memory, data-parallel algorithm, it splits the training records horizontally in equal-sized blocks, among the processors. It follows a master-slave paradigm, where the master builds the tree, and finds the best split points. The slaves are responsible for sending class frequency statistics to the master. For categorical attributes, each processor gathers local class frequencies, and forwards them to the master. For numeric attributes, each processor sorts the local values, finds class frequencies for split points, and exchanges these with all other slaves. Each slave can then calculate the best local split point, which is sent to the master, who then selects the best global split point. This work supports our approach in terms of using multicomputer cluster which is a viable and attractive method due to the high communication speeds of modern networks.

Tekiner et. al [49], proposed a parallel learning algorithm for part of speech tagging. The parallel algorithm is based on the Maximum Entropy algorithm. They used Genia which is a sequential POS tagger as a baseline for comparison. Genia is built with maximum entropy and it is specifically tuned for biomedical text. They implemented a parallel version of genia tagger application and performance has been compared. The focus has been particularly on scalability of the application. Scaling up to 96 processors has been achieved and a hundred thousand abstracts have been processed in less than 5 minutes, whereas serial processing would take around 8 hours. The parallel implementation of genia tagger is done using MPI library. They used two datasets; the first dataset is Medline which is a collection of Medline abstracts contain around 1.7 billion words, another dataset contains 1 Million abstracts. This work supports our approach in terms of using multicomputer cluster

which is a viable and attractive method due to the high communication speeds of modern networks.

Kruengkrai and Jaruskulchai [27], proposed 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 augment 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. The experimental results on the efficiency indicate that the parallel algorithm has good speed up characteristics when the problem sizes are scaled up. They used the 20 Newsgroups data set. It contains approximately 20,000 documents. This work supports our approach in terms of using multicomputer cluster which is a viable and attractive method due to the high communication speeds of modern networks.

3.4 Summary

In this chapter, we presented a review of existing works closely related to the thesis and identifies the drawbacks of existing approaches, we classified the methods to improve the efficiency of sequential classification algorithms into three categories: The first category includes using the feature selection, reduction and pruning strategies that decrease the computation complexity, reduce the dimensionality, and improve the accuracy rate of classification. However, the size of the used corpora is small and this strategies could not do well in the case of reducing computation complexity for a large volume of text documents with high number of features and in particular in the Arabic language which has a rich nature and very complex morphology. The pruning strategy can reduce the size of training corpus significantly, decrease the computation complexity, but it can damage the classification quality of k-NN for text classification. The second category includes combination with other algorithms such as clustering algorithm that reduce the time

complexity of traditional k-NN algorithm. However, The clustering algorithm can take a large amount of time for clustering the training samples especially in the case of the large volume of text documents. The third category includes using the parallel computing to improve the efficiency of sequential k-NN algorithm, their platform comprises a multiprocessors with shared memory that connect multiple processors to a single memory system. However, sharing memory in this way can easily lead to a performance bottleneck and the scalability of the processors is limited.

In this research, we intend to develop a parallel classifier for large-scale Arabic text that achieves the enhanced level of speedup, scalability, and accuracy. The proposed parallel classifier is based on the sequential k-NN algorithm. Our platform comprises a set of processors and their own exclusive memory (multicomputer cluster) which is a viable and attractive method due to the high communication speeds of modern networks, this platform is programmed using *send* and *receive* primitives, libraries such MPI provide such primitives.

In the next chapter, we will present the sequential k-NN algorithm and describe the text data collection and preprocessing stages.

Chapter 4 The Sequential k-NN Algorithm and Text Preprocessing

In this chapter we present the sequential k-NN algorithm which is the base of the proposed parallel classifier and describe the text data collection and preprocessing stages. Text preprocessing is the important stage in text classification and it includes many steps including feature reduction using morphological analysis techniques, and term weighting.

4.1 The Sequential k-NN Algorithm

The k-NN algorithm [21]: was first described in the early 1950. 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 of the 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=(x1,x2,…,xn)* and *Y=(y1,y2,…,yn)* is:

$$d(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \qquad (4.1)$$

Figure 4.1 shows that the test sample (circle) should be classified either to the first class of squares or to the second class of triangles. If k = 3 it is classified to the second class because there are 2 triangles and only 1 square inside the inner circle. If k = 5 it is classified to first class (3 squares vs. 2 triangles inside the outer circle).

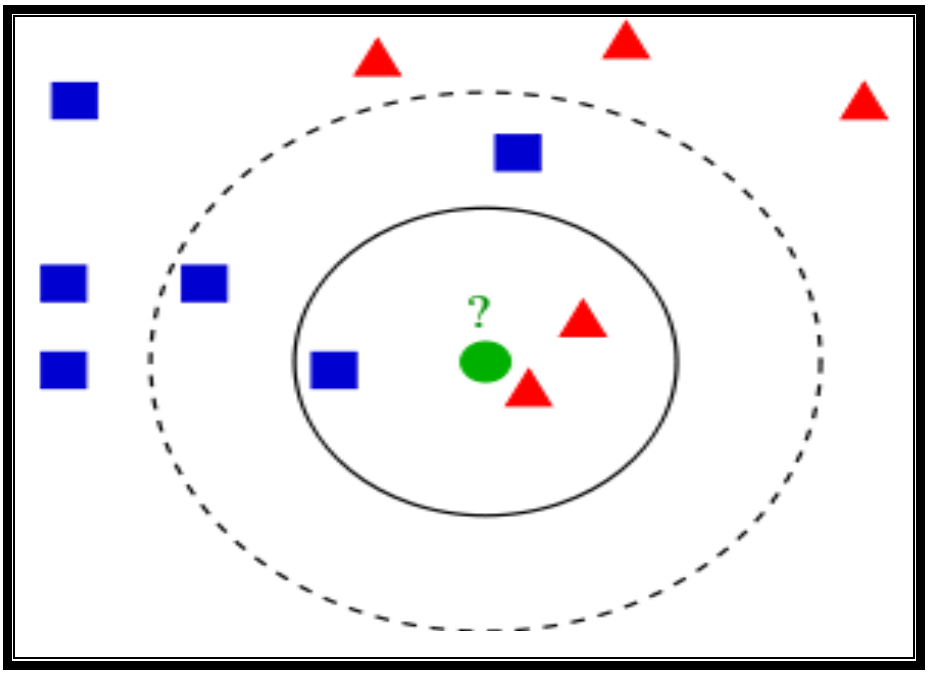

Figure 4.1: Example of k-NN Classification [26].

The sequential k-NN algorithm is briefly described as follows [21]:

- Determine parameter k = number of nearest neighbors.
- Calculate the distance between the query-instance and all the training samples.
- Sort the distance and determine nearest neighbors based on the k-th minimum distance.
- Gather the category of the nearest neighbors.
- Use simple majority of the category of nearest neighbors as the prediction value of the query instance.

The time complexity of the k-NN algorithm is *O(nm)*, where *n* is the number of tested samples to classify and *m* is the number of training samples in the training set, because for each unknown sample the similarity with each training sample is calculated. On the other hand, the space complexity of the k-NN algorithm is *O(m)*, because the whole training set is stored.

The pseudo code of the sequential k-NN algorithm is shown in Algorithm 4.1.

Algorithm 4.1: The k-NN Algorithm [36].

	Input: Training set D = {(***x1*** , ***y1***), . . . , (***xn***, ***yn***)}.
	x′ new instance to be classified.
	Output: predicted class label ***y′*** for ***x′***.
	ALGORITHM
1	FOR each labeled instance (***xi***, ***yi***) calculate d(***xi*** , ***x′***) from equation (4.1)
2	Order d(***xi*** , ***x′***) from lowest to highest, (i = 1, . . . , n).
3	Select the *k* nearest instances to ***x′***: D***x′***.
4	Output ***y′*** that is the most frequent class in D***x′***.

4.2 Text Data Collection and Preprocessing

4.2.1 Text Data Collection

We use the largest freely public Arabic corpus of text documents which is called OSAC from [38] to perform our experimentations. The OSAC corpus is available publically at [41].

4.2.2 Text Preprocessing

Arabic Language is the 5th widely used language in the world. It is spoken by more than 422 million people as a first language and by 250 million as a second language. Arabic has 3 forms; Classical Arabic (CA), Modern Standard Arabic (MSA), and Dialectal Arabic (DA). CA includes classical historical liturgical text, MSA includes news media and formal speech, and DA includes predominantly spoken vernaculars and has no written standards. Arabic alphabet consists of the following 28 letters (أ، ب، ت، ث، ج، ح، خ، د، ذ، ر، ز، س، ش، ص، ض، ط، ظ، ع، غ، ف، ق، ك، ل، م، ن، ه، و، ي). The orientation of writing in Arabic is from right to left [5].

One of widely used methods for text mining presentations is viewing text as a Bag Of Tokens (BOT) (words, n-grams). Under that model we can already classify text. These are quite useful for mining and managing large volumes of text. However, there is a potential to do much more. The BOT approach loses a lot of information contained in text, such as word order, sentence structure, and context. These are precisely the features that humans use to interpret text. Natural Language Processing (NLP) attempts to understand document completely (at the level of a human reader). General NLP has proven to be too difficult. The reason that NLP in general is so difficult is that text is highly ambiguous. Natural Language is meant for human consumption and often contains ambiguities under the assumption that humans will be able to develop context and interpret the intended meaning [1, 2, 3, 23].

Some preprocessing in the OSAC corpus is performed. It includes tokenizing string to words, normalizing the tokenized words, applying stopwords removal, applying the suitable 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. We use the open source machine learning tool RapidMiner for text preprocessing. See Appendix B for more information.

1. String Tokenization

It is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens. The list of tokens becomes input for further processing such text mining [50].

2. Stopwords Removal

Stopwords are terms that are too frequent in the text. These terms are insignificant. So, removing them reduces the space of the items significantly. There is no definite list of stop words which all NLP tools incorporate. Not all NLP tools use a stoplist. Some tools specifically avoid using them to support phrase searching [16, 22].

Typically, a default list of English stop words includes "the", "a", "of", "since", etc., i.e., words that are used in the respective language very frequently, but communicate very little unique information about the contents of the document.

For Arabic, stopwords list includes punctuations (?, !, …), pronouns (…، همـا، التـي، الـذي، هـي، هـو), adverbs (... بـين، تحـت، فـوق), days of week (...، الاثنين، الأحد، السبت), month of year (... مـارس، فبرايـر، ينـاير). Stopwords list are removed because they do not help determining document topic and to reduce features.

3. Morphological Analysis Techniques (Stemming and Light Stemming)

In linguistics, morphology is the identification, analysis and description of the structure of morphemes and other units of meaning in a language like words, affixes, and parts of speech [16, 22].

For Arabic Language, there are two different morphological analysis techniques; stemming and light stemming.

Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form – generally a written word form. Stemming would reduce the Arabic words (المكتبـة، الكاتـب، الكتـاب) which mean (the library), (the writer), and (the book) respectively, to one stem (كتب), which means (write) [16, 22].

Stemming algorithm by Khoja [25] is one of the well known Arabic stemmers. Khoja‘s stemmer removes the longest suffix and the longest prefix. It then matches the remaining word with verbal and noun patterns, to extract the root. The stemmer makes use of several linguistic data files such as a list of all diacritic characters, punctuation characters, definite article, and stopwords. The steps of Khoja Arabic stemmer is described in Algorithm 4.2.

Light stemming, in contrast, removes common affixes from words without reducing them to their stems.

The main idea for using light stemming is that many word variants do not have similar meanings or semantics. However, these word variants are generated from the

same root. Thus, root extraction algorithms affect the meanings of words. Light stemming aims to enhance the classification performance while retaining the words meanings. It removes some defined prefixes and suffixes from the word instead of extracting the original root [11, 12].

Algorithm 4.2: Arabic Stemming Algorithm Steps [25].

1	Remove diacritics.
2	Remove stopwords, punctuation, and numbers.
3	Remove definite article (ال).
4	Remove inseparable conjunction (و).
5	Remove suffixes.
6	Remove prefixes.
7	Match result against a list of patterns. - If a match is found, extract the characters in the pattern representing the root. - Match the extracted root against a list known "valid" roots.
8	Replace weak letters واي with و.
9	Replace all occurrences of *Hamza* ئ ء ؤ with ا.
10	Two letter roots are checked to see if they should contain a double character. If so, the character is added to the root.

Formally speaking, the aforementioned Arabic words (المكتبة، الكاتب، الكتاب) which mean (the library), (the writer), and (the book) respectively, belong to one stem (كتب) despite they have different meanings. Thus, the stemming approach reduces their semantics. The light stemming approach, on the other hand, maps the word (الكتاب) which means (the book) to (كتاب) which means (book), and stems the word (الكاتب) which means (the writer) to (كاتب) which means (writer). Another example

for light stemming is the words (المسافرون، المسافرين) which mapped to word (مسافر). Light stemming keeps the words' meanings unaffected. Algorithm 4.3 shows the steps of Arabic light stemming. A light stemmer [28] is a standard Arabic light stemmer.

Algorithm 4.3: Arabic Light Stemming Algorithm Steps [28].

1	Normalize word: - Remove diacritics. - Replace آ أ إ with ا. - Replace ة with ه. - Replace ى with ي.
2	Stem prefixes: - Remove prefixes: ال، وال، بال، كال، فال، لل، و.
3	Stem suffixes: - Remove suffixes: ها، ان، ات، ون، ين، ية، ه، ي.

4. Term Pruning

It is the process of eliminating the words that its count is less or greater than a specific threshold [40].

5. Vector Space Model (VSM) and Term Weighting Schemes

The aim of term weighting schemes is to enhance text document representation as feature vector. Popular term weighting schemes are:

- **Binary Term Occurrences (BTO):** which indicates absence or presence of a word with Booleans 0 or 1 respectively.
- **Term Frequency (TF):** the ratio between the number of occurrences of term *t* in the document *d* and the number of all terms in the document *d*.

- **Term Occurrences (TO) :** the number of occurrences of term *t* in the document *d*.

- **Term Frequency-Inverse Document Frequency (TF-IDF):** the TF-IDF is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. Term frequency *tf(t, d)* is the number that the term *t* occurred in the document *d*. Document frequency *df(t)* is number of documents in which the term *t* occurs at least once [16, 22, 23, 42, 43]. The inverse document frequency can be calculated from document frequency using the formula:

log(num of Docs/num of Docs with word i) *(4.2)*

A reasonable measure of term importance may then be obtained by using the product of the term frequency and the inverse document frequency *(tf * idf)*.

4.3 Summary

In this chapter, we presented and described the sequential k-NN algorithm which is the base for the proposed parallel classifier, and described the text data collection and preprocessing stages. Text preprocessing is the important stage in text classification and it includes many steps including feature reduction using morphological analysis techniques, and term weighting.

In the next chapter, we will provide a detailed description about the proposed parallel classifier.

Chapter 5 The Proposed Parallel Classifier

In this chapter we present and describe the proposed parallel classifier model including the decomposition and mapping techniques, the steps of the proposed parallel classifier and the appropriate strategies to minimize overheads. The proposed parallel classifier will be described using algorithms and flowcharts.

The parallel classifier model is a way of structuring a parallel classifier by selecting the most suitable decomposition and mapping techniques and applying the appropriate strategies to minimize overheads.

5.1 Decomposition Technique

The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently by identifying the data on which computations are performed, then partition this data across various tasks.

The task performs the computations with its part of the data. In our classifier, the input training data partitioning is the natural decomposition technique because the output (the computed distances) is not clearly known a-priori. It divides the data set equally according to the number of worker processors by sending a one data partition for each of them. See section 2.4 for more information.

Figure 5.1 shows a decomposition based on a partitioning of the input training data. Each of the two tasks computes the distances of the new test document in its respective subset of training data. The two sets of distances, which are the independent outputs of the two tasks, represent intermediate results. Combining the intermediate results yields the final result.

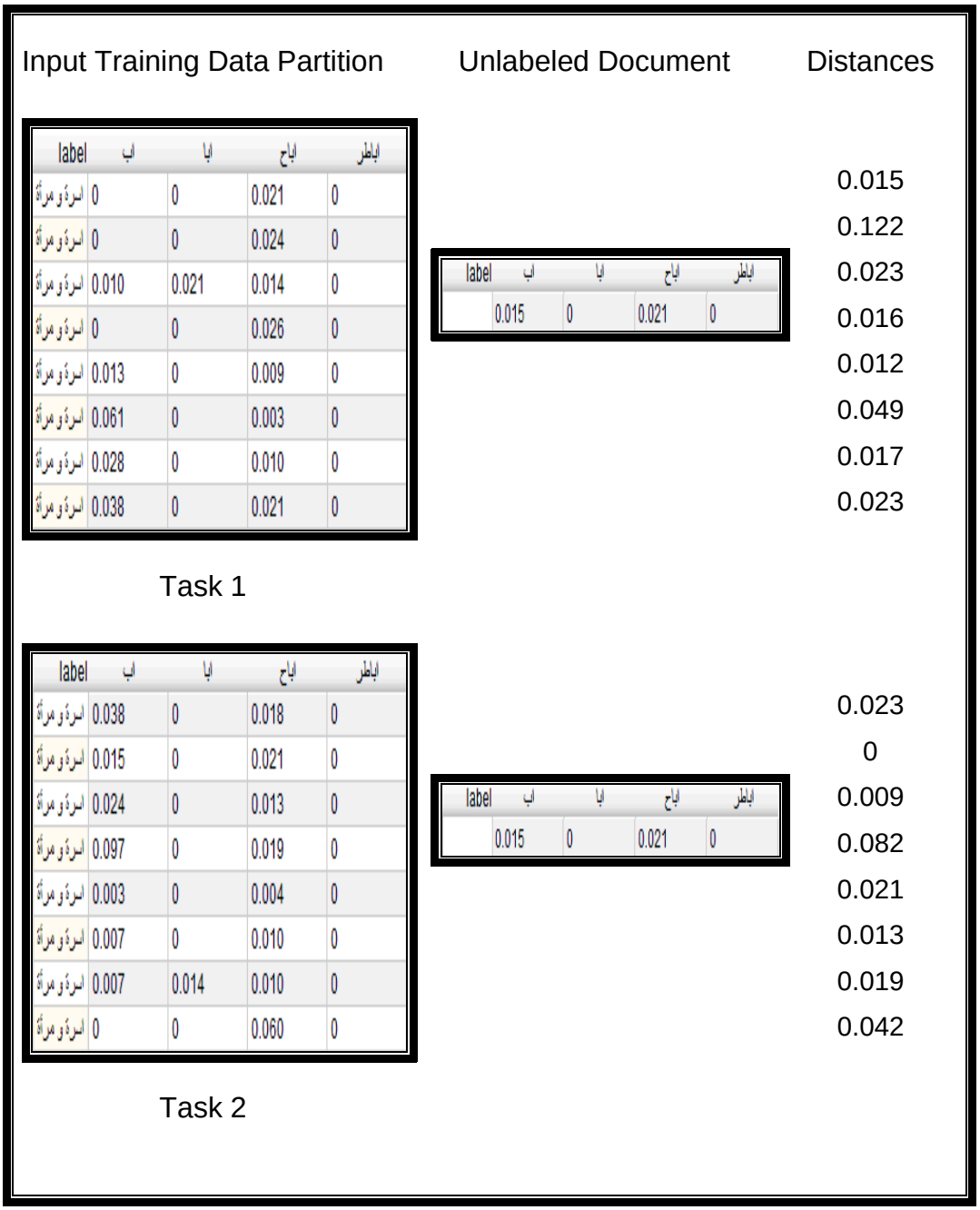

Figure 5.1: Partitioning the Training Data Among the Processors.

5.2 Mapping Technique

Once a problem has been decomposed into concurrent tasks, these must be mapped to processors (that can be executed on a parallel platform).

In our classifier, we use the static mapping technique that distribute the tasks among processes prior to the execution of the program.

The scheme for this static mapping is mapping based on data partitioning because our data represented in a two-dimensional array. So, the most suitable scheme used for distributing the two-dimensional array among processes is the row-wise 1-D block array distribution that distribute the array and assign uniform contiguous portions of the array to different processes. See section 2.4 for more information.

According to the previous selected decomposition and mapping techniques, the suitable parallel model is the master-slave model in which the master processor divides the input training data equally according to the number of worker processors and sending a one data partition for each of them with the new document to be classified. See section 2.7 for more information.

Since the most time consuming in the k-NN algorithm taken by the calculation of the distance between the query-instance and all the training samples, and the sorting of the distances to determine nearest neighbors based on the k-th minimum distance. Our classifier takes into consideration these two factors by partitioning the work of distances computation and sorting among several worker processors.

The time complexity of the proposed parallel classifier is $O(nm/p)$, where n is the number of tested documents to classify and m is the number of training documents in the training set, because each processor calculates the similarities between each sample and its m/p training documents. On the other hand, the space complexity of the proposed parallel classifier is $O(m/p)$, because the whole training set is scattered among the p processors. Thus, the proposed parallel classifier has space scalability.

The pseudo code of the proposed parallel classifier is shown in Algorithm 5.1.

Algorithm 5.1: The Proposed Parallel Classifier.

	Input: Training set D = {(***x1*** , ***y1***), . . . , (***xn***, ***yn***)}.
	x′ new document to be classified.
	Output: predicted class label ***y′*** for ***x′***.
	ALGORITHM
1	The master processor divides D equally among worker processors and sends a one partition for each of them.
2	While True:
a	If processor = master:
i	Load ***x′***.
ii	Send ***x′*** to the worker processors.
viii	Receive D***x′*** from the worker processors and put the combined k-th ordered lists in LD***x′***.
ix	Order LD***x′*** from lowest to highest.
x	Output ***y′*** that is the most frequent class in LD***x′***.
b	Else:
iii	Receive ***x′*** from the master processor.
iv	FOR each labeled instance (***xi***, ***yi***) calculate d(***xi*** , ***x′***) from equation (4.1).
v	Order d(***xi*** , ***x′***) from lowest to highest, (i = 1, . . . , n).
vi	Select the *k* nearest instances to ***x′***: D***x′***.
vii	Send D***x′*** to the master processor.

Figure 5.2 exhibits the flow chart of the proposed parallel classifier.

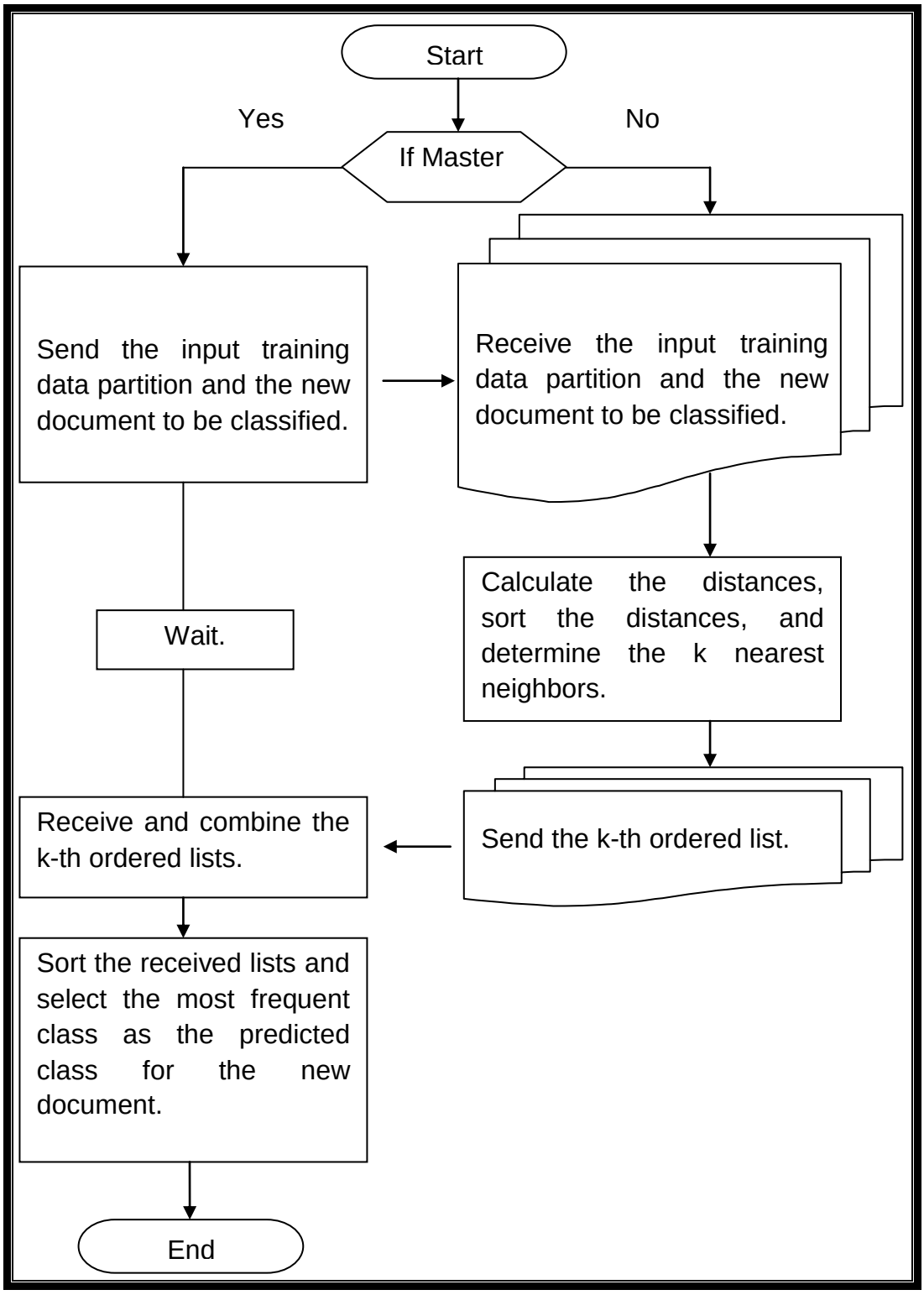

Figure 5.2: The Flow Chart of the Proposed Parallel Classifier.

As we see from Figure 5.2, The master-slave paradigm is the programming paradigm used in this parallel program. The master processor divides the input training data equally according to the number of worker processors and sending a one data partition for each of them with the new document to be classified. Each worker processor receives its data partition and the new document to be classified, calculates the distance between the new document and all the training samples, sorts the distances, determines the nearest neighbors based on the k-th minimum distance locally, and sends the k-th ordered list to the master which include the k-th distances and classes. The master processor receives from each worker the k-th ordered list and combining them in a k-th master list. Finally, the master processor sorts the k-th master list elements in ascending order, selects the k-th top elements, and selects the most frequent class in the k-th top elements as the predicted class for the new document.

5.3 Applying the Appropriate Strategies to Minimize Overheads

We apply several strategies to minimize overheads in our proposed parallel classifier. These are:

- **Load Balancing:** To maximize the efficiency of parallelism, each processor have approximately the same amount of input training data to do. This good initial data distribution can help provide load-balancing.

- **Avoiding Duplication:** In our parallel classifier, no processor do redundant work performed by another processor.

- **Using The Master-Slave Programming Paradigm:** The master-slave paradigm is the main programming paradigm used in our parallel classifier that allow the subtasks to carried out separately in parallel without any inter-subtask communication and this approach will produce small amounts of idle time for the worker processors.

- **Overlapping Computations with Interactions:** The amount of time that the master processor spend waiting for results to arrive from worker processors can be reduced, by doing some useful computations during this waiting time.

In our parallel classifier we keeping 1 input training data partition for local processing by the master processor.

5.4 Summary

In this chapter, we presented and described our proposed parallel classifier using the algorithms and flowcharts, and described the parallel classifier model including the decomposition and mapping techniques, the steps of the proposed parallel classifier and the appropriate strategies to minimize overheads.

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

Chapter 6 Experimental Results and Evaluation

This chapter discusses the experimental results to provide evidence that our parallel classifier design can improve both the computational efficiency and the quality of classification. The chapter includes three sections: Section 6.1 presents the corpus used in our experimentation and gives insight into the main characteristics of it. Section 6.2 describes the experimental environment. Finally, in Section 6.3, we present and discuss the experimental results and make a comparison with related approaches.

6.1 The Corpus

We use the OSAC corpus which is the largest freely public Arabic corpus of text documents to perform our experimentations.

The OSAC Arabic corpus collected from multiple websites as presented in Table 6.1, 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 and 449,600 district keywords after stopwords removal.

We generate all text representations for OSAC corpus to evaluate the obtained classification results using different classification measures such as accuracy, precision, recall, and F-measure which are generally accepted ways of measuring systems' success in this field. See Appendix B for more information.

Table 6.1: The OSAC Corpus.

Category	Number of text documents	Sources
Economic	3102	bbcarabic.com – cnnarabic.com – aljazeera.net- khaleej.com – banquecentrale.gov.sy
History	3233	www.hkam.net – تاريخ الحكام moqatel.com – altareekh.com – التاريخ تاريخ الإسلام islamichistory.net
Education and family	3608	saaid.net – صيد الفوائد نصائح للسعادة الأسرية almurabbi.com – المربي naseh.net
Religious and Fatwas	3171	CCA corpus – EASC corpus – moqatel.com – islamic-fatwa.com – شبكة الفتاوى الشرعية صيد الفوائد saaid.net
Sport	2419	bbcarabic.com – cnnarabic.com – khaleej.com
Health	2296	dr-ashraf.com – العيادة الالكترونية CCA corpus – kids.jo – صحة الطفل EASC corpus – W corpus – العلاج البديل العربي arabaltmed.com
Astronomy	557	arabstronomy.com – الفلك العربي الكون نت alkawn.net – بوابة الفلك المغربية bawabatalfalak.com – الفلك - موسوعة النابلسي nabulsi.com – www.alkoon.alnomrosi.net
Low	944	lawoflibya.com – القانون الليبي قانون كوم qnoun.com
Stories	726	kids.jo – قصص الأطفال CCA corpus – صيد الفوائد said.net
Cooking Recipes	2372	aklaat.com – fatafeat.com
Total	22,428	

The generated text representations for OSAC corpus are:

- Light stemming + percentual term pruning (min threshold = 3%, max threshold = 30%) + TF-IDF.
- Light stemming + percentual term pruning (min threshold = 3%, max threshold = 30%) + TF.
- Light stemming + percentual term pruning (min threshold = 3%, max threshold = 30%) + TO.
- Light stemming + percentual term pruning (min threshold = 3%, max threshold = 30%) + BTO.
- Stemming + percentual term pruning (min threshold = 3%, max threshold = 30%) + TF-IDF.
- Stemming + percentual term pruning (min threshold = 3%, max threshold = 30%) + TF.
- Stemming + percentual term pruning (min threshold = 3%, max threshold = 30%) + TO.
- Stemming + percentual term pruning (min threshold = 3%, max threshold = 30%) + BTO.

We have described these text representations in more details in section 4.2.

6.2 Experimental Setup

This section describes the experimental environment for evaluating our proposed approach.

We implemented the sequential k-NN algorithm using C++ programming language to serve as a baseline when we compare it with the proposed parallel classifier to give a fair comparison. We implemented the proposed parallel classifier using C++ programming language and the MPI library on a multicomputer cluster. See Appendix A for more information.

The target platform for our experiments is a cluster of computers and their own exclusive memory connected through local area network with speed 10/100 Mbps. The cluster consists of 14 node, all nodes have the same specifications; Intel(R) Core(TM) i3-2120 CPU @ 3.30 GHz, 4.00 GB RAM, 320 GB hard disk drive. The sequential k-NN algorithm and the proposed parallel classifier have been implemented on Windows 7 operating system, and we have used the parallel message passing software MPICHI2 that offers small latencies and high bandwidths.

6.3 Experimental Results and Discussion

This section summarizes and discusses the results of the numerous experiments that have been conducted.

6.3.1 Discussion of the Parallel Classifier Results

We used the largest text representation for OSAC corpus which is (Light stemming + percentual term pruning (min threshold = 3%, max threshold = 30%) + TF-IDF), (22,428 documents that are represented as records and 2114 words that are represented as attributes) to evaluate the proposed parallel classifier using different performance metrics for parallel systems such as execution time, parallel overhead, speedup, and efficiency which determines the scalability.

For evaluation purposes, we split the largest generated text representation for OSAC corpus into two parts; 50% of the corpus for training (11214 documents) and the remaining 50% for testing (11214 documents) using stratified sampling which keep class distributions remains the same after splitting. Then we convert these text data parts into two text files with .txt format in order to read it by the classifier. We used the open source machine learning tool RapidMiner for this purpose. We splitting the corpus in this way to achieve higher classification results and to evaluate the performance of the parallel classifier.

We have executed the parallel classifier varying the number of processors from 2 to 14, also we varied the number of tested documents to observe the effects of different problem sizes on the performance. Three sets were used with the number of tested documents 2803, 5607, and 11214 documents.

The execution time in seconds is recorded in Table 6.2.

Table 6.2: The Execution Time of the Sequential and Parallel Classifiers.

Problem Size / No. of Processors		2803 Documents	5607 Documents	11214 Documents
Sequential k-NN		870.97	1755.41	3586.70
Parallel Classifier	2-Processors	484.07	960.99	1914.20
	4-Processors	256.75	510.75	997.95
	6-Processors	176.53	344.50	679.53
	8-Processors	148.94	288.69	566.34
	10-Processors	132.44	252.34	496.38
	12-Processors	117.49	222.18	435.61
	14-Processors	107.25	204.22	398.64

As we note from Table 6.2, the sequential version 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 sequential one. Figure 6.1 shows the curves of execution time for the classifiers on the OSAC corpus. The time curve decreases from 1 processor until using 14 processors.

Several observations can be made by analyzing the results in Figure 6.1. First, the sequential k-NN algorithm spent a lot of time classifying the text documents. Second, the proposed parallel classifier clearly reduce the sequential time. Notice that the sequential k-NN algorithm takes about 1 hour to classify this collection, while the proposed parallel classifier reduces this time to 6 minutes on 14 processors.

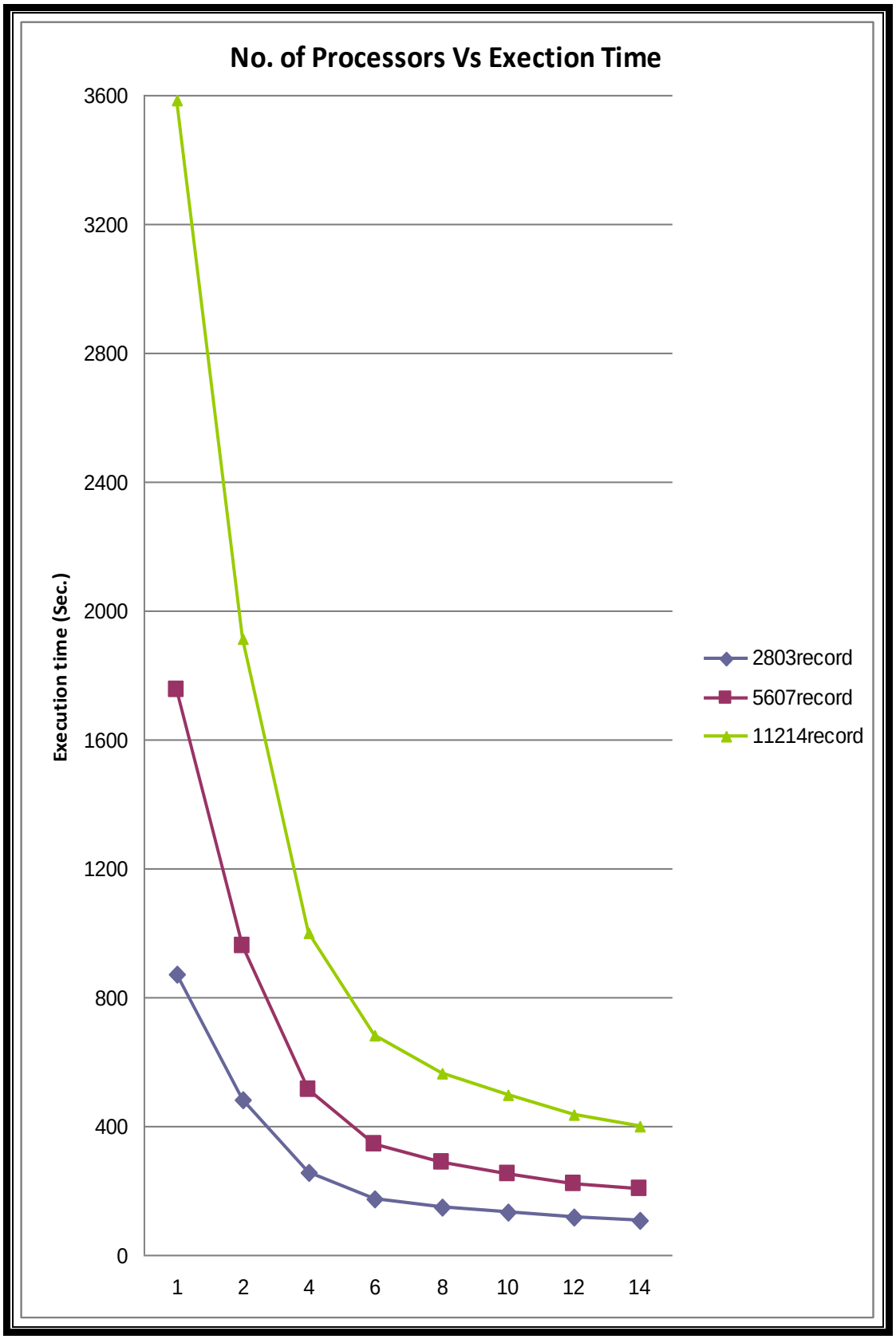

Figure 6.1: The Curves of Execution Time for the Two Classifiers.

Also, we compute the speedup which gained from this parallelization. The speedup is recorded in Table 6.3. Figure 6.2 demonstrates the relative speedup.

Table 6.3: The Relative Speedup of the Proposed Parallel Classifier.

Problem Size / No. of Processors	2803 Documents	5607 Documents	11214 Documents
2-Processors	1.80	1.83	1.87
4-Processors	3.39	3.44	3.59
6-Processors	4.93	5.10	5.28
8-Processors	5.85	6.08	6.33
10-Processors	6.58	6.96	7.23
12-Processors	7.41	7.90	8.23
14-Processors	8.12	8.60	9.00

The speedup curves increase linearly in some cases. For example, on the largest tested set (11214 documents), it achieves the relative speedups of 1.87, 3.59, 6.33, and 9.00 on 2, 4, 8, and 14 processors, respectively. When it accesses to a smaller set of tested documents, the speedup curves tend to drop from the linear curve. The classifier achieves the relative speedups of 1.83, 3.44, 6.08, and 8.60 on 2, 4, 8, and 14 processors, respectively. The smallest tested documents sizes give the same trend. If we increase the number of processors further, the speedup curves tend to significantly drop from the linear curve. For a given problem instant, the relative speedups saturates as the number of processors is increased due to increased overheads. This is a normal situation when the problem size is fixed as the number of processors increases. However, it can be solved by scaling the problem size. For example, in Figure 6.2, the speedups for three sets on 4 processors improve from 3.39 to 3.59, on 8 processors improve from 5.85 to 6.33, and on 14 processors

improve from 8.12 to 9.00. It can be seen that our parallel classifier yields better performance for the larger data sets.

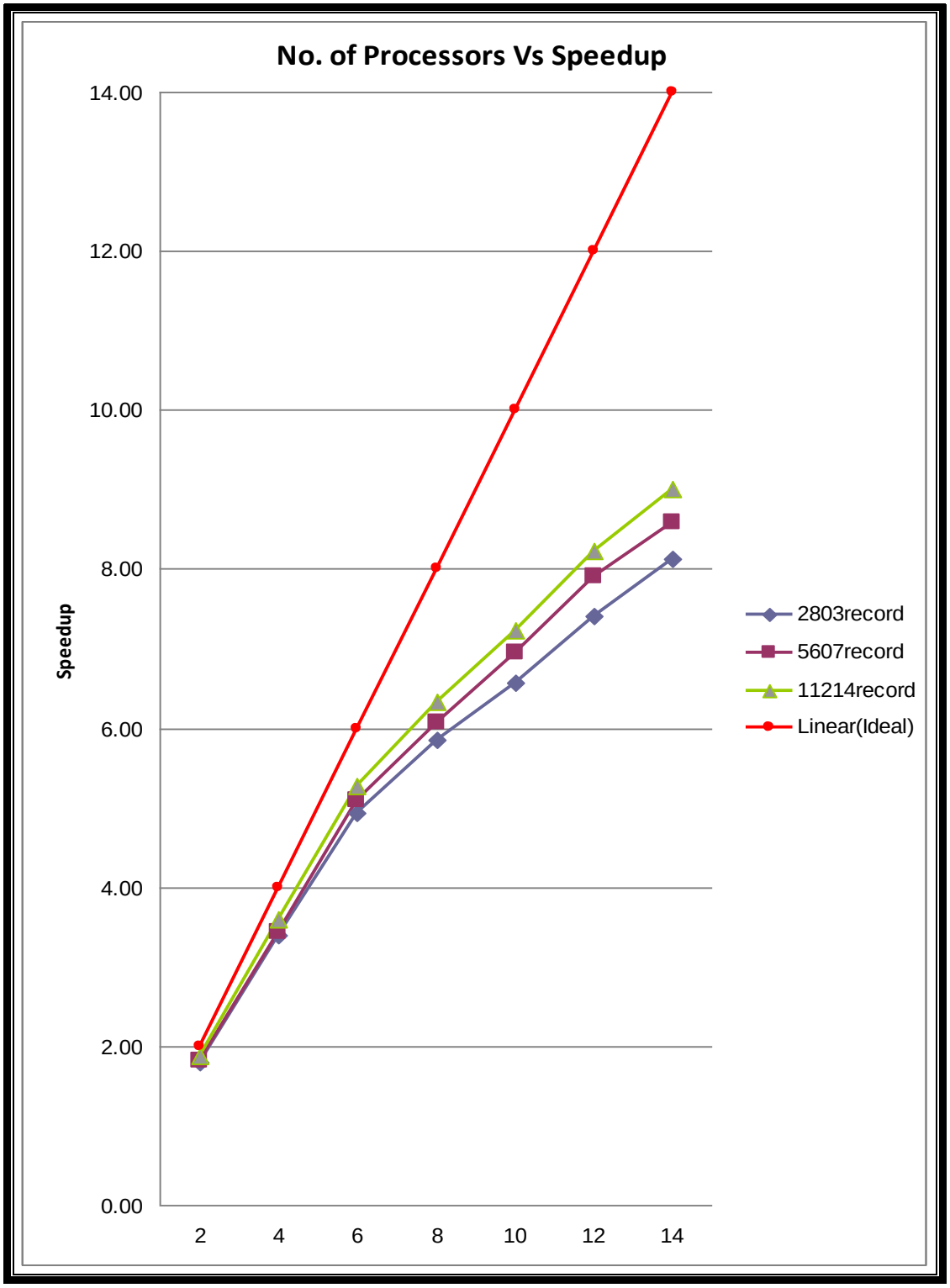

Figure 6.2: The Relative Speedup Curves of the Proposed Parallel Classifier.

From the speedup we can compute the efficiency. The efficiency values are recorded in Table 6.4. Figure 6.3 illustrates the efficiency curves.

Table 6.4: The Efficiency of the Proposed Parallel Classifier.

Problem Size / No. of Processors	2803 Documents	5607 Documents	11214 Documents
2-Processors	0.90	0.91	0.94
4-Processors	0.85	0.86	0.90
6-Processors	0.82	0.85	0.88
8-Processors	0.73	0.76	0.79
10-Processors	0.66	0.70	0.72
12-Processors	0.62	0.66	0.69
14-Processors	0.58	0.61	0.64

As we note from Table 6.4, The value of efficiency is between zero and one. We note that the efficiency decrease as the number of processing elements is increased for a given problem size and this is common to all parallel programs due to increased overheads.

Also, we note that the efficiency of the parallel classifier increases if the problem size is increased (from 2803 documents to 11214 documents) while keeping the number of processing elements constant.

It can be seen that our parallel classifier 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 2803 documents to 11214 documents).

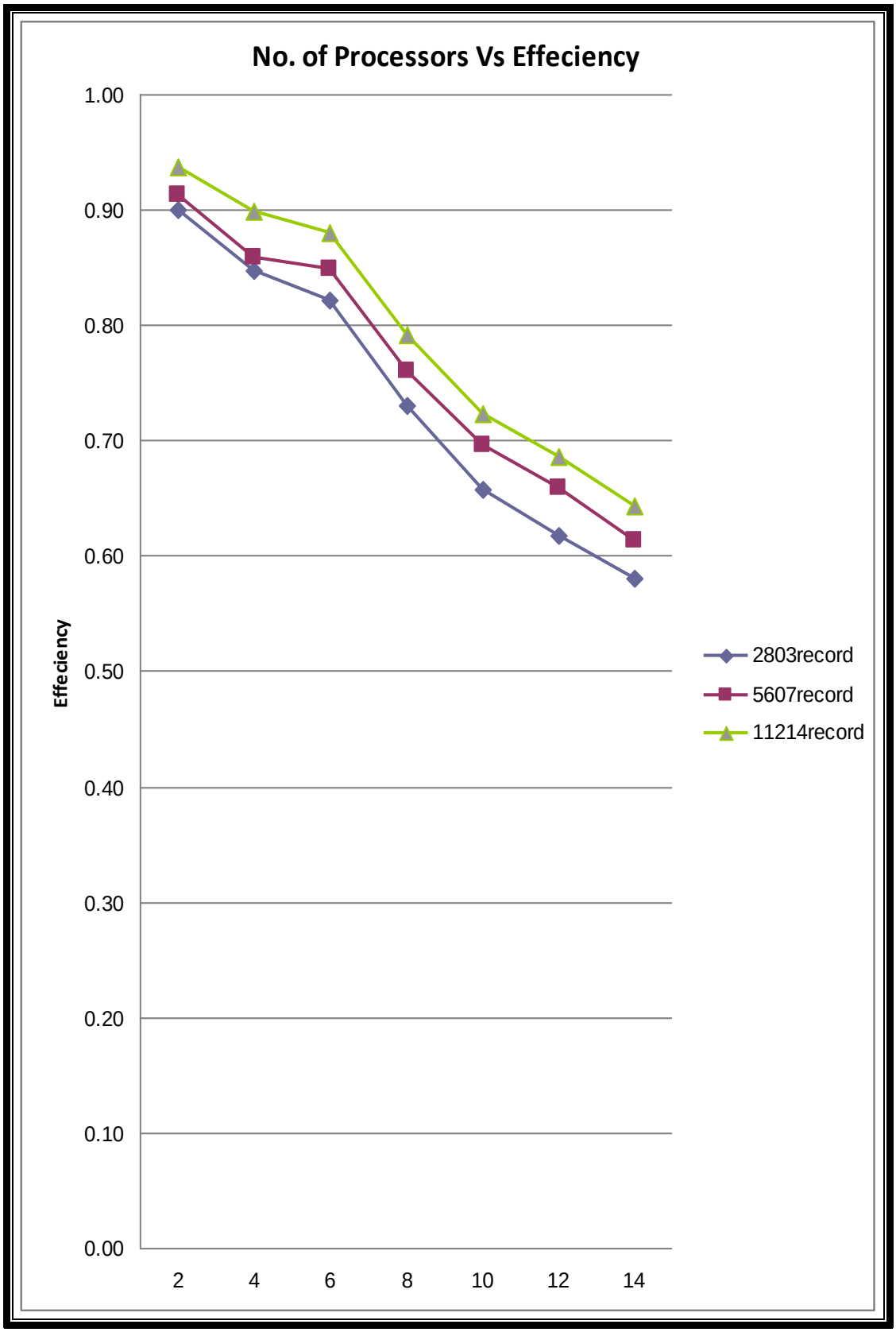

Figure 6.3: The Efficiency Curves of the Proposed Parallel Classifier.

Also, we compute the parallel overhead. The parallel overhead values are recorded in Table 6.5. Figure 6.4 illustrates the parallel overhead curves.

Table 6.5: The Parallel Overhead of the Proposed Parallel Classifier.

Problem Size / No. of Processors	2803 Documents	5607 Documents	11214 Documents
2-Processors	0.11	0.09	0.07
4-Processors	0.18	0.16	0.11
6-Processors	0.22	0.18	0.14
8-Processors	0.37	0.32	0.26
10-Processors	0.52	0.44	0.38
12-Processors	0.62	0.52	0.46
14-Processors	0.72	0.63	0.56

As we note from Table 6.5, the parallel overhead of the parallel classifier increases as we increase the number of processing elements for a given problem size. This is a normal situation when the problem size is fixed as the number of processors increases. However, it can be solved by scaling the problem size. we note that the parallel classifier has a parallel overhead that decreases as the data set increases (from 2803 documents to 11214 documents). It can be seen that our parallel classifier yields better performance for the larger data sets.

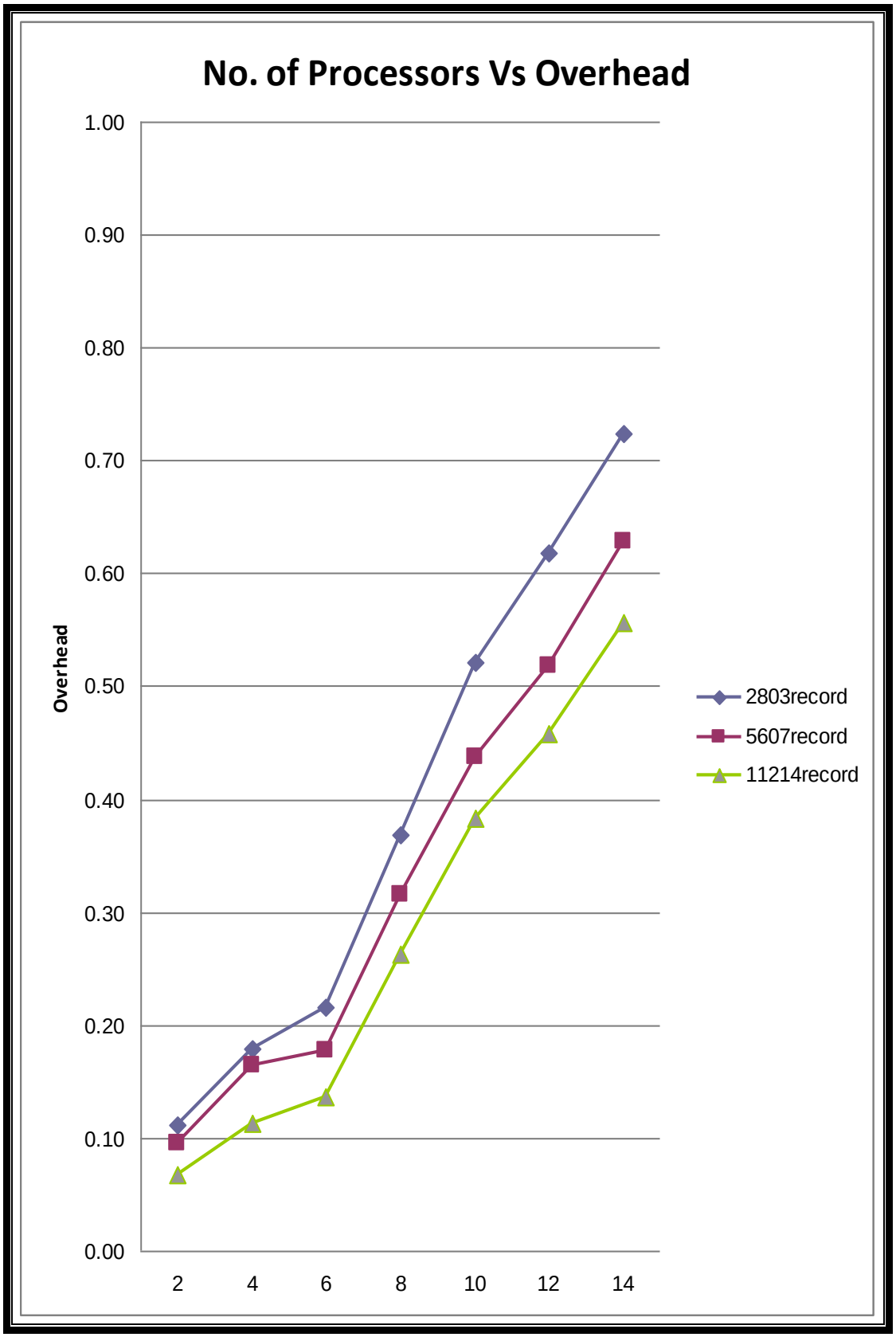

Figure 6.4: The Parallel Overhead Curves of the Proposed Parallel Classifier.

6.3.2 Comparison with Related Approaches

We now compare our work with related approaches along nine criteria which are the most common criteria. The comparison between our work and the related approaches is summarized in Table 6.6. The nine criteria we use are: 1) The language. 2) The size of dataset. 3) The type of dataset. 4) the number of processors. 5) The speedup. 6) The parallel platform. 7) The programming model. 8) The processor speed, and 9) The memory size.

Research efforts have focused on shared memory parallelization of the k-NN algorithm. Ruoming et. al [37], proposed a parallel learning algorithm. The parallel algorithm is based on the k-NN algorithm. They evaluated the parallel implementation on a multiprocessor with shared memory that connect multiple processors to a single memory system. They experimented with a 800 MB main memory resident dataset. The reduction object in this algorithm's parallel implementation is the list of k-nearest neighbors. The speedup results was suitable up to four processors. However, sharing memory in this way can easily lead to a performance bottleneck and the scalability of the processors is limited. Their Experiments are performed on a shared memory machine with 4 (1 GHz) processors and 1 GB of memory.

Our work is significantly different, because on the largest tested set (11214 documents), the parallel classifier achieved the relative speedup of 9.00 on 14 processors. It 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 2803 documents to 11214 documents). We implemented our proposed algorithm with C++ language ,our dataset containing (22428 * 2114) value, the size of the dataset is 241 MB. The target platform for our experiments is a cluster of computers and their own exclusive memory connected through a fast local area network. The cluster consists of 14 nodes, all nodes have the same specifications; Intel(R) Core(TM) i3-2120 CPU @ 3.30 GHz, 4.00 GB RAM.

Table 6.6: The Comparison Between Our Work and Related approaches.

Criteria	Our Experiment	Their Experiment
Language	C++	C++
The Size of Dataset	(22428 * 2114) value, 241 MB	800 MB
The Type of Dataset	The OSAC Arabic corpus	Synthetic two dimensional dataset
Number of processors	2,4,6,8,10,12,14 processors	2,3,4 processors
The Speedup	1.87, 3.59, 6.33, and 9.00 on 2, 4, 8, and 14 processors	1.75, 2.22, and 2.24 on 2, 3, and 4 processors
The Parallel Platform	A multicomputer cluster	A shared memory multiprocessor
The Programming Model	MPI	OpenMP
The Processor Speed	3.30 GHz	1 GHz
The Memory Size	4 GB	1 GB

6.3.3 Discussion of the Classification Results

To ensure that the classifier works well with the tested documents, we also examined the quality of the classification. we split all generated text representations of OSAC corpus (we have described these text representations in section 6.1) into two parts; 50% of the corpus for training (11214 documents) and the remaining 50% for testing (11214 documents) using stratified sampling which keep class

distributions remains the same after splitting. Then we convert these text data parts into two text files with .txt format in order to read it by the classifier. We used the open source machine learning tool RapidMiner for this purpose. We splitting the corpus in this way to achieve higher classification results.

For the purpose of evaluating the classification results, we use confusion matrices that are the primary source of performance measurement for the classification problem. Each column of the confusion matrix represents the instances in an actual class, while each row represents the instances in a predicted class as shown in Table 6.7.

Table 6.7: 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):** refer to the number of positive instances that correctly labeled the classifier [21].
- **True Negative (TN):** refer to number of negative instances that correctly labeled the classifier [21].
- **False Positive (FP):** refer to the number of negative instances that were incorrectly labeled the classifier [21].

  False Negative (FN): refer to number of positive instances that were incorrectly labeled the classifier [21].

We have evaluated the obtained classification results using different classification measures such as accuracy (Eq. 6.1), precision (Eq. 6.2), recall (Eq. 6.3), and F-measure (Eq. 6.4) which are generally accepted ways of measuring systems' success in this field.

- **Accuracy:** refer the percentage of test set instances that are correctly classified by the classifier [21].

$$Overall\ Accuracy = (TP+TN) / (TP+TN+FP+FN) \quad (6.1)$$

- **Precision:** refer to the percentage of predicted documents for the given topic that are correctly classified [21].

$$Precision = TP / (TP+FP) \quad (6.2)$$

- **Recall:** refer to the percentage of the total documents for the given topic that are correctly classified [21].

$$Recall = TP / (TP+FN) \quad (6.3)$$

- **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 [21].

$$F\text{-measure} = 2*Precision*Recall / (Precision + Recall) \quad (6.4)$$

In our experiments, we computed the accuracy, precision, recall, and F-measure for all generated text representations of OSAC corpus (we have described these text representations in section 6.1). The average classification results are recorded in Table 6.8.

The morphological analysis (stemming, light stemming), term pruning and term weighting schemes (TF-IDF, TF, TO, BTO) have obvious impact on the classifier performance as shown in Figure 6.5.

The Figure emphasizes that light stemming and TF representation with k=10 has the best classification results, this is because light stemming is more proper than

stemming from linguistics and semantic view point and keeps the words meanings unaffected.

The Figure also emphasizes that the classifier is very sensitive to term weighting schemes because it depends on distance function to determine the nearest neighbors. For example, the BTO weighting scheme has the worst classification results because the text representation is 0 or 1.

Table 6.8: The Classification Results for All Text Representations of OSAC.

Performance Measures / Text Representations	Accuracy	Precision	Recall	F-Measure
light stemming + TF-IDF	96.12	95.89	95.36	95.62
light stemming + TF	96.35	96.18	95.36	95.77
light stemming + TO	92.77	93.58	92.20	92.88
light stemming + BTO	77.80	91.55	79.56	85.13
Stemming + TF-IDF	93.10	91.91	92.35	92.13
Stemming + TF	93.83	93.09	92.99	93.04
Stemming + TO	89.60	89.12	88.98	89.05
Stemming + BTO	77.45	89.16	78.93	83.73

When we recorded the performance for each class of the ten categories for the best text representation (light stemming + TF) that achieved the best classification results, we got the results as in Table 6.9.

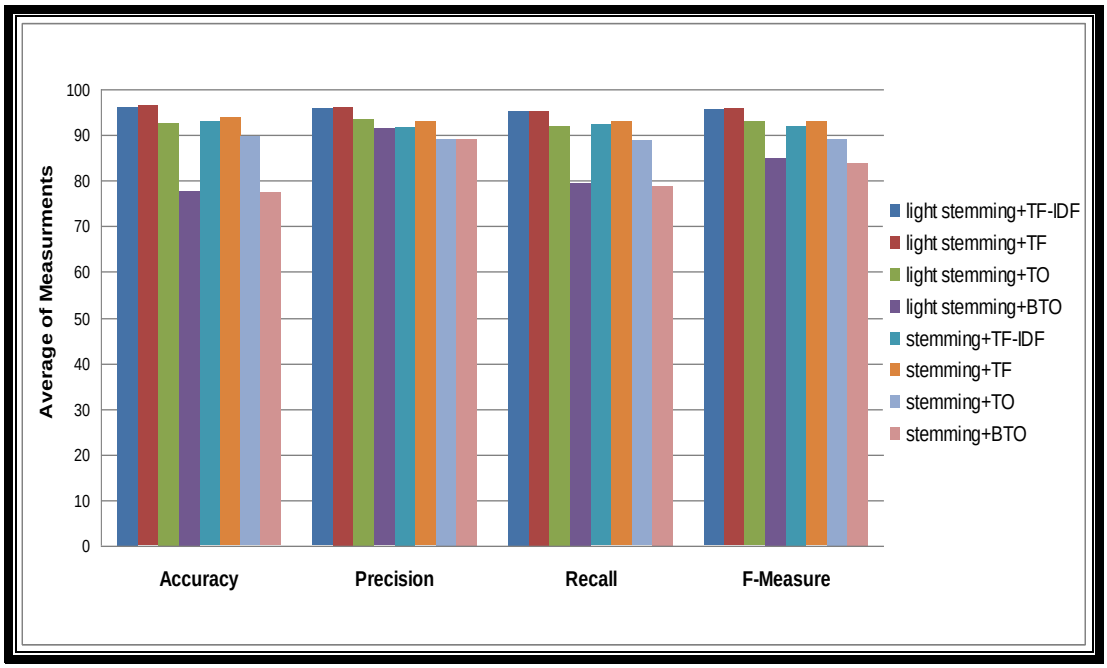

Figure 6.5: The Classification Results for All Text Representations of OSAC.

Table 6.9: The Classification Results for Light Stemming + TF.

Performance Measures / Category	Precision	Recall	F-Measure
Education and Family	93.33	94.68	94.00
History	93.10	96.91	94.97
Stories	91.33	81.27	86.01
Sport	98.19	98.92	98.55
Low	95.71	94.49	95.10
Astronomy	97.14	97.84	97.49
Cooking Recipes	99.24	99.58	99.41
Religious and Fatwas	99.16	96.28	97.70
Health	96.95	96.86	96.90
Economic	97.66	96.78	97.22

Figure 6.6 shows the classification results for the best text representation of OSAC corpus (light stemming + TF) in each of the domain category. From Figure 6.6 we can see that the best performance is recorded in Cooking Recipes domain that because Cooking Recipes has limited space of words that are limited and cleared comparing to other domains. Also, it shows that Stories has lowest performance may be that also because Stories have a large space domain.

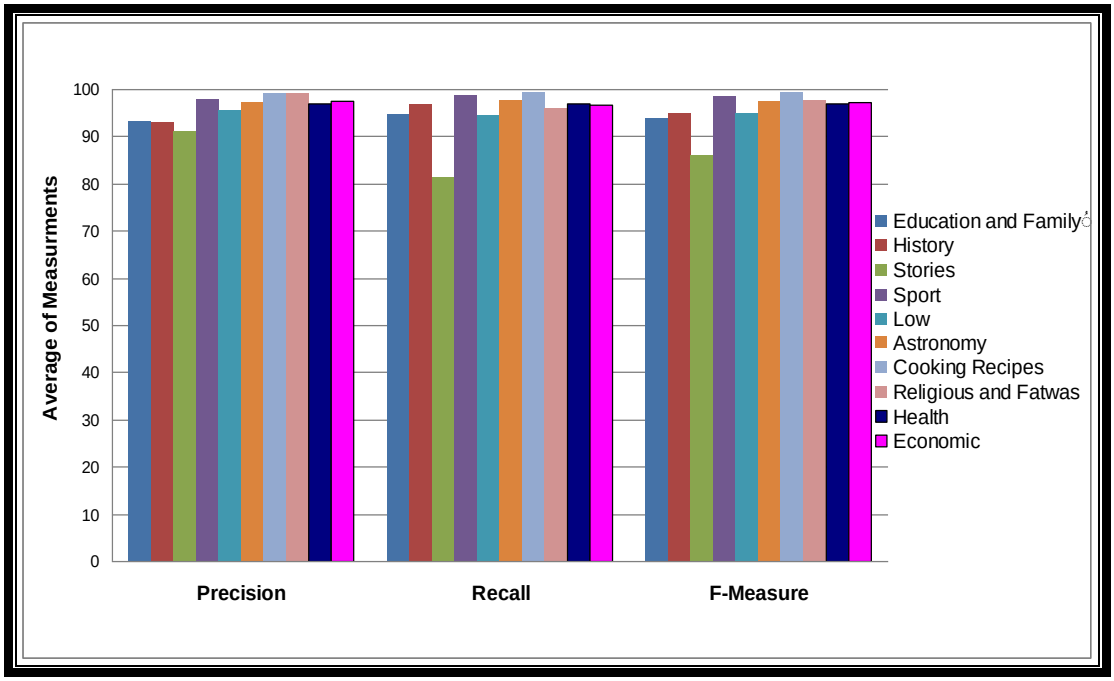

Figure 6.6: The Classification Results for Light Stemming + TF.

Chapter 7 Conclusion and Future Works

7.1 Conclusion

Text classification has become one of the most important techniques in text mining. One of the common classification algorithms is the k-NN which is known to be one of the best classifiers applied for different languages including Arabic language. However, the k-NN algorithm is of low efficiency because it requires a large amount of computational power for evaluating a measure of the similarity between a test document and every training document and for sorting the similarities. Such a drawback makes it unsuitable to handle a large volume of text documents with high dimensionality and in particular in the Arabic language.

In this thesis, a parallel classifier for large-scale Arabic text has been introduced. The proposed parallel classifier is based on the sequential k-NN algorithm.

Five stages are involved in the approach: determine the large text collection, preprocess the text in this collection, design the proposed parallel classifier model, implement the sequential k-NN algorithm as well as the proposed parallel classifier, and conduct the experiments.

We tested the parallel classifier using the OSAC corpus which is the largest freely public Arabic corpus of text documents.

We experimented the parallel classifier on a multicomputer cluster that consists of 14 computers. 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%.

Finally, The proposed parallel classifier can be used efficiently and accurately to categorize a large volume of Arabic text with high dimensionality and solved the problem of low efficiency for the sequential k-NN algorithm. It is suitable for applications where the classification efficiency is crucial such as online text classification, in which the classifier has to respond to a lot of documents arriving simultaneously in stream format.

7.2 Future Works

There are several directions for improvement and future investigation. Our work can be extended to cover larger computer clusters and text corpora to assess the performance of our parallel implementation. Additionally, we can apply this parallel classifier to various application domains such as weather data, internet traffic, log files, medical information, among others to check its generalization. We will also extend our work to cover a popular distributed programming paradigms like MapReduce in a cloud environment.

We believe that our results are encouraging and show that managed code can deliver high performance classifiers. In the future we will investigate further algorithms and apply them to interesting applications.

References

[1] Abdelali, A., Cowie, J. and Soliman, H. (2005) 'Building a modern standard corpus', Workshop on Computational Modeling of Lexical Acquisition, In the Split Meeting.

[2] Al-Shalabi, R., Kannan, G. and Al-Serhan, H. (2003) 'New approach for extracting Arabic roots', The International Conference on Information Technology (ACIT 2003) – Conference Proceedings, Egypt.

[3] Al-Shalabi, R., Kannan, G. and Gharaibeh, H. (2006) 'Arabic text categorization using K-NN algorithm', The 4th International Multiconference on Computer and Information Technology (CSIT 2006) – Conference Proceedings, Amman, Jordan.

[4] Apte, C., Damerau, F. and Weiss, S. (1998) 'Text mining with decision rules and decision trees', The Conference on Automated Learning and Discovery (CONALD 1998) – Conference Proceedings, Pittsburgh, USA, June.

[5] "Arabic language" - Wikipedia, the free encyclopedia, [Online], Available: http://ar.wikipedia.org/wiki/لغة_عربية [19 December 2012].

[6] Buana, P., Jannet, S. and Putra, l. (2012) 'Combination of K-Nearest Neighbor and K-Means based on Term Re-weighting for Classify Indonesian News', International Journal of Computer Applications, vol. 50, no. 11, pp. 37-42.

[7] Buyya, R. (1999) High Performance Cluster Computing: Architectures and systems, Prentice Hall.

[8] Croskey, R. Microkernel based OS, [Online], Available: http://books.google.ps/books?id=kXT6hwzJWWcC&printsec=frontcover&dq=inauthor:%22Rodrigo+Carvallo+Croskey%22&hl=ar&sa=X&ei=_HL5UPWUJuT-4QSm3oAY&ved=0CCsQ6wEwAA [19 November 2012].

[9] Culler, D., Singh, J. and Gupta, A. (1998) Parallel Computer Architecture : A Hardware/Software Approach, Morgan Kaufmann.

[10] Dayde, M. and Dongarra, J. (2005) High Performance Computing for Computational Science, VECPAR.

[11] Duwairi, R., Al-Refai, M., Khasawneh, N. (2009) 'Feature reduction techniques for Arabic text categorization', Journal of the American Society for Information Science, vol. 60, no. 11, pp. 2347-2352.

[12] Duwairi, R., Al-Refai, M. and Khasawneh, N. (2007) 'Stemming Versus Light Stemming as Feature Selection Techniques for Arabic Text Categorization', The 4th International Conference of Innovations in Information Technology (IIT 2007) – Conference Proceedings, pp. 446 – 450.

[13] El-Halees, A. (2008) 'A Comparative Study on Arabic Text Classification', Egyptian Computer Science Journal, vol. 30 , no. 2.

[14] El-Kourdi, M., Bensaid, A., and Rachidi, T. (2004) 'Automatic Arabic Document Categorization Based on the Naïve Bayes Algorithm', The 20th international conference on Computational Linguistics – Conference Proceedings, Geneva, August.

[15] Fasiku, A., Olawale, J. and Jinadu, O. (2012) 'A Review of Architectures - Intel Single Core, Intel Dual Core and AMD Dual Core Processors and the Benefits', International Journal of Engineering and Technology, vol. 2, no. 5, pp. 809-817.

[16] Feldman, R. and Sanger, J. (2007) The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data, Cambridge University Press.

[17] Goux, E., Kulkarni, S., Linderoth, J. and Yoder, M. (2000) 'Master-Worker: An Enabling Framework for Applications on the Computational Grid', The 9th IEEE International Symposium on High Performance Distributed Computing (HPDC 2000) – Conference Proceedings, pp. 43-50.

[18] Guan, J. and Zhou, S. (2002) 'Pruning training corpus to speed up text classification', The 13th International Conference on Database and Expert Systems Applications (DEXA 2002) – Conference Proceedings, Aix-en-Provence, France, September, vol. 2453, pp. 831-840.

[19] Grama, A., Gupta, A., Karypis, G. and Kumar, V. (2003) Introduction to Parallel Computing, 2nd edition, Addison Wesley.

[20] Gropp, W., Lusk, E., Doss, N. and Skjellum, A. (1996) 'A high-performance, portable implementation of the MPI message passing interface standard', Journal of Parallel Computing, vol. 22, no. 6, pp. 789-828.

[21] Han, J. and Kamber, M. (2006) Data Mining: Concepts and Techniques, 2nd edition. The Morgan Kaufmann Series in Data Management Systems, Jim Gray, Series Editor.

[22] Hill, T. and Lewicki, P. (2007) STATISTICS Methods and Applications, 1st edition, StatSoft, Tulsa, OK.

[23] Jing, L., Huang, H. and Shi, H. (2002) 'Improved feature selection approach TFIDF in text mining', The 1st International Conference of machine learning and cybernetics – Conference Proceedings, Beijing.

[24] Joachims, T. (1998) 'Text Categorization with Support Vector Machines: Learning with Many Relevant Features', The 10th European Conference on Machine Learning (ECML 1998) – Conference Proceedings, London, UK, pp. 137-142.

[25] Khoja, S. and Garside, R. (1999) 'Stemming Arabic text', Computer Science Department, Lancaster University, Lancaster, UK.

[26] "k-nearest neighbor algorithm" - Wikipedia, the free encyclopedia, [Online], Available: http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm [13 November 2012].

[27] Kruengkrai, C. and Jaruskulchai, C. (2002) 'A parallel learning algorithm for text classification', The 8th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2002) – Conference Proceedings, New York, USA, pp. 201-206.

[28] Larkey, L., Ballesteros, L. and Connell, M. (2007) 'Light Stemming for Arabic Information Retrieval', Arabic Computational Morphology, book chapter, Springer.

[29] Lewis, D. (1998) ‘Naïve (Bayes) at forty: The Independent Assumption in Information Retrieval’, The 10th European Conference on Machine Learning (ECML 1998) – Conference Proceedings, Berlin, pp. 4–15.

[30] Lianga, S., Liua, Y., Wangb, C., and Jiana, L. (2009) ‘CUKNN: A parallel Implementation of k-Nearest Neighbor on Cuda-Enabled GPU’, The 2009 IEEE Youth Conference on Information, Computing and Telecommunication (ICT2009) – Conference Proceedings, pp. 415-418.

[31] MacDonald, N., Minty, E., Harding, T. and Brown, S. Writing Message Passing Parallel Programs with MPI: A Two Day Course on MPI Usage, Edinburgh Parallel Computing Centre, The University of Edinburgh.

[32] Manning, D., Raghavan, P. and Schütze, H. (2006) An introduction to information retrieval, Cambridge, England: Cambridge University Press.

[33] Message Passing Interface Forum (1994) ‘MPI: A message-passing interface standard’.

[34] Moore, G. (1998) ‘Cramming More Components onto Integrated Circuits’, The Proceedings of the IEEE, vol. 86 , no. 1, pp. 82-85.

[35] Nguyen, A. (2011) ‘Development Of A Framework For Structural Optimization Using Parallel Computers’, M.Sc. Dissertation, Department of Computer Engineering, Ruhr University of Bochum.

[36] Nishida, K. (2008) ‘Learning and Detecting Concept Drift’, Ph.D. Dissertation, Department of Information Science and Technology, Hokkaido University.

[37] Ruoming, J., Yang, G. and Agrawal, G. (2005) ‘Shared memory parallelization of data mining algorithms: Techniques, programming interface and performance’, IEEE Transactions on Knowledge and Data Engineering, vol. 17, no .1, pp. 71-89.

[38] Saad, M. and Ashour, W. (2010) ‘OSAC: Open Source Arabic Corpus’, The 6th International Conference on Electrical and Electronics Engineering and

Computer Science (EEECS 2010) – Conference Proceedings, European University of Lefke, Cyprus, November 25-26, pp. 1-6.

[39] Saad, M. and Ashour, W. (2010) 'Arabic Text Classification Using Decision Trees', The 12th international workshop on computer science and information technologies (CSIT 2010) – Conference Proceedings, Moscow, Saint-Petersburg, Russia, vol. 2, pp. 75-79.

[40] Saad, M. (2010) 'The Impact of Text Preprocessing and Term Weighting on Arabic Text Classification', M.Sc. Dissertation, Department of Computer Engineering, The Islamic University-Gaza.

[41] Saad, M. (2010) 'Open Source Arabic Language and Text Mining Tools', (2010, August), [Online], Available: http://sourceforge.net/projects/ar-text-mining [10 August 2012].

[42] Said, D., Wanas, N., Darwish, N. and Hegazy, N. (2009) 'A Study of Arabic Text preprocessing methods for Text Categorization', The 2nd International Conference of on Arabic Language Resources and Tools – Conference Proceedings, Cairo, Egypt.

[43] Salton, G. and Buckley, C. (1998) 'A Study of Arabic Text preprocessing methods for Text Categorization', The Conference of of information processing & management – Conference Proceedings, vol. 24, no. 5, pp. 513-523.

[44] Sauban, M. and Pfahringer, B. (2003) 'Text Categorization Using Document Profiling', The 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2003) – Conference Proceedings, Cavtat-Dubrovnik, Croatia, September 22-26, pp. 411-422.

[45] Sebastiani, F. (2002) 'Machine learning in automated text categorization', Journal of ACM Computing Surveys (CSUR), vol. 34 , no. 1, pp. 1-47.

[46] Sloan, J. (2004) High Performance Linux Clusters, Nutshell Handbooks Series.

[47] Sutter, H. (2005) 'The free lunch is over: a fundamental turn toward concurrency in software', Dr. Dobb's Journal, vol. 30 , no. 3.

[48] Tan, S. (2005) ‘Binary k-nearest neighbor for text categorization’, Online Information Review, vol. 29, no. 4, pp. 391-399.

[49] Tekiner, F., Tsuruoka, Y., Tsujii, J. and Ananiadou, S. (2009) ‘Highly Scalable Text Mining – Parallel Tagging Application’, The 5th International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control (ICSCCW 2009) – Conference Proceedings, September, pp. 1-4.

[50] “Tokenization” - Wikipedia, the free encyclopedia, [Online], Available: http://en.wikipedia.org/wiki/Tokenization [19 December 2012].

[51] Tripathy, M. and Tripathy, C. (2013) ‘A Review on Literature Survey of Clusters’, International Journal of Advances in Engineering & Technology, vol. 5, no. 2, pp. 381-398.

[52] Yang, Y. (1999) ‘An Evaluation of Statistical Approaches to Text Categorization’, Journal of Information Retrieval, vol. 1 , no. 1-2, pp. 69-90.

[53] Yang, Y., Slattery, S. and Ghani, R. (2002) ‘A Study of approaches to hypertext Categorization’, Journal of Intelligent Information Systems, vol. 18, no. 2-3, pp. 219-241.

[54] Yiqun, C. (2005) ‘Parallel and Distributed Techniques in Biomedical Engineering’, M.Sc. Dissertation, Department of Electrical Computer Engineering, National University of Singapore.

[55] Zufrin, R. (1995) ‘Decision trees on parallel processors’, The International Joint Conference on Artificial Intelligence (IJCAI 1995) – Conference Proceedings, Montreal, Canada, August.

Appendix A

Parts of the Classifiers Source Code

Figure A.1 to A.6 display parts of the source code used to implement the classifiers.

```
// Calculate the distance between the query-instance and all the training samples.
 for(int t = 0; t < 11214; t++)
 {
     float distance=0.0;

     for (y = 0; y < NR; y++) {

         distance=0.0;

         for (x = 0; x < NC-2 ; x++) {

             if((data[y][x]!=0)||(test[t][x]!=0))

                 distance =  distance + pow((data[y][x] - test[t][x]),2);
         }
         dist[y][0] = sqrt(distance);
         dist[y][1] = data[y][NC-2];
     }

     //sort the distances using Quick sort algorithm.

     int left=0,right=NR-1;

     quickSort(dist, left, right);
```

Figure A.1: Calculate the Distance and Sort the Distances.

```
// Determine nearest neighbors and majority class
int c[C]={0};

for(int i=0; i<K; i++){

     for(int j=0; j<C; j++)
          if(dist[i][1]==(j)) c[j]++;
}

int class1 = 0;
for(int i=0; i<C; i++){
     if(c[i] > c[class1]){
          class1 = i;
     }
}
```

Figure A.2: Determine the Nearest Neighbors and Determine the Majority Class.

```
// Quick Sort Function
void quickSort(float** dist, int left, int right){

     int i = left, j = right;
     float tmp;
     float pivot = dist[(left + right) / 2][0];
      //  partition
     while (i <= j) {
          while (dist[i][0] < pivot)
               i++;
          while (dist[j][0] > pivot)
               j--;

          if (i <= j) {
               for(int m=0; m<=1; m++){
                    tmp = dist[i][m];
                    dist[i][m] = dist[j][m];
                    dist[j][m] = tmp;
               }
               i++;
               j--;
          }
     }
       // recursion
     if (left < j)
          quickSort(dist, left, j);

     if (i < right)
          quickSort(dist, i, right);
}
```

Figure A.3: The Quick Sort Function.

```
//Initialize MPI
rc = MPI_Init(&argc,&argv);

//Error starting MPI program; Terminating.
if (rc != MPI_SUCCESS) {
 MPI_Abort(MPI_COMM_WORLD, rc);
 }
//get the size of the process group
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);

//get the process ID number
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
```

Figure A.4: Initializing MPI and Defining Communicator

```
// send matrix data to the worker tasks

mtype = FROM_MASTER;
averow = NR/numworkers;
extra = NR%numworkers;
offset = 0;

for (dest=1; dest<=numworkers; dest++){
    rows = (dest <= extra) ? averow+1 : averow;

    // send the number of rows each process is required to compute
    MPI_Send(&rows, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);

    // send each process rows*NC bits of data starting at offset
    MPI_Send(&data[offset][0], rows*NC, MPI_FLOAT, dest, mtype, MPI_COMM_WORLD);
    offset = offset + rows;
}
count=-1;
for(int s = 0; s < 11214; s++)
{
    count=count+1;
    for (dest=1; dest<=numworkers; dest++){
        MPI_Send(&test1[count][0], NC1, MPI_FLOAT, dest, mtype, MPI_COMM_WORLD);
    }
    mtype = FROM_WORKER;
    offset = 0;
    for (i=1; i<=numworkers; i++){
        source = i;
        // recive each worker k'th elements

        MPI_Recv(&result[offset][0], K*2, MPI_FLOAT, source, mtype, MPI_COMM_WORLD, &status);
        offset = offset + K;
    }
```

Figure A.5: The Essential Work for the Master Processor.

The master processor divides the training data equally according to the number of worker processors and sending a one data partition for each of them with new document to be classified and receives from each worker the k-th ordered list and combining them in a k-th master list.

```
if (taskid > MASTER){
    mtype = FROM_MASTER;

    // receive the number of rows the worker required to compute
    MPI_Recv(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);

    // receive the matrix
    MPI_Recv(&data1[0][0], rows*NC, MPI_FLOAT, MASTER, mtype, MPI_COMM_WORLD, &status);

    for(int t = 0; t < 11214; t++)
    {
        MPI_Recv(&test[0][0], NC1, MPI_FLOAT, MASTER, mtype, MPI_COMM_WORLD, &status);

        float distance=0.0;

        // compute  distance
        for (y = 0; y < rows; y++) {
            distance=0.0;
            for (x = 0; x < NC-2 ; x++) {
            if((data1[y][x]!=0)||(test[0][x]!=0))
            distance =  distance + pow((data1[y][x] - test[0][x]),2);
            }
            dist[y][0] = sqrt(distance);
            dist[y][1] = data1[y][NC-2];
        }
        int left=0,right=rows-1;
        quickSort(dist, left, right);
        mtype = FROM_WORKER;
        // send (return) first k recors
        MPI_Send(&dist[0][0], K*2, MPI_FLOAT, MASTER, mtype, MPI_COMM_WORLD);
    }
```

Figure A.6: The Essential Work for the Worker Processors.

The worker processors receives its data partition with new document to be classified and calculates the distance between the new test document and all the training samples in its data partition, sorts the distance and determine nearest neighbors based on the k-th minimum distance locally. Then sends the k-th ordered list to the master processor which include the k-th distances and classes.

Appendix B

The Text Preprocessing Using RapidMiner

We used the open source machine learning tool RapidMiner for text preprocessing for the OSAC corpus including tokenizing string to words, normalizing the tokenized words, applying stopwords removal, applying the term stemming and pruning methods as a feature reduction techniques, and finally applying the term weighting schemes to enhance text document representation as feature vector. Figure B.1 depicts the whole process of applying the text preprocessing methods in the OSAC corpus.

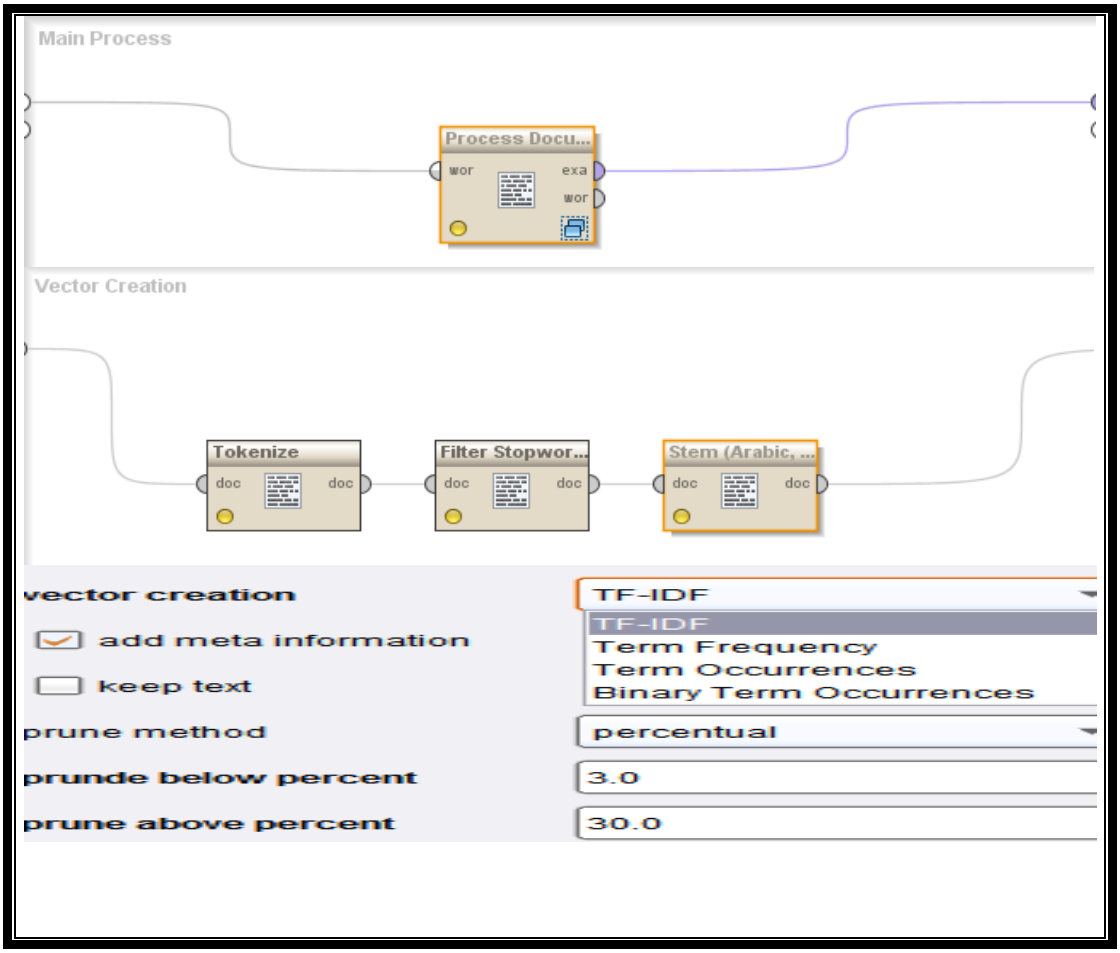

Figure B.1: The Process of Applying the Text Preprocessing in the OSAC Corpus.

Table B.1 to B.8 display parts of the generated text representations for OSAC corpus.

Table B.1: Part of the Light Stemming + TF-IDF Text Representation (22428 Record and 2114 Attribute).

Row No.	label	ا	اء	الم	اب	ابا	اباح	اباطر	ابحاث	ابحث	ابد	ابراهيم	ابرز	ابطال	ابن
1	بيئة و اسرة و مرأة	0	0	0	0	0	0.021	0	0	0	0	0	0	0	0
2	بيئة و اسرة و مرأة	0	0	0	0	0	0.024	0	0	0	0	0	0	0	0.019
3	بيئة و اسرة و مرأة	0	0	0	0.010	0.021	0.014	0	0	0	0	0	0	0	0.033
4	بيئة و اسرة و مرأة	0	0	0	0	0	0.026	0	0	0	0	0	0	0	0
5	بيئة و اسرة و مرأة	0	0	0	0.013	0	0.009	0	0	0.011	0.025	0	0	0	0.020
6	بيئة و اسرة و مرأة	0.948	0.110	0	0.061	0	0.003	0	0	0	0	0	0	0	0
7	بيئة و اسرة و مرأة	0.615	0.053	0.012	0.028	0	0.010	0	0	0	0	0.011	0	0	0
8	بيئة و اسرة و مرأة	0.294	0.043	0	0.038	0	0.021	0	0	0	0	0	0	0	0.008
9	بيئة و اسرة و مرأة	0.129	0	0	0	0	0.020	0	0	0	0	0	0	0.029	0.031
10	بيئة و اسرة و مرأة	0	0	0	0	0	0.024	0	0	0	0.035	0	0	0	0
11	بيئة و اسرة و مرأة	0	0	0	0	0	0.033	0	0	0	0	0	0	0	0
12	بيئة و اسرة و مرأة	0.089	0.048	0	0.038	0	0.018	0	0	0	0	0	0.044	0	0.014
13	بيئة و اسرة و مرأة	0.122	0.029	0	0.015	0	0.021	0	0	0	0	0	0	0	0.033
14	بيئة و اسرة و مرأة	0.447	0.037	0	0.024	0	0.013	0	0.011	0	0.010	0	0.008	0	0
15	بيئة و اسرة و مرأة	0	0	0	0.097	0	0.019	0	0	0	0	0	0	0	0
16	بيئة و اسرة و مرأة	0.008	0	0	0.003	0	0.004	0	0.006	0.005	0	0	0	0	0.003
17	بيئة و اسرة و مرأة	0.125	0	0	0.007	0	0.010	0	0	0	0.014	0.022	0.024	0	0.007
18	بيئة و اسرة و مرأة	0	0	0	0.007	0.014	0.010	0	0	0	0	0	0.012	0.014	0.091
19	بيئة و اسرة و مرأة	0.022	0	0	0	0	0.060	0	0.024	0	0	0	0	0	0
20	بيئة و اسرة و مرأة	0.070	0.026	0	0.014	0	0.019	0	0	0	0	0	0	0	0.030
21	بيئة و اسرة و مرأة	0	0	0	0.017	0	0.023	0	0.018	0	0	0	0.029	0	0
22	بيئة و اسرة و مرأة	0	0	0	0.015	0	0.021	0	0	0.027	0	0.025	0	0	0
23	بيئة و اسرة و مرأة	0	0	0	0	0	0.014	0	0	0	0	0	0	0	0.011
24	بيئة و اسرة و مرأة	0.025	0	0.021	0	0	0.086	0	0	0	0	0	0	0	0.080
25	بيئة و اسرة و مرأة	0	0	0	0	0	0.047	0	0	0.020	0	0	0	0	0

Table B.2: Part of the Light Stemming + TF Text Representation (22428 Record and 2114 Attribute).

Row No.	label	ا	اء	ائم	اب	ابا	اباح	اباطر	ابحاث	ابحث	ابد	ابراهيم	ابرز	ابطال	ابن
1	بيئة و اسرة و مرأة	0	0	0	0	0	0.017	0	0	0	0	0	0	0	0
2	بيئة و اسرة و مرأة	0	0	0	0	0	0.020	0	0	0	0	0	0	0	0.020
3	بيئة و اسرة و مرأة	0	0	0	0.013	0.013	0.013	0	0	0	0	0	0	0	0.038
4	بيئة و اسرة و مرأة	0	0	0	0	0	0.022	0	0	0	0	0	0	0	0
5	بيئة و اسرة و مرأة	0	0	0	0.015	0	0.008	0	0	0.008	0.015	0	0	0	0.023
6	بيئة و اسرة و مرأة	0.975	0.060	0	0.062	0	0.002	0	0	0	0	0	0	0	0
7	بيئة و اسرة و مرأة	0.690	0.032	0.008	0.032	0	0.008	0	0	0	0	0.008	0	0	0
8	بيئة و اسرة و مرأة	0.353	0.027	0	0.045	0	0.018	0	0	0	0	0	0	0	0.009
9	بيئة و اسرة و مرأة	0.152	0	0	0	0	0.017	0	0	0	0	0	0	0.017	0.034
10	بيئة و اسرة و مرأة	0	0	0	0	0	0.020	0	0	0	0.020	0	0	0	0
11	بيئة و اسرة و مرأة	0	0	0	0	0	0.027	0	0	0	0	0	0	0	0
12	بيئة و اسرة و مرأة	0.106	0.030	0	0.046	0	0.015	0	0	0	0	0	0.030	0	0.015
13	بيئة و اسرة و مرأة	0.150	0.019	0	0.019	0	0.019	0	0	0	0	0	0	0	0.038
14	بيئة و اسرة و مرأة	0.471	0.020	0	0.026	0	0.010	0	0.010	0	0.005	0	0.005	0	0
15	بيئة و اسرة و مرأة	0	0	0	0.101	0	0.014	0	0	0	0	0	0	0	0
16	بيئة و اسرة و مرأة	0.011	0	0	0.004	0	0.004	0	0.007	0.004	0	0	0	0	0.004
17	بيئة و اسرة و مرأة	0.141	0	0	0.008	0	0.008	0	0	0	0.008	0.016	0.016	0	0.008
18	بيئة و اسرة و مرأة	0	0	0	0.007	0.007	0.007	0	0	0	0	0	0.007	0.007	0.090
19	بيئة و اسرة و مرأة	0.025	0	0	0	0	0.049	0	0.025	0	0	0	0	0	0
20	بيئة و اسرة و مرأة	0.078	0.016	0	0.016	0	0.016	0	0	0	0	0	0	0	0.031
21	بيئة و اسرة و مرأة	0	0	0	0.019	0	0.019	0	0.019	0	0	0	0.019	0	0
22	بيئة و اسرة و مرأة	0	0	0	0.019	0	0.019	0	0	0.019	0	0.019	0	0	0
23	بيئة و اسرة و مرأة	0	0	0	0	0	0.012	0	0	0	0	0	0	0	0.012
24	بيئة و اسرة و مرأة	0.030	0	0.015	0	0	0.075	0	0	0	0	0	0	0	0.090
25	بيئة و اسرة و مرأة	0	0	0	0	0	0.042	0	0	0.014	0	0	0	0	0

Table B.3: Part of the Light Stemming + TO Text Representation (22428 Record and 2114 Attribute).

Row No.	label	ا	اء	ألم	اب	ابا	اباح	اباطر	ابحاث	ابحث	ابد	ابراهيم	ابرز	ابطال	ابن
1	بيئة و اسرة و مرأة	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	بيئة و اسرة و مرأة	0	0	0	0	0	1	0	0	0	0	0	0	0	1
3	بيئة و اسرة و مرأة	0	0	0	1	1	1	0	0	0	0	0	0	0	3
4	بيئة و اسرة و مرأة	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5	بيئة و اسرة و مرأة	0	0	0	2	0	1	0	0	1	2	0	0	0	3
6	بيئة و اسرة و مرأة	422	26	0	27	0	1	0	0	0	0	0	0	0	0
7	بيئة و اسرة و مرأة	87	4	1	4	0	1	0	0	0	0	1	0	0	0
8	بيئة و اسرة و مرأة	39	3	0	5	0	2	0	0	0	0	0	0	0	1
9	بيئة و اسرة و مرأة	9	0	0	0	0	1	0	0	0	0	0	0	1	2
10	بيئة و اسرة و مرأة	0	0	0	0	0	1	0	0	0	1	0	0	0	0
11	بيئة و اسرة و مرأة	0	0	0	0	0	1	0	0	0	0	0	0	0	0
12	بيئة و اسرة و مرأة	7	2	0	3	0	1	0	0	0	0	0	2	0	1
13	بيئة و اسرة و مرأة	8	1	0	1	0	1	0	0	0	0	0	0	0	2
14	بيئة و اسرة و مرأة	92	4	0	5	0	2	0	2	0	1	0	1	0	0
15	بيئة و اسرة و مرأة	0	0	0	7	0	1	0	0	0	0	0	0	0	0
16	بيئة و اسرة و مرأة	3	0	0	1	0	1	0	2	1	0	0	0	0	1
17	بيئة و اسرة و مرأة	18	0	0	1	0	1	0	0	0	1	2	2	0	1
18	بيئة و اسرة و مرأة	0	0	0	1	1	1	0	0	0	0	0	1	1	12
19	بيئة و اسرة و مرأة	1	0	0	0	0	2	0	1	0	0	0	0	0	0
20	بيئة و اسرة و مرأة	5	1	0	1	0	1	0	0	0	0	0	0	0	2
21	بيئة و اسرة و مرأة	0	0	0	1	0	1	0	1	0	0	0	1	0	0
22	بيئة و اسرة و مرأة	0	0	0	1	0	1	0	0	1	0	1	0	0	0
23	بيئة و اسرة و مرأة	0	0	0	0	0	1	0	0	0	0	0	0	0	1
24	بيئة و اسرة و مرأة	2	0	1	0	0	5	0	0	0	0	0	0	0	6
25	بيئة و اسرة و مرأة	0	0	0	0	0	3	0	0	1	0	0	0	0	0

Table B.4: Part of the Light Stemming + BTO Text Representation (22428 Record and 2114 Attribute).

Row No.	label	ا	اء	ألم	اب	ابا	اباح	اباطر	ابحاث	ابحث	ابد	ابراهيم	ابرز	ابطال	ابن
1	تربية و اسرة و مرأة	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	تربية و اسرة و مرأة	0	0	0	0	0	1	0	0	0	0	0	0	0	1
3	تربية و اسرة و مرأة	0	0	0	1	1	1	0	0	0	0	0	0	0	1
4	تربية و اسرة و مرأة	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5	تربية و اسرة و مرأة	0	0	0	1	0	1	0	0	1	1	0	0	0	1
6	تربية و اسرة و مرأة	1	1	0	1	0	1	0	0	0	0	0	0	0	0
7	تربية و اسرة و مرأة	1	1	1	1	0	1	0	0	0	0	1	0	0	0
8	تربية و اسرة و مرأة	1	1	0	1	0	1	0	0	0	0	0	0	0	1
9	تربية و اسرة و مرأة	1	0	0	0	0	1	0	0	0	0	0	0	1	1
10	تربية و اسرة و مرأة	0	0	0	0	0	1	0	0	0	1	0	0	0	0
11	تربية و اسرة و مرأة	0	0	0	0	0	1	0	0	0	0	0	0	0	0
12	تربية و اسرة و مرأة	1	1	0	1	0	1	0	0	0	0	0	1	0	1
13	تربية و اسرة و مرأة	1	1	0	1	0	1	0	0	0	0	0	0	0	1
14	تربية و اسرة و مرأة	1	1	0	1	0	1	0	1	0	1	0	1	0	0
15	تربية و اسرة و مرأة	0	0	0	1	0	1	0	0	0	0	0	0	0	0
16	تربية و اسرة و مرأة	1	0	0	1	0	1	0	1	1	0	0	0	0	1
17	تربية و اسرة و مرأة	1	0	0	1	0	1	0	0	0	1	1	1	0	1
18	تربية و اسرة و مرأة	0	0	0	1	1	1	0	0	0	0	0	1	1	1
19	تربية و اسرة و مرأة	1	0	0	0	0	1	0	1	0	0	0	0	0	0
20	تربية و اسرة و مرأة	1	1	0	1	0	1	0	0	0	0	0	0	0	1
21	تربية و اسرة و مرأة	0	0	0	1	0	1	0	1	0	0	0	1	0	0
22	تربية و اسرة و مرأة	0	0	0	1	0	1	0	0	1	0	1	0	0	0
23	تربية و اسرة و مرأة	0	0	0	0	0	1	0	0	0	0	0	0	0	1
24	تربية و اسرة و مرأة	1	0	1	0	0	1	0	0	0	0	0	0	0	1
25	تربية و اسرة و مرأة	0	0	0	0	0	1	0	0	1	0	0	0	0	0

Table B.5: Part of the Stemming + TF-IDF Text Representation (22428 Record and 1295 Attribute).

Row No.	label	ء	أ	أبن	أبه	أبي	أتي	أجر	أجل	أحد	أخا	أخت	أخذ	أخر	أدا
1	بيئة و اسرة و مرأة	0	0	0.023	0	0	0.028	0.036	0	0.018	0.038	0	0	0.018	0
2	بيئة و اسرة و مرأة	0	0	0.043	0	0	0.027	0.035	0	0.017	0	0.042	0.023	0.052	0
3	بيئة و اسرة و مرأة	0	0	0.095	0	0.034	0.017	0.022	0	0	0	0.052	0.028	0.044	0
4	بيئة و اسرة و مرأة	0	0	0.030	0	0	0	0	0	0.048	0	0	0	0.024	0
5	بيئة و اسرة و مرأة	0	0	0.023	0	0.007	0.029	0.019	0	0.014	0	0.011	0.030	0.009	0
6	بيئة و اسرة و مرأة	0.082	0.007	0.002	0	0	0	0	0	0	0.003	0	0	0	0
7	بيئة و اسرة و مرأة	0.053	0	0.007	0	0.008	0.008	0.011	0.010	0	0.022	0	0.007	0.016	0
8	بيئة و اسرة و مرأة	0.073	0	0.019	0	0.012	0	0	0	0.015	0	0	0.019	0.022	0
9	بيئة و اسرة و مرأة	0	0	0.076	0	0	0	0	0	0.015	0	0.037	0	0	0
10	بيئة و اسرة و مرأة	0	0	0.021	0.087	0	0	0.034	0.150	0	0	0	0.022	0.050	0
11	بيئة و اسرة و مرأة	0	0	0.083	0	0	0	0	0	0	0	0	0	0	0
12	بيئة و اسرة و مرأة	0.058	0	0.015	0	0.018	0.054	0	0	0.012	0.024	0	0.046	0.047	0
13	بيئة و اسرة و مرأة	0.065	0.033	0.050	0	0	0.061	0	0	0.052	0	0	0	0.026	0
14	بيئة و اسرة و مرأة	0.056	0.012	0.006	0	0.007	0	0.018	0	0.009	0.057	0.022	0.018	0.009	0
15	بيئة و اسرة و مرأة	0	0	0.027	0	0.271	0	0	0	0.043	0	0	0.085	0.022	0
16	بيئة و اسرة و مرأة	0	0	0.022	0	0.004	0.008	0.036	0	0.010	0	0	0.023	0.008	0
17	بيئة و اسرة و مرأة	0	0	0.026	0	0	0.022	0.042	0.013	0.028	0	0	0.036	0.007	0.017
18	بيئة و اسرة و مرأة	0	0	0.151	0	0.014	0	0	0.017	0.018	0	0	0.024	0.019	0
19	بيئة و اسرة و مرأة	0	0	0.031	0	0	0	0	0	0.024	0	0.059	0.032	0.024	0
20	بيئة و اسرة و مرأة	0.046	0	0.071	0	0	0	0	0	0	0	0	0	0.038	0
21	بيئة و اسرة و مرأة	0	0	0.020	0	0	0	0	0	0	0	0	0	0	0.038
22	بيئة و اسرة و مرأة	0	0	0.017	0	0.021	0	0	0	0.040	0	0	0.124	0	0
23	بيئة و اسرة و مرأة	0	0	0.041	0	0	0.017	0	0.039	0.021	0	0	0.014	0.043	0.051
24	بيئة و اسرة و مرأة	0	0	0.116	0	0	0	0	0	0	0	0	0	0.066	0
25	بيئة و اسرة و مرأة	0	0	0.015	0	0	0	0	0	0.012	0	0	0	0.012	0

Table B.6: Part of the Stemming + TF Text Representation (22428 Record and 1295 Attribute).

Row No.	label	ء	أ	أبن	أبه	أبي	أتي	أجر	أجل	أحد	أخا	أخت	أخذ	أخر	أدا
1	بيئة و اسرة و مرأة	0	0	0.024	0	0	0.024	0.024	0	0.024	0.024	0	0	0.024	0
2	بيئة و اسرة و مرأة	0	0	0.052	0	0	0.026	0.026	0	0.026	0	0.026	0.026	0.078	0
3	بيئة و اسرة و مرأة	0	0	0.106	0	0.030	0.015	0.015	0	0	0	0.030	0.030	0.061	0
4	بيئة و اسرة و مرأة	0	0	0.033	0	0	0	0	0	0.066	0	0	0	0.033	0
5	بيئة و اسرة و مرأة	0	0	0.028	0	0.007	0.028	0.014	0	0.021	0	0.007	0.035	0.014	0
6	بيئة و اسرة و مرأة	0.045	0.003	0.002	0	0	0	0	0	0	0.002	0	0	0	0
7	بيئة و اسرة و مرأة	0.029	0	0.007	0	0.007	0.007	0.007	0.007	0	0.015	0	0.007	0.022	0
8	بيئة و اسرة و مرأة	0.048	0	0.024	0	0.012	0	0	0	0.024	0	0	0.024	0.036	0
9	بيئة و اسرة و مرأة	0	0	0.089	0	0	0	0	0	0.022	0	0.022	0	0	0
10	بيئة و اسرة و مرأة	0	0	0.026	0.051	0	0	0.026	0.128	0	0	0	0.026	0.077	0
11	بيئة و اسرة و مرأة	0	0	0.099	0	0	0	0	0	0	0	0	0	0	0
12	بيئة و اسرة و مرأة	0.034	0	0.017	0	0.017	0.052	0	0	0.017	0.017	0	0.052	0.069	0
13	بيئة و اسرة و مرأة	0.036	0.018	0.054	0	0	0.054	0	0	0.072	0	0	0	0.036	0
14	بيئة و اسرة و مرأة	0.032	0.006	0.006	0	0.006	0	0.013	0	0.013	0.038	0.013	0.019	0.013	0
15	بيئة و اسرة و مرأة	0	0	0.033	0	0.262	0	0	0	0.065	0	0	0.098	0.033	0
16	بيئة و اسرة و مرأة	0	0	0.022	0	0.003	0.006	0.022	0	0.013	0	0	0.022	0.010	0
17	بيئة و اسرة و مرأة	0	0	0.030	0	0	0.020	0.030	0.010	0.040	0	0	0.040	0.010	0.010
18	بيئة و اسرة و مرأة	0	0	0.178	0	0.014	0	0	0.014	0.027	0	0	0.027	0.027	0
19	بيئة و اسرة و مرأة	0	0	0.035	0	0	0	0	0	0.035	0	0.035	0.035	0.035	0
20	بيئة و اسرة و مرأة	0.028	0	0.085	0	0	0	0	0	0	0	0	0	0.057	0
21	بيئة و اسرة و مرأة	0	0	0.024	0	0	0	0	0	0	0	0	0	0	0.024
22	بيئة و اسرة و مرأة	0	0	0.021	0	0.021	0	0	0	0.062	0	0	0.145	0	0
23	بيئة و اسرة و مرأة	0	0	0.046	0	0	0.015	0	0.030	0.030	0	0	0.015	0.061	0.030
24	بيئة و اسرة و مرأة	0	0	0.138	0	0	0	0	0	0	0	0	0	0.098	0
25	بيئة و اسرة و مرأة	0	0	0.017	0	0	0	0	0	0.017	0	0	0	0.017	0

Table B.7: Part of the Stemming + TO Text Representation (22428 Record and 1295 Attribute).

Row No.	label	ء	أ	أبن	أبه	أبي	أتي	أجر	أجل	أحد	أخا	أخت	أخذ	أخر	أدا
1	بيئة و اسرة و مرأة	0	0	1	0	0	1	1	0	1	1	0	0	1	0
2	بيئة و اسرة و مرأة	0	0	2	0	0	1	1	0	1	0	1	1	3	0
3	بيئة و اسرة و مرأة	0	0	7	0	2	1	1	0	0	0	2	2	4	0
4	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	2	0	0	0	1	0
5	بيئة و اسرة و مرأة	0	0	4	0	1	4	2	0	3	0	1	5	2	0
6	بيئة و اسرة و مرأة	26	2	1	0	0	0	0	0	0	1	0	0	0	0
7	بيئة و اسرة و مرأة	4	0	1	0	1	1	1	1	0	2	0	1	3	0
8	بيئة و اسرة و مرأة	4	0	2	0	1	0	0	0	2	0	0	2	3	0
9	بيئة و اسرة و مرأة	0	0	4	0	0	0	0	0	1	0	1	0	0	0
10	بيئة و اسرة و مرأة	0	0	1	2	0	0	1	5	0	0	0	1	3	0
11	بيئة و اسرة و مرأة	0	0	2	0	0	0	0	0	0	0	0	0	0	0
12	بيئة و اسرة و مرأة	2	0	1	0	1	3	0	0	1	1	0	3	4	0
13	بيئة و اسرة و مرأة	2	1	3	0	0	3	0	0	4	0	0	0	2	0
14	بيئة و اسرة و مرأة	5	1	1	0	1	0	2	0	2	6	2	3	2	0
15	بيئة و اسرة و مرأة	0	0	1	0	8	0	0	0	2	0	0	3	1	0
16	بيئة و اسرة و مرأة	0	0	7	0	1	2	7	0	4	0	0	7	3	0
17	بيئة و اسرة و مرأة	0	0	3	0	0	2	3	1	4	0	0	4	1	1
18	بيئة و اسرة و مرأة	0	0	13	0	1	0	0	1	2	0	0	2	2	0
19	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	1	0	1	1	1	0
20	بيئة و اسرة و مرأة	1	0	3	0	0	0	0	0	0	0	0	0	2	0
21	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	0	0	0	0	0	1
22	بيئة و اسرة و مرأة	0	0	1	0	1	0	0	0	3	0	0	7	0	0
23	بيئة و اسرة و مرأة	0	0	3	0	0	1	0	2	2	0	0	1	4	2
24	بيئة و اسرة و مرأة	0	0	7	0	0	0	0	0	0	0	0	0	5	0
25	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	1	0	0	0	1	0

Table B.8: Part of the Stemming + BTO Text Representation (22428 Record and 1295 Attribute).

Row No.	label	ء	أ	أبن	أبه	أبي	أتي	أجر	أجل	أحد	أخا	أخت	أخذ	أخر	أدا
1	بيئة و اسرة و مرأة	0	0	1	0	0	1	1	0	1	1	0	0	1	0
2	بيئة و اسرة و مرأة	0	0	1	0	0	1	1	0	1	0	1	1	1	0
3	بيئة و اسرة و مرأة	0	0	1	0	1	1	1	0	0	0	1	1	1	0
4	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	1	0	0	0	1	0
5	بيئة و اسرة و مرأة	0	0	1	0	1	1	1	0	1	0	1	1	1	0
6	بيئة و اسرة و مرأة	1	1	1	0	0	0	0	0	0	1	0	0	0	0
7	بيئة و اسرة و مرأة	1	0	1	0	1	1	1	1	0	1	0	1	1	0
8	بيئة و اسرة و مرأة	1	0	1	0	1	0	0	0	1	0	0	1	1	0
9	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	1	0	1	0	0	0
10	بيئة و اسرة و مرأة	0	0	1	1	0	0	1	1	0	0	0	1	1	0
11	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	0	0	0	0	0	0
12	بيئة و اسرة و مرأة	1	0	1	0	1	1	0	0	1	1	0	1	1	0
13	بيئة و اسرة و مرأة	1	1	1	0	0	1	0	0	1	0	0	0	1	0
14	بيئة و اسرة و مرأة	1	1	1	0	1	0	1	0	1	1	1	1	1	0
15	بيئة و اسرة و مرأة	0	0	1	0	1	0	0	0	1	0	0	1	1	0
16	بيئة و اسرة و مرأة	0	0	1	0	1	1	1	0	1	0	0	1	1	0
17	بيئة و اسرة و مرأة	0	0	1	0	0	1	1	1	1	0	0	1	1	1
18	بيئة و اسرة و مرأة	0	0	1	0	1	0	0	1	1	0	0	1	1	0
19	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	1	0	1	1	1	0
20	بيئة و اسرة و مرأة	1	0	1	0	0	0	0	0	0	0	0	0	1	0
21	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	0	0	0	0	0	1
22	بيئة و اسرة و مرأة	0	0	1	0	1	0	0	0	1	0	0	1	0	0
23	بيئة و اسرة و مرأة	0	0	1	0	0	1	0	1	1	0	0	1	1	1
24	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	0	0	0	0	1	0
25	بيئة و اسرة و مرأة	0	0	1	0	0	0	0	0	1	0	0	0	1	0

We splitted the generated text representations for OSAC corpus into two parts; 50% of the corpus for training (11214 document) and the remaining 50% for testing (11214 document) using stratified sampling which keep class distributions remains the same after splitting. Then we convert these text data parts into two text files with .txt format in order to read it by the classifier. Figure B.2 depicts the whole process of splitting the generated text representations for OSAC corpus.

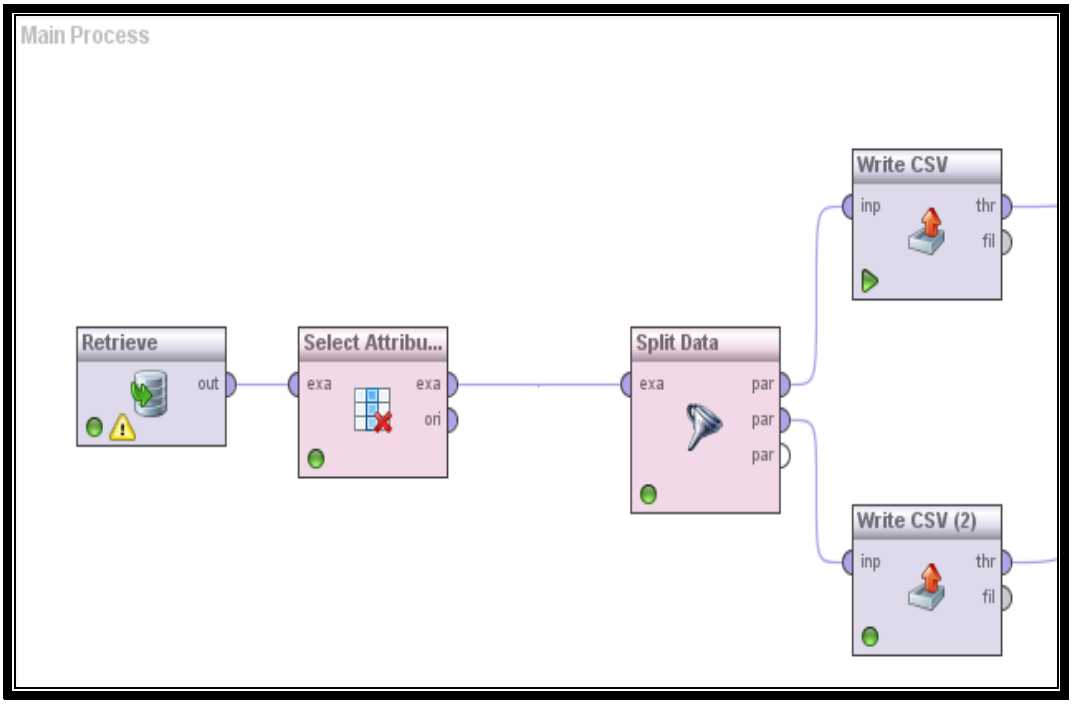

Figure B.2: The Process of Splitting the Text Representations for OSAC Corpus.

Appendix C

Tools and Programs

Special tools and programs are used to complete the implementation of the sequential and parallel classifiers and the text preprocessing of the OSAC corpus:

- **RapidMiner 5:** we used the open source machine learning tool RapidMiner for text preprocessing for the OSAC corpus including tokenizing string to words, normalizing the tokenized words, applying stopwords removal, applying the term stemming and pruning methods as a feature reduction techniques, and finally applying the term weighting schemes to enhance text document representation as feature vector.

- **Microsoft Visual Studio .Net 2008:** this is the program that help us to develop, build, compile, validate and execute our sequential and parallel classifiers using C++ programming language.

- **MPICHI2 Software:** it is a new implementation of MPI. The parallel implementation of the classifier is done using MPI for achieving portable code.

- **Microsoft Office Excel 2010:** it is used to calculate and analyze the results.