# SCALABLE PARTICLE SWARM OPTIMIZATION AND DIFFERENTIAL EVOLUTION APPROACHES APPLIED TO CLASSIFICATION

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Jamil Ahmad Al-Sawwa

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Computer Science

June 2019

Fargo, North Dakota

# NORTH DAKOTA STATE UNIVERSITY

Graduate School

**Title**

SCALABLE PARTICLE SWARM OPTIMIZATION AND DIFFERENTIAL EVOLUTION APPROACHES APPLIED TO CLASSIFICATION

**By**

Jamil Ahmad Al-Sawwa

The supervisory committee certifies that this dissertation complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Chair

Dr. Kendall Nygard

Dr. Jun Kong

Dr. Ying Huang

Approved:

07/01/2019
Date

Dr. Kendall Nygard
Department Chair

# ABSTRACT

Applying the nature-inspired methods in the data mining area has been gaining more attention by researchers. Classification is one of the data mining tasks which aims to analyze historical data by discovering hidden relationships between the input and the output that would help to predict an accurate outcome for an unseen input. The classification algorithms based on nature-inspired methods have been successfully used in numerous applications such as medicine and agriculture. However, the amount of data that has been collected or generated in these areas has been increasing exponentially. Thus, extracting useful information from large data requires computational time and consumes memory space. Besides this, many algorithms suffer from not being able to handle imbalanced data.

Apache Spark is an in-memory computing big data framework that runs on a cluster of nodes. Apache Spark is more efficient for handling iterative and interactive jobs and runs 100 times faster than Hadoop Map-Reduce for various applications. However, the challenge is to find a scalable solution using Apache Spark for the optimization-based classification algorithms that would scale very well with large data.

In this dissertation, we firstly introduce new variants of a centroid-based particle swarm optimization (CPSO) classification algorithm in order to improve its performance in terms of misclassification rate. Furthermore, a scalable particle swarm optimization classification algorithm (SCPSO) is designed and implemented using Apache Spark. Two variants of SCPSO, namely SCPSO-F1 and SCPSO-F2, are proposed based on different fitness functions. The experiments revealed that SCPSO-F1 and SCPSO-F2 utilize the cluster of nodes efficiently and achieve good scalability results.

Moreover, we propose a cost-sensitive differential evolution classification algorithm to improve the performance of the differential evolution classification algorithm when applied to imbalanced data sets. The experimental results demonstrate that the proposed algorithm efficiently handles highly imbalanced binary data sets compared to the current variants of differential evolution classification algorithms.

Finally, we designed and implemented a parallel version of a cost-sensitive differential evolution classifier using the Spark framework. The experiments revealed that the proposed algorithm achieved good speedup and scaleup results and obtained good performance.

# ACKNOWLEDGEMENTS

First and foremost, all praise is due to Allah (God) alone. Thanks and appreciation to Allah for giving me the power to complete a Ph.D. degree and write up this dissertation.

I would like to express my thanks and appreciation to my advisor Professor Simone A. Ludwig for her encouraging support and guidance during my research journey. She helped and encouraged me as I pursued my degree. Without her valuable advice, I would not have achieved high-quality research and completed my Ph.D. degree. I do appreciate everything that she did for me, thanks a lot, Professor Ludwig. Additionally, I am grateful to the committee members; Professor Kendall Nygard, Professor Jun Kong, and Professor Ying Huang for their valuable comments and suggestions, thanks a lot, professors. I also would like to thank the computer science department faculty.

Finally, I would like to express undescribable thanks to my wife, Malaak, for her patience and unlimited support. I also would like to extend special thanks to my parents and family for their motivation and support.

# DEDICATION

To my grandmother's soul

To my father and mother

To my wife, Malaak and my son, Ahmad

To my brothers and sisters

To my nephews and nieces

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# 1. INTRODUCTION

This chapter starts with a description of the topics related to the research of this dissertation given in Section 1.1-1.3. Then, it discusses the motivation and the problem statement in Section 1.4. After that, our contributions are introduced in Section 1.5. Ultimately, the dissertation structure is presented in Section 1.6.

## 1.1. Data Classification

Data mining is a multidisciplinary research area which combines machine learning, statistics, and databases. The main goal of data mining is to analyze historical data by discovering hidden relationships and extracting valuable information in order to predict the future. The data mining task is either based on supervised learning or unsupervised learning. Supervised learning is a process of building and training a model using a sophisticated algorithm and a labeled data set that comprises of the input with their corresponding output. The aim is to discover hidden relationships between the input and the output that would help to predict an accurate outcome for an unseen data set. In contrast, unsupervised learning is a process of exploring hidden and interrelated structures between the input in an unlabeled data set to learn more about the data [1],[2].

Classification is a supervised learning task, which starts with a labeled data set that is split into a training and a testing data set. The training data set and a classification algorithm are used to build and train a model. After that, the model is applied on the testing data set, to analyze each input instance in the testing data set to predict an outcome. All outcomes of the model are compared with the actual outcomes to measure the model performance [1],[2]. In recent decades, classification applications were successfully applied in many domains, e.g., agricultural, engineering, biomedical, and finance.

Generally, classification problems can be categorized into three categories, binary, multi-class, and multi-label problems. In binary classification, each object belongs to one of two classes, e.g., a tumor could be benign or malignant, whereas in multi-class classification, a target outcome of an object could be one of several class labels, e.g., a vehicle could be a car, SUV, bus or truck. For multi-label classification problems, one or more class labels are assigned to an object, e.g, the text of a document might be relevant to one or more topics.

### 1.2. Nature-Inspired Optimization Algorithms

Nature-inspired algorithms belong to a paradigm that simulates the biological processes in nature. These algorithms were mainly proposed to solve the real optimization problems such as engineering optimization problems [3]. Nature-inspired algorithms have been successfully applied to optimize single-objective and multi-objective functions and solve NP-hard problems [3, 4, 5]. Furthermore, researchers successfully used these algorithms in the data mining field to solve data mining problems (classification or clustering) [6, 7] or to find optimal initial parameter values for classification algorithms [8]. Nature-inspired algorithms are mainly divided into two families; Swarm Intelligence Algorithms and Evolutionary Algorithms.

A swarm intelligence (SI) algorithm is a model that represents the social behavior of a group of individuals (usually insects) and how they interact with each other and with their environment to achieve a particular goal, e.g. finding the best food source. In the group of individuals, there is no leader to guide the individuals; each individual somehow behaves randomly within the search space and share information with its neighbors until the best source of food in a certain area is discovered [9, 10]. Since the 1990s, many of the SI methods have been proposed which are inspired by various biological systems, for instance, an ant colony optimization [11] is inspired by the foraging behavior of an ant colony, which imitates the behavior of ants seeking the optimal path between their nest and a food source. Similarly, particle swarm optimization [12] simulates the motion of bird flocking when they are searching for food in a certain area.

Evolutionary Algorithms (EAs) are stochastic optimization algorithms that imitate the biological evolution theory. EAs are based on the population which consists of individuals that represent a candidate solution. The idea of EAs is to find the best solution by applying the principle of the "survival of the fittest" to generate a new population. During the last decades, researchers proposed various algorithms based on this theory such as Differential Evolution (DE) [13].

#### 1.2.1. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a stochastic search method that was inspired by the social behavior of a flock of birds when they are looking for the best source of food in a certain area, which was introduced by Kennedy and Eberhart in 1995 [12]. In PSO, each particle represents a

potential solution within a solution space, which moves through a search space seeking the best solution. The direction of a particle's movement is affected by three factors:

- Inertia Weight ($w$) which controls how much the particle follows its current position.
- Cognitive Learning ($c_1$) which controls how much the particle follows the best position that it has found so far.
- Social Learning ($c_2$) which controls how much the particle follows the best position that was found by the entire swarm.

The pseudo code of the PSO algorithm is given in Algorithm 1.1. The PSO algorithm starts with a predefined number of particles, where each particle's position vector $p = \{p_1, p_2, p_3, \ldots, p_n\}$ is initialized randomly in an N-dimensional search space with a random velocity vector $v = \{v_1, v_2, v_3, \ldots, v_n\}$. At each iteration of PSO, a fitness function computes a fitness value for each particle by evaluating the particle's position to determine how close this particle is to achieve the goal (best solution). Based on the fitness value, the $PBest$ position, which is the best position that has been discovered so far by each particle, and the $GBest$ position, which is the best position that has been discovered so far by any particle of the entire swarm, are updated. The current velocity vector of each particle is updated using the $PBest$ and $GBest$ vectors as follows:

$$v_j^{(t+1)} = wv_j^t + r_1c_1\left(PBest - p_j^t\right) + r_2c_2\left(GBest - p_j^t\right) \tag{1.1}$$

Here, $p_j^t$ and $v_j^t$ are the current position vector and the current velocity vector of particle $j$ at iteration $t$, respectively; $r_1$ and $r_2$ are the random numbers between [0,1], $c_1$ and $c_2$ are constant coefficients for cognitive and social factors, respectively, which are specified by the user. The inertia weight value $w$ is computed using Equation 1.2 which starts from $w_{max}$, and then its value is linearly decremented as the number of iterations increase until $w_{min}$ is reached [14]. $v_j^{t+1}$ is a new velocity vector of particle $j$, where the new value at each dimension of the velocity vector is clamped within the range [$v_{min}$, $v_{max}$].

$$w(t) = w_{max} - \left((w_{max} - w_{min})\frac{t}{T_{max}}\right) \tag{1.2}$$

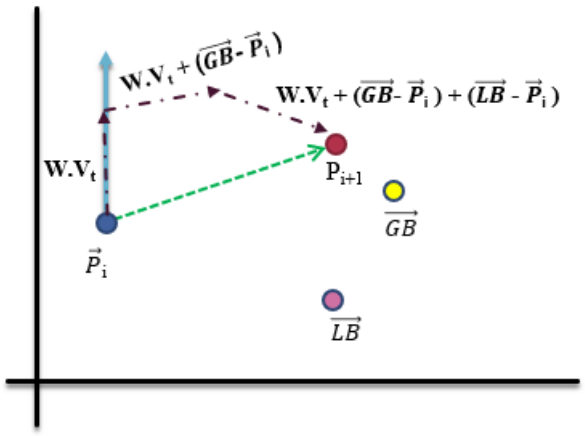



Figure 1.1. Illustration of Velocity and Position Updates

After the particle's velocity vector is updated, the new position of the particle is updated as follows:

$$p_j^{(t+1)} = p_j^t + v_j^{(t+1)} \tag{1.3}$$

where $p_j^t$ is the current position vector of particle $j$, $v_j^{(t+1)}$ is the new velocity vector of particle $j$, and $p_j^{(t+1)}$ is the new position vector of particle $j$. Figure 1.1 illustrates the velocity and position updates of particle $P_i$ at iteration $t$ in the two-dimensional space. All previous operations are repeated until a stopping criterion is satisfied.

### 1.2.2. Differential Evolution

Differential evolution (DE) is a simple and robust stochastic search method which belongs to the Evolutionary Algorithm (EA) family of optimization algorithms [15]. Storn and Price proposed DE for solving global optimization problems in continuous search spaces [13]. The idea of DE is mainly inspired by the Genetic algorithm (GA) with the difference in the way new offsprings are generated. Since then, DE has been applied to various real-world optimization problems and showed itself as an efficient and powerful technique for solving these problems [16],[17]. Furthermore, the researchers and contributors have proposed improved versions of the original DE [18],[19].

DE starts with a set of $NP$ individuals that form the population. Each individual is represented by a $d$-dimensional vector $\vec{x}_{i,G} = \{y_1, y_2, y_3, \ldots, y_d\}$, $i = \{1, 2, \ldots, NP\}$, which is randomly initialized in an $d$-dimensional problem space. Here, $G$ is the generation.

**Algorithm 1.1** PSO Algorithm

```
for each particle do
    Randomly initialize particle's position and velocity
end for
repeat
    for each particle do
        Compute fitness value (FV)
        if FV is better than FVPBest then
            FVPBest = FV
            Take current particle position as PBest
        end if
    end for
    Take the position of particle whose best FV value as GBest
    for each particle do
        Update particle velocity using Equation (1.1)
        Update particle position using Equation (1.3)
    end for
    Update the inertia weight using Equation (1.2)
until stopping criterion is satisfied
```

After initialization, the fitness of the individuals are evaluated using an objective function. Then, the current population in generation $G$ goes through the mutation, crossover, and selection operations to generate a new population in generation $G + 1$. In the mutation operation, the $DE/best/1/bin$ schema is used to create a mutant vector $\vec{m_{i,G}}$ for each target vector $\vec{x}_{i,G}$ as follows:

$$\vec{m}_{i,G} = \vec{x}_{best,G} + F.(\vec{x}_{r1,G} - \vec{x}_{r2,G}) \tag{1.4}$$

where *best* is the index of the best vector in the current population at generation $G$. $r1$ and $r2$ are random values chosen from $\{1, 2, \ldots, NP\}$, which should be mutually different and different from the values of the best vector and the target vector. The $F$ parameter is a scaling factor of the difference vector, which controls the evolution rate of the generation. The $F$ value is a random value within the range [0.0, 2.0].

After the mutant vectors are generated, the crossover operation is carried out to improve the diversity of the population. During the crossover operation, a trail vector $\vec{t_i}$ is created by combining the target vector $\vec{x}_i$ with its mutant vector $\vec{m}_i$ at the current generation $G$ as follows:

$$\vec{t}_{j,i,G} = \begin{cases} \vec{m}_{j,i,G} & if \quad (rand(0,1.0) \leq CR \quad or \quad rand_j == j) \\ \vec{x}_{j,i,G} & otherwise \end{cases} \tag{1.5}$$

Here, the crossover parameter $CR \in [0,1]$ represents the probability of inheriting an element from a mutant vector, which is defined by a user [20]. Besides, the condition $rand_j == j$ is added to ensure that at least one element is inherited from the mutant vector [20].

The outcome of the crossover operation is the trail vectors which are the candidate vectors for the next generation $(G+1)$. The target and trial vectors are evaluated using an objective function to measure their fitness level. After that, the "survival of the fittest" principle is applied to choose between the target vector $\vec{x}_{i,G}$ and its trail vector $\vec{t}_{i,G}$ using the following selection rule:

$$\vec{x}_{i,G+1} = \begin{cases} \vec{t}_{i,G} & if \quad (FL(\vec{t}_{i,G}) \quad is \quad better \quad than \quad FL(\vec{x}_{i,G})) \\ \vec{x}_{i,G} & otherwise \end{cases} \tag{1.6}$$

Here, $FL$ is the fitness. According to the selection rule given in Equation 1.6, the target vector $\vec{x}_{i,G}$ is replaced by the trial vector $\vec{t}_{i,G}$ in the next generation $(G+1)$, if the $\vec{t}_{i,G}$ vector has a fitness level better than the $\vec{x}_{i,G}$ vector. Otherwise, the $\vec{x}_{i,G}$ vector survives to the next generation $(G+1)$.

After the new population for the new generation $(G+1)$ is created, the mutation, crossover, and selection operations are repeated until the maximum number of generations is reached.

### 1.3. Apache Spark

Apache Spark is an in-memory computing framework for processing big data using a cluster of nodes, which was initiated and implemented by a research team from the University of California-Berkeley in 2009 [21]. Apache Spark addresses the major drawback of the Hadoop Map-Reduce framework [22] which is the overhead due to disk I/O operations that are needed to write an intermediate result in a shared file system during the running of a Map and Reduce job by distributing the data across a cluster of nodes and keeping it in memory as long as necessary. Moreover, Apache Spark overcomes the shortcoming of Hadoop MapReduce for handling iterative and interactive jobs [23].

The Resilient Distributed Dataset (RDD) is a basic component in Apache Spark, which is an immutable collection of objects distributed across a cluster of nodes in a fault-tolerant manner. Apache Spark provides two types of operations that can be performed on the RDD, which are a transformation and an action. The transformation operation is used to create a new RDD from an existing RDD using pre-defined functions such as Map and MapToPair. On the contrary, the action operation is used to compute a result by running a computation job on the RDD such as reduce and count, and the result can be returned to the driver program or written on a shared file system. The transformation operations on the RDD are executed lazily, which means that the transformations on the RDD are only executed when a Spark application triggeres an action operation on the RDD. This effectively avoids repeated evaluations. Furthermore, fault-tolerance of Spark applications is achieved by keeping track of the "lineage" of each RDD, which represent the sequence of operations that produced it. This allows the reconstruction of the operations and data flow in the case of data loss [23, 24, 25, 26].

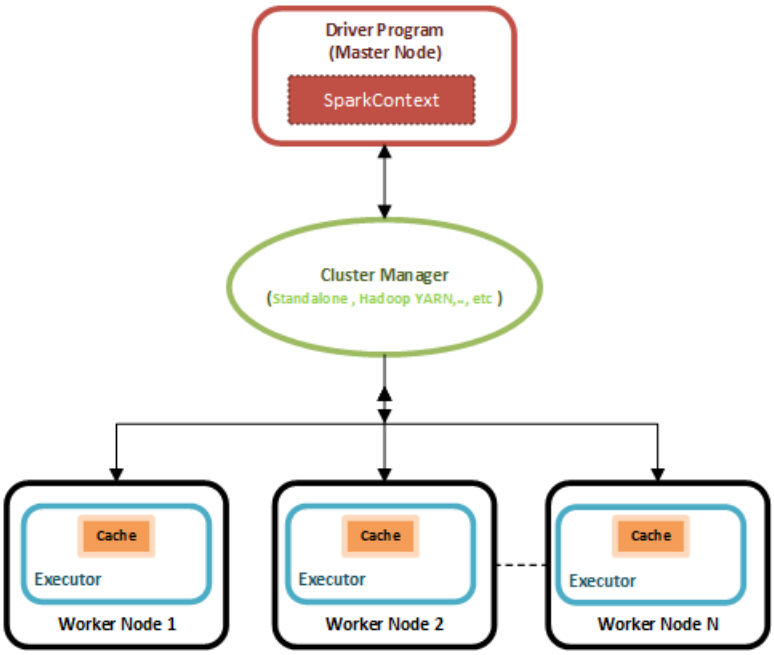



Figure 1.2. Spark Architecture

As shown in Figure 1.2, the Apache Spark architecture consists of two layers and a cluster manager. The spark application starts by creating one driver program in the master node and

launches a spark context object within it. The driver program creates a logical plan for the RDD operations as a directed acyclic graph (DAG). The aim of the DAG is the recovery of the RDDs in case of a worker node failure or any RDD loss during the running of the application. Next, the DAG is converted using pipelining transformations to a physical plan, which is a set of stages, where each stage comprises of a set of tasks. After the physical plan is created, the driver program asks the cluster manager through the spark context to allocate the resources (Memory, CPU) and run executors on the worker nodes. At this stage, the cluster manager registers the executors to the driver program. During the running of the spark application, the driver program monitors the executors and sends the tasks to the executors to run in multi-thread mode. The spark application keeps running until the spark context's stop method is invoked or the main function of the application is finished [23, 24, 25, 26].

Although Apache Spark does not support a shared global memory between a driver program and the tasks that are running on the worker nodes, it provides two types of shared variables, Broadcast and Accumulator. The broadcast variable is a read-only variable that is created by the driver program and is cashed on the memory of executors, which is used by the executors during the task execution. The Accumulator variable is a write-only variable, which is used to aggregate values from the executors in the driver program [23, 24, 25, 26].

### 1.4. Motivation and Problem Statement

In the last decade, researchers proposed many classification algorithms based on different nature-inspired algorithms to tackle the classification task [27, 28, 29, 30, 31]. Among the nature-inspired algorithms, Particle Swarm Optimization (PSO) and Differential Evolution (DE) have been successfully applied to data classification. The idea behind these algorithms is finding the optimal centroid of all labels in a data set. Then, assigning an unseen input to the closest centroid [27, 30, 31].

The PSO and DE algorithms showed that they are robust and efficient to do data classification [27, 30, 31]. However, these algorithms lack scalability with increasing data sizes. Furthermore, these algorithms need more memory space and computational time when applied to large data.

Moreover, the nature of data in various real-life applications such as Biomedical, Intrusion Detection System, and Credit Card Fraud, is typically imbalanced where the number of instances that belong to one class label (minority class) is significantly lower compared to another class label

(majority class). Furthermore, in some circumstances, the misclassification cost of the minority class is much larger than the misclassification cost of another class [32]. For example, in an intrusion detection system, undetected attacks are much more serious and costly than detecting normal behavior as an attack. The PSO and DE classification algorithms have shown ineffective to cope with imbalanced data.

To overcome the above drawbacks, these algorithms need to be developed by designing a scalable solution using big data frameworks such as Hadoop MapReduce and Apache Spark. Furthermore, finding suitable fitness functions to handle the imbalanced data is of importance.

Among the big data frameworks that have been introduced in the last decades, Apache Spark is more efficient and approximately 10 to 100 times faster for certain data processing on clusters of nodes compared to the Hadoop MapReduce framework. Moreover, Apache Spark provides rich APIs in several languages (Java, Scala, Python, and R) for developers to perform complex operations on distributed RDDs.

In this dissertation, our motivation is to take advantage of Apache Spark's ability for in-memory processing of data on a cluster of nodes to develop a scalable design of the PSO and DE classification algorithm. The aim is to help the scientific community to perform the classification task on large data. The parallelization capability of the scalable design is evaluated by applying this design to real-world applications. Furthermore, we improve the performance of nature-inspired classification algorithms when applied to imbalanced binary data by proposing a new fitness function.

### 1.5. Contributions

This dissertation makes various contributions towards enhancing the performance of the Particle Swarm Optimization (PSO) and the Differential Evolution (DE) classification algorithm. Additionally, a scalable solution for these algorithms was designed using Apache Spark framework. The contributions are as follows:

1. A new approach for data classification based on PSO is proposed to extend the existing work in [27] with aims to improve the performance of the PSO-based classification algorithm. The idea is to find the optimal centroid using PSO for each target label. Then, the optimal centroid is used to compute the standard deviation for each target label. After that, the normal

distribution probability density function and the probability of each target label are used to classify unseen inputs. Three versions were proposed based on different fitness functions, which were tested using ten data sets in order to measure the performance of these versions in terms of the misclassification error.

2. A parallel PSO-based classification algorithm is designed and implemented using the Apache Spark framework. The aim is to show how the PSO classification takes advantage of Apache Spark to work on large data sets to achieve high accuracy and scalability. The proposed approach uses the PSO algorithm to find the optimal centroid for each target label. Then, each instance in a test data set is assigned to the closest centroid according to the Euclidean distance. Two versions have been proposed based on different fitness functions, which have been analyzed using real large data sets to assess its performance and scalability.

3. A new variant of a differential evolution classification algorithm is proposed to cope with imbalanced binary data sets. In this work, we introduce a new objective function by minimizing the misclassification cost instead of the misclassification error to address the inefficient performance of the current variants of differential evolution classification algorithm when applied to imbalanced binary data sets. The proposed algorithm is applied to several cancer data sets to analyze and investigate the capability of the proposed algorithm for predicting the survivability of cancer patients compared to the performance of the current variants of differential evolution classification algorithm and five cost-sensitive machine learning algorithms.

4. A parallel version of a cost-sensitive differential evolution classification algorithm (Chapter 4) based on Apache Spark is introduced. The idea of our proposed algorithm is to find the optimal centroid of each class label in a training data set by minimizing the total misclassification cost. The aim is to handle imbalanced binary data sets and take advantage of Apache Spark to work on massive data sets to achieve high performance and scalability.

### 1.6. Dissertation Overview

This dissertation is divided into chapters. Each chapter is derived from a paper that was published or submitted based on research done during the PhD study.

The remaining chapters of this dissertation are:

- In **Chapter 2**, a centroid-based particle swarm optimization variant for data classification is introduced, which is derived from the publication:
  *Jamil Al-Sawwa and Simone A. Ludwig. Centroid-Based Particle Swarm Optimization Variant for Data Classification. In 2018 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 672-679). November 2018.*

- In **Chapter 3**, a parallel particle swarm optimization classification algorithm variant implemented with Apache Spark is discussed, which is derived from the publication:
  *Jamil Al-Sawwa and Simone A. Ludwig. Parallel Particle Swarm Optimization Classification Algorithm Variant implemented with Apache Spark. Journal of Concurrency and Computation: Practice and Experience, Wiley, accepted June 13, 2019.*

- In **Chapter 4**, a cost-sensitive centroid-based differential evolution classification algorithm applied to cancer data sets is discussed, which is derived from the publication:
  *Jamil Al-Sawwa and Simone A. Ludwig, A Cost-Sensitive Centroid-based Differential Evolution Classification Algorithm applied to Cancer Data Sets. 2019 IEEE Symposium Series on Computational Intelligence, submitted June 15, 2019.*

- In **Chapter 5**, spark-based cost-sensitive differential evolution classification algorithm is discussed, which is derived from the publication:
  *Jamil Al-Sawwa and Simone A. Ludwig, Performance Evaluation of a Cost-Sensitive Differential Evolution Classifier Using Spark. Journal of Computational Science, Elsevier, submitted April 2019.*

- **Chapter 6** presents the dissertation summary and the future research directions.

# 2. CENTROID-BASED PARTICLE SWARM OPTIMIZATION VARIANT FOR DATA CLASSIFICATION

Recently, data mining has become more attractive for researchers as a technique to analyze and transform raw data into useful information that would help with decision support. Over the last decade, many data mining applications have been proposed in various research areas such as medicine, agriculture, and finance. Particle swarm optimization (PSO) is one of the most popular swarm intelligence methods that simulates the behavior of bird flocking whereby the best source of food in a certain area is sought. In this chapter, a new approach for data classification based on PSO abbreviated as CPSO is proposed. The main idea of CPSO is to find the optimal centroid and the standard deviation for each target label and then use the normal distribution probability density function and the probability of each target label to classify unseen data. The performance of CPSO was tested using ten data sets and was compared to twelve classification algorithms. The experimental results show that the CPSO algorithm is competitive compared to other classification algorithms. In addition, the algorithm can be efficiently used for data classification.

The rest of this chapter is organized as follows. Section 2.1 presents the related work in data mining using swarm intelligence methods. Section 2.2 presents the existing approach to solve data mining tasks. Section 2.3 presents our proposed approach. Section 2.4 describes the data sets that are used in the experiments as well as the preprocessing task performed on these data sets. Section 2.5 presents the experiments and the results. Section 2.6 concludes our work.

## 2.1. Related Work

In last decades, researchers successfully used swarm intelligence methods in the data mining area to solve classification and clustering problems. In this section, we review existing approaches and techniques of using swarm intelligence methods to solve data mining problems.

The authors in [27] proposed three versions of a PSO-based classification algorithm according to the fitness function used to evaluate the particles. The idea of their work is to find the optimal centroids for each class label in an N-dimensional search space and then assign each data instance in a testing data set to the closest centroid. The fitness function of the first version computes the

percentage of misclassifications on a training data set after each data instance is assigned to the closest class centroid. In the second version, the fitness function computes the sum of all training data based on the Euclidean distance between the centroid of class label $c_j$ and the data instance that belongs to class label $c_j$ according to the training data set. For the last version, the fitness function is a linear combination of the fitness functions. The performance of three versions of PSO were validated using thirteen benchmark data sets and compared to nine well-known classification algorithms. From the experimental results, the authors deduced that the third version of PSO outperformed the other two versions as well as five out of nine classification algorithms in terms of the classification error rate.

In [33], the authors introduced two algorithms, a gbest PSO and a HybridPSO, for data clustering. The idea of the gbest PSO is to use the basic PSO algorithm to find the optimal centroids for a predefined number of clusters, while in the HybridPSO, the authors used the result of the k-means algorithm as the initial position of one of the particles before running the gbest PSO algorithm. According to the inter-cluster distance, intra-cluster distance, and quantization errors, the authors concluded that the performance of the gbest PSO and HybridPSO algorithms are comparable to the k-means clustering performance.

The PSO clustering algorithm was efficiently used to solve an image classification problem in [34]. In this work, the PSO clustering algorithm was applied to MRI (magnetic resonance imaging) and satellite images. The experimental results showed that the performance of the PSO algorithm outperforms the performance of the four clustering algorithms which are K-means, Fuzzy C-means, K-Harmonic means and Genetic algorithms. Another work was proposed in [35] for document clustering using the PSO clustering algorithm. The authors applied the PSO and Hybrid PSO algorithms on four different document data sets and compared the results of both algorithms with the k-means algorithm's result. The authors concluded that the hybrid PSO is better than PSO and k-means since it is able to generate higher compact clusters.

In [36], the authors used a PSO-based clustering algorithm to investigate the performance among four types of clusters, validity index, Euclidean distance based PBM index, the kernel function induced measure, the point symmetry distance-based index, and the manifold distance induced index. According to that, four versions of PSO clustering were proposed based on the validity index that is used to compute the fitness of a particle. Comprehensive experiments were

performed on real and synthetic data sets to evaluate the performance of each version. The results revealed that the PSO clustering, which uses the manifold distance induced index as the fitness function achieved better accuracy and robustness than the other versions.

Another work for data clustering can be found in [37]. In this work, an algorithm (CGSO) is proposed for data clustering using a glowworm swarm optimization approach. Glowworm swarm optimization is one of the newer swarm intelligence methods that simulates the behavior of the lighting worms. Three variants of the CGSO algorithm were proposed based on three fitness functions to achieve high-quality clustering. The performance of the three variants of CGSO were verified using seven data sets and were compared with the performance of other clustering algorithms. In terms of purity and entropy of the clustering, the first variant of CGSO obtained the best result compared to the other versions and other clustering algorithms.

A new approach for data classification using BCO is proposed in [28]. The BCO method was introduced to imitate the foraging behavior of honey bees [38]. The main idea of the work is to find the optimal centroid of each class label by minimizing the sum of all training data based on the Euclidean distance between the centroid of class label $c_j$ and the data instance that belongs to $c_j$. The experimental results showed that the proposed approach can efficiently be used for data classification.

Another work in classification using BCO is [39]. The authors presented a bee-colony based classification rule algorithm (ABC-Miner) to discover and extract classification rules from data sets, where each rule consists of an antecedent and consequent clause. Using three benchmark data sets, they concluded that the average accuracy of ABC-miner and the average number of extracted rules using ABC-miner are comparable and competitive with the C4.5 decision tree and PSO algorithms.

Ant colony optimization (ACO) is one of the SI techniques that imitates the behavior of ants seeking the optimal path between their nest and a food source, which was proposed to mainly solve shortest path type of problems [11]. In [40], a new algorithm for data classification using ACO named Ant-miner was proposed. The goal of Ant-miner is to extract simple rules from the data set in the IF-THEN form. The authors performed an experiment on six data sets to evaluate the performance of Ant-miner. From the experimental results, the authors observed that the rules extracted by the Ant-miner are simpler than those extracted by a CN2 [41] classifier (rule-induction classifier). In addition, the overall accuracy of Ant-miner is better than the CN2 classifier. Other

ACO approaches in data mining are applied to [42] are data clustering, [43] features selection, [44] web page classification, and [45] data classification.

In [46], a PSO-based classifier (FCM) was proposed based on c-mean fuzzy clustering, which consist of two phases, unsupervised clustering and supervised classification. In this work, PSO was used to optimize the centroid of the clusters and the parameters of the membership function. The FCM performance was evaluated using eight data sets and outperformed well-known classifiers such as support vector machine and k-nearest neighbors in terms of classification error rates.

Fuzzy c-means clustering is one of the fuzzy clustering algorithms, which divides the objects into groups where the intra-distance among objects within the group is minimized while the inter-distance among objects that belong to a different group is maximized. However, fuzzy c-means clustering suffers from two drawbacks; the first one is that the number of clusters must be predefined before the algorithm is run, and the second drawback is that most of the objects in overlapping areas are incorrectly assigned. In [47], a new approach using PSO to overcome the aforementioned drawbacks is proposed. The main goal of that approach is to automatically find the optimal number of clusters. The approach was tested using synthetic and real data sets and the experimental results revealed that the approach can correctly identify the number of clusters, however, it needs more than one run to achieve that.

### 2.2. Existing Approach

In [27], the authors proposed a new approach for data classification using PSO. In their work, each particle flies through an N-dimensional space to find the optimal centroids for all target classes $\{c_1, c_2, \ldots, c_i\}$ in a data set. Each particle $j$'s position and velocity are encoded as a vector as follows:

$$\vec{p_j} = \{p_j^{c_1}, p_j^{c_2}, \ldots, p_j^{c_i}\} \tag{2.1}$$

$$\vec{v_j} = \{v_j^{c_1}, v_j^{c_2}, \ldots, v_j^{c_i}\} \tag{2.2}$$

where $p_j^{c_i}$ and $v_j^{c_i}$ are the position and the velocity vector for particle $j$'s class label $c_i$, respectively, which are represented in an N-dimensional space as follows:

$$p_j^{c_i} = \{d_1, d_2, \ldots, d_n\} \tag{2.3}$$

$$v_j^{c_i} = \{v_1, v_2, \ldots, v_n\} \tag{2.4}$$

where $d_n$ and $v_n$ are the real values of position and velocity at dimension $n$, respectively.

Furthermore, each particle contains the following attributes:

- Current Fitness Value ($FV$)
- Best Centroids Vector that has been found so far during the journey of the particle ($PBest$)
- Best fitness value that has been found so far during the journey of the particle ($FV\_PBest$)

In addition, the entire swarm keeps track of the best centroid vector ($GBest$) that has been found so far by any particle and the best group fitness value found so far by any particle.

For the particle's fitness evaluation, three fitness functions are used to evaluate the fitness of each particle. The first fitness function ($F1$) computes the fitness value of each particle in two steps. In the first step, each data instance in a training data set is assigned a class label whose position is closest to that data instance. In the second step, the function $F1$ computes the misclassification rate. In mathematical terms, the fitness value of particle $j$ is calculated as follows:

$$F1\,(j) = \frac{1}{D_T} \sum_{i=1}^{D_T} \delta(\vec{x}_i) \tag{2.5}$$

$$\delta(\vec{x}_i) = \begin{cases} 1 & if c_{predicted}(\vec{x}_i) \neq c_{actual}(\vec{x}_i) \\ 0 & otherwise \end{cases} \tag{2.6}$$

where $D_T$ is the number of data instances in a training data set, $c_{actual}$ is the actual outcome of $\vec{x}_i$, and $c_{predicted}$ is the predicted outcome of $\vec{x}_i$.

For the second fitness function ($F2$), the fitness value of each particle is computed by taking the average of the sum of all Euclidean distances (Equation 2.8) between the current position of class label $c_i$ and the data instances that belongs to class label $c_i$ according to the training data set. In mathematical terms, the fitness value of particle $j$ is given by:

$$F2(j) = \frac{1}{D_T} \sum_{i=1}^{D_T} d\,(\vec{x}_i, \vec{p_j}^{c_i}) \tag{2.7}$$

where $D_T$ is the number of data instances in a training data set, $\vec{x}_j$ is a data instance vector that belongs to class label $c_i$ according to a training data set, and $\vec{p_j}^{c_i}$ is the current centroid vector for class label $c_i$.

$$d\left(\vec{x}, \vec{y}\right) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2} \tag{2.8}$$

The last fitness function $F3$ is a linear combination of $F1$ and $F2$. The fitness value of particle $j$ is computed as follow:

$$F3(j) = \frac{F1(j) + F2(j)}{2} \tag{2.9}$$

The algorithm starts by initializing a predefined number of particles randomly in a search space. At each iteration of CPSO, the fitness value of each particle is computed using one of the above fitness functions. After computing the fitness value and finding $PBest$ and $GBest$, the particle updates its current velocity vector and current centroid vector using Equation 1.1 and Equation 1.3, respectively.

The above operations are repeated until a stopping criterion is met. Finally, the optimal centroid vector of each class label is used to classify unseen data based on distance.

### 2.3. Proposed Approach

In this work, we propose a new approach that extends the existing work to improve the performance of the classification model. The new approach is further referred to as CPSO. In CPSO, the optimal centroid vector of a class label that has been found by PSO, is used to compute the class label's standard deviation vector, which is the average of the squared differences between the class label's centroid vector and the data instances that belong to that class label. The standard deviation at dimension $i$ is calculated as follows:

$$\sigma_i^{c_i} = \sqrt{\frac{1}{N} \sum_{j=1}^{N} \left(x_{ji}^{c_i} - p_i^{c_i}\right)^2} \tag{2.10}$$

where $\sigma_i^{c_i}$ is the standard deviation of class label $c_i$ at dimension $i$, $p_i^{c_i}$ is the value of dimension $i$ for class label $c_i$'s centroid, and $x_{ji}^{c_i}$ is the value for the $i$ dimension for the data instance j with class label $c_i$.

Furthermore, the probability ($P$) of each class label is calculated by the following equation:

$$P(c_i) = \frac{\#\ instances\ belonging\ to\ class\ label\ c_i}{\#records} \tag{2.11}$$

where $P(c_i)$ is the probability of class label $c_i$, and $\#records$ is the total number of instances in the data set.

The outcome of the previous operations is a CPSO model which is applied on a testing data set. The CPSO model assigns each data instance to the class label $c_i$ based on the best value of the CLASSIFY function (Equation 2.13).

$$dnorm\left(x_i, p_i, \sigma_i\right) = \frac{1}{\sigma_i\sqrt{2\pi}} \cdot e^{-(x_i-p_i)^2/2{\sigma_i}^2} \tag{2.12}$$

$$CLASSIFY\left(c_i, \vec{x}, p^{\vec{c_i}}, \sigma^{\vec{c_i}}\right) = P\left(c_i\right) \cdot \prod_{i=1}^{D} dnorm\left(x_i, p_i^{c_i}, \sigma_i^{c_i}\right) \tag{2.13}$$

In the CLASSIFY function, $x_i$ is the value of dimension $i$ of data instance $x$ and $p_i^{c_i}$, $\sigma_i^{c_i}$ are the values of dimension $i$ in the centroid vector $p^{\vec{c_i}}$ and the standard deviation vector $\sigma^{\vec{c_i}}$, respectively, for class label $c_i$.

### 2.4. Data Sets and Preprocessing

Two types of data sets, binary data sets and multi-class data sets are used to evaluate the performance of the classification algorithms. In this section, we present the data sets and its preprocessing.

#### 2.4.1. Data Sets

The two types of data sets were taken from the UCI Machine Learning Repository [48], which is one of the most popular database repository to evaluate machine learning algorithms. Table 2.1 shows the data sets and their characteristics: the total number of instances ($I$), the number of features ($D$), the type of features ($T$), the number of class labels ($C$), and whether it is sorted by class label or not ($L$). The following is a brief description of these data sets:

- **Cancer-Int:** contains the diagnosis of breast cancer of patients with two outcomes: benign or malignant. It contains 699 samples and 9 features.

- **Credit:** is a mixed type data set for estimating a customer's credit card application based on 14 features: 6 real features and 8 discrete features. The instances in the data set are classified into 2 class labels: approval (+) and disapproval (-).

- **Diabetes:** was collected by the National Institute of Diabetes and Digestive and Kidney Diseases. The data samples show whether a patient has diabetes or not based on diagnostic measurements included in the data set.

- **Heart-Satlog:** contains samples of 270 patients, where each sample has 13 features that are used to predict whether heart disease exists or not for each patient.

- **Hepatitis:** contains data of patients who have the hepatitis disease and whether they are alive or died.

- **Balance:** is the result of a model psychological experiment. It has 625 instances that are classified into three class labels: tip to the right ($R$), tip to the left ($L$), and balance ($B$) based on four features included in the data set.

- **Iris:** contains 150 samples for three of the iris flower species: versicolor, setosa, and virginica.

- **Thyroid:** contains 5 features that are used to predict a patient's thyroid condition whether they have hypothyroidism, hyperthyroidism, or a normal thyroid.

- **Wine:** contains 13 features of a chemical analysis result of three wine cultivators that were grown in the same region in Italy.

- **E. Coli:** contains 8 features beside the class label and 327 instances that are distributed into eight class labels, but three of the eight class labels represent only up to 5 instances in the data set, so we removed them from the data set.

### 2.4.2. Data Preprocessing

Data preprocessing is one of the data mining processing steps that prepares a raw data set for further processing. In this stage, all data sets in Table 2.1 are normalized using Min-Max

Table 2.1. Properties of Data Sets

| Data set | I | T | D | C | L |
|---|---|---|---|---|---|
| Cancer-Int | 699 | Real | 9 | 2 | No |
| Credit | 690 | Categorical, Integer, Real | 14 | 2 | No |
| Diabetes | 768 | Integer, Real | 8 | 2 | No |
| Heart-Satlog | 270 | Categorical, Real | 13 | 2 | No |
| Hepatitis | 155 | Categorical, Integer, Real | 19 | 2 | Yes |
| Balance | 625 | Categorical | 4 | 3 | No |
| Iris | 150 | Real | 4 | 3 | Yes |
| Thyroid | 215 | Categorical, Real | 5 | 3 | Yes |
| Wine | 178 | Integer, Real | 13 | 3 | Yes |
| E. Coli | 327 | Real | 5 | 5 | Yes |

normalization to rescale the features to the range [0,1]. In addition, we used the "ReplaceMissingValues" filter in the WEKA tool [49, 50] to replace the missing values in some data sets.

We have also noticed that the instances of some of the data sets are sorted by class label, so we decided to shuffle them to make sure that all class labels appear in both the training and testing phases.

### 2.5. Experiments and Results

In our work, we performed three experiments to evaluate the performance of CPSO compared to the performance of other classification algorithms. The validation method for the evaluation of the classifiers in three experiments are different. In the first and second experiments, we used K-fold cross-validation. For this validation method, the data set is split into $k$ number of approximately equal sized subsets. Next, each subset is used once to test a model and the remaining $k-1$ sub sets are used to train the model; this process is repeated $k$ times. In the second experiment, we used the same validation method as described in [27], which is the Hold-Out method.

For the three experiments, the $MCR$ measure is used to evaluate the performance of the classifiers, which is computed as follows:

$$MCR \;=\; 100 \;\times\; \frac{\#\; incorrectly\; classified\; instances}{size\; of\; testing\; data\; set} \tag{2.14}$$

In this section, we present the three experiments as well as the results.

Table 2.2. Misclassification Rate (MCR) for Algorithms on All Data Sets

| | Cancer-Int | Credit | Diabetes | Heart-Satlog | Hepatitis | Balance | Iris | Thyroid | Wine | E. Coli |
|---|---|---|---|---|---|---|---|---|---|---|
| **CPSO-F2** | 3.91 | 17.00 | 25.78 | 16.04 | 16.13 | 9.65 | 4.66 | **3.56** | 2.80 | **11.52** |
| **BayesNet** | **2.72** | 13.91 | 25.65 | 18.89 | 17.42 | 8.48 | *7.33* | 4.65 | 1.68 | 14.06 |
| **SMO** | 3.00 | 14.78 | **22.66** | **15.93** | **14.84** | 10.08 | 4.00 | *11.16* | **0.56** | 14.37 |
| **MLP** | 4.58 | 16.96 | 24.61 | 21.85 | 15.48 | **0.48** | **3.33** | 6.04 | 3.37 | 12.84 |
| **KStar** | 4.58 | *21.16* | 30.86 | *24.81* | 18.71 | 12.16 | 5.33 | 5.11 | 1.12 | 16.51 |
| **IBK** | 4.72 | 17.97 | 29.82 | *24.81* | 19.35 | 15.36 | 4.66 | 3.72 | 4.49 | 17.73 |
| **Bagging** | 4.00 | 14.35 | 24.61 | 18.89 | 16.77 | 23.20 | 4.66 | 6.04 | *6.74* | 15.60 |
| **VFI** | 4.29 | 13.91 | *36.07* | 20.00 | **14.84** | 16.00 | 4.00 | 10.70 | 2.80 | *19.27* |
| **Ridor** | *5.44* | 15.51 | 25.00 | 21.85 | *23.23* | 31.04 | 5.33 | 9.30 | 6.18 | 16.82 |
| **NBTree** | 3.72 | 14.49 | 26.43 | 19.63 | 18.71 | 8.48 | *7.33* | 5.58 | 2.24 | 17.74 |
| **J48** | 4.86 | 14.93 | 26.17 | 23.33 | 19.35 | *37.12* | 6.00 | 6.98 | 6.18 | 15.90 |
| **RandomForest** | 3.58 | **12.46** | 25.65 | 18.14 | 16.13 | 20.64 | 6.00 | 5.58 | 1.68 | 12.84 |

Table 2.3. Ranking of Algorithms

| | MLP | CPSO-F2 | SMO | BayesNet | RandomForest | NBTree | Bagging | KStar | VFI | IBK | Ridor | J48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Average** | 10.95 | 11.11 | 11.14 | 11.48 | 12.27 | 12.44 | 13.49 | 14.04 | 14.19 | 14.26 | 15.97 | 16.08 |
| **Rank** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

Table 2.4. Misclassification Rate (MCR) for Three Versions of CPSO Based on Fitness Functions

| **Data set** | **CPSO-F1** | **CPSO-F2** | **CPSO-F3** |
|---|---|---|---|
| Cancer-Int | 4.58 | **3.91** | 4.00 |
| Credit | **17.00** | **17.00** | 17.72 |
| Diabetes | 30.64 | **25.78** | 26.22 |
| Heart-Satlog | 22.83 | **16.04** | 21.11 |
| Hepatitis | 17.42 | 16.13 | **15.05** |
| Balance | 16.26 | **9.65** | 11.68 |
| Iris | 6.22 | **4.66** | 5.33 |
| Thyroid | 15.81 | **3.56** | 7.90 |
| Wine | 4.11 | 2.80 | **2.24** |
| E. Coli | 19.57 | **11.52** | 17.94 |
| *Average* | *15.44* | *11.11* | *12.92* |

### 2.5.1. First Experiment

To evaluate the impact of the three fitness functions on the performance of the CPSO algorithm, we ran the three variants of CPSO (CPSO-F1, CPSO-F2, and CPSO-F3) on all data sets as described in Table 2.1 with 10-fold cross validation and compared the performance of these variants in terms of MCR. The parameters for the CPSO classifier are the same as in [27], which are:

- maximum number of iterations = 100
- acceleration coefficient constants $c_1$ and $c_2$ = 2.0
- swarm size = 50
- velocity range [$v_{min}$ = -0.05, $v_{max}$ = 0.05]
- weight inertia range [$w_{min}$ = 0.4, $w_{max}$ = 0.9]

Table 2.4 shows the average $MCR$ results of CPSO that are achieved by each version of CPSO for each data set using 10-fold cross validation. From the results in Table 2.4 we can easily see that the performance of CPSO-F2 outperforms the performance of CPSO-F1 and CPSO-F3 in 7 out of 10 data sets in terms of $MCR$, which achieved the best value on the Cancer-Int, Diabetes, Heart-Satlog, Balance, Iris, Thyroid, and E. Coli data sets, where the $MCR$ results were 3.91%,

25.78%, 16.04%, 9.65%, 4.66%, 3.56%, and 11.52%, respectively. For the Wine and Hepatitis data sets, CPSO-F3 obtained the best $MCR$ with results of 2.24% and 15.05%, respectively. For the Credit data set, the CPSO-F1 and CPSO-F2 variants achieved the best $MCR$ of 17.00%.

Furthermore, the $MCR$ results of each version are averaged over the 10 data sets, as shown in the last row of Table 2.4. From the average $MCR$ result, we can conclude that CPSO-F2 achieved the best performance compared to CPSO-F1 and CPSO-F3.

### 2.5.2. Second Experiment

In this experiment, we compared the performance of the best version of CPSO, which is CPSO-F2, with the performance of eleven well-known classification algorithms. We used the Waikato Environment for Knowledge Analysis (WEKA) tool version 3.6 [49, 50], which contains numerous implemented classification algorithms in order to execute the eleven well-known classification algorithms. From different groups of classification algorithms in WEKA, we chose the following classifiers: BayesNet from package Bayes, Multilayer Perceptron Network (MLP) and Support Vector Machine (SMO) from package Functions, K-nearest-neighbor (IBK) and KStar from package Lazy, Bagging from package Meta, Ridor from package Rules, VFI from package Misc, and RandomForest and NBTree from package Trees. The default setting parameters in WEKA were used for the twelve classification algorithms.

Table 2.2 shows the misclassification rate ($MCR$) obtained by each classifier with 10-fold cross-validation for each data set. The best values of the misclassification rate obtained among all classifiers is marked in bold while the worst value is marked in red italics.

At a glance, we can easily see that the CPSO classifier obtained the best $MCR$ for the Thyroid and E. Coli data sets, where the $MCR$ results were 3.56% and 11.52%, respectively. The Support Vector Machine (SMO) is the only classifier that outperformed CPSO, which obtained the best $MCR$ in 4 out of 10 data sets, where the $MCR$ results were 22.66%, 15.93%, 14.84%, and 0.56% for Diabetes, Heart-Satlog, Hepatitis, and Wine data sets, respectively. For the other classifiers, the Multilayer Perceptron Network (MLP), Bayes Net, RandomForest, and VFI obtained the best $MCR$ in 2,1,1,1 data sets, respectively.

In addition to that, another significant conclusion that can be drawn from the results is that the CPSO classifier never obtained the worst $MCR$ on any of the data sets.

To summarize the results in Table 2.2, the $MCR$ results of each classifier are averaged over the 10 data sets, then the classifiers are ranked based on the averaged values as shown in Table 2.3. From this we can see that the CPSO-F2 classifier is ranking second with an average $MCR$ of 11.11%, which is quite close to the best classifier (MLP).

### 2.5.3. Third Experiment

In this experiment, we used Hold-Out validation as described in [27] applied to the data sets to compare the performance of the CPSO-F2 classifier with the performance of the best two versions of PSO (version 2 and 3) as given in [27]. In the Hold-Out validation method, a data set is split into two sub data sets; 75% of the instances is used for training and the remaining for testing. The parameters for the CPSO-F2 classifier are the same as in [27], which are:

- maximum number of iterations = 1000
- acceleration coefficient constants $c_1$ and $c_2 = 2.0$
- swarm size = 50
- velocity range [$v_{min}$ = -0.05, $v_{max}$ = 0.05]
- weight inertia range [$w_{min} = 0.4$, $w_{max} = 0.9$]

Table 2.5 shows the $MCR$ obtained by CPSO-F2, PSO-$\psi_2$, and PSO-$\psi_3$ for each data set as well as the $MCR$ average over all data sets. The first conclusion that can be drawn from the results, is that CPSO-F2 outperforms the performance of PSO-$\psi_3$ and PSO-$\psi_2$ in 6 out of 7 data sets in terms of $MCR$, whereas PSO-$\psi_3$ obtained a somewhat better $MCR$ result for the Diabetes data set. For the Wine data set, CPSO-F2 and PSO-$\psi_2$ obtained the same $MCR$. The second conclusion is interesting since CPSO-F2 obtains improved $MCR$ values averaged over all data sets by a factor of 1.75 and 1.5 compared to the average $MCR$ of PSO-$\psi_2$, PSO-$\psi_3$, respectively.

## 2.6. Summary

In this chapter, we proposed the CPSO algorithm for data classification. CPSO uses the basic PSO algorithm to find the optimal centroid for each target class and then computes the standard deviation based on that centroid. Next, the normal distribution density function (PDF) and the probability of each target class is used to classify each instance in a testing data set.

Table 2.5. Misclassification Rate (MCR) for CPSO-F2, PSO-$\psi_2$, and PSO-$\psi_3$

| **Data set** | **CPSO-F2** | **PSO-$\psi_2$ [27]** | **PSO-$\psi_3$ [27]** |
|---|---|---|---|
| Balance | **8.97** | 25.47 | 13.12 |
| Cancer-Int | **1.71** | 2.87 | 2.64 |
| Diabetes | 22.92 | 22.50 | **21.77** |
| Iris | **0.00** | 2.63 | 5.26 |
| E. Coli | **7.31** | 14.63 | 13.90 |
| Wine | **2.22** | **2.22** | 2.88 |
| Thyroid | **0.00** | 5.55 | 3.88 |
| *Average* | *6.16* | *10.84* | *9.06* |

Three experiments were performed to evaluate the performance of CPSO starting by evaluating the performance of CPSO based on the fitness functions and then compared the performance of the best version of CPSO to the performance of the other classification algorithms in terms of $MCR$. The first experiment results showed that the performance of CPSO-F2 outperformed the performance of the other versions of CPSO (CPSO-F1 and CPSO-F3). In the second experiment, the results showed that the performance of CPSO-F2 outperformed 10 out of 11 classification algorithms and its average $MCR$ is quite close to the average $MCR$ of the best classification algorithm, which is MLP. For the third experiment, the results showed that CPSO-F2 outperformed the performance of PSO-$\psi_2$ and PSO-$\psi_3$. From these results, we can conclude that the performance of CPSO-F2 is comparable and competitive with other classification algorithms and that CPSO-F2 can be efficiently used for data classification.

# 3. PARALLEL PARTICLE SWARM OPTIMIZATION CLASSIFICATION ALGORITHM VARIANT IMPLEMENTED WITH APACHE SPARK

Recently, with the rapid development of technologies such as the internet, the amount of data that is collected or generated in many areas such as in the agricultural, biomedical, and finance sectors pose challenges to the scientific community because of the volume and complexity of the data. Furthermore, the need of analysis tools that extract useful information for decision support has been receiving more attention in order for researchers to find a scalable solution to traditional algorithms. In this chapter, we proposed a scalable design and implementation of a particle swarm optimization classification (SCPSO) approach that is based on the Apache Spark framework. The main idea of the SCPSO algorithm is to find the optimal centroid for each target label using particle swarm optimization and then assign unlabeled data points to the closest centroid. Two variants of SCPSO, SCPSO-F1 and SCPSO-F2, were proposed based on different fitness functions, which were tested on real data sets in order to evaluate their scalability and performance. The experimental results revealed that SCPSO-F1 and SCPSO-F2 scale very well with increasing data set sizes and the speedup of SCPSO-F2 is almost identical to the linear speedup while the speedup of SCPSO-F1 is very close to the linear speedup. Thus, SCPSO-F1 and SCPSO-F2 can be efficiently parallelized using the Apache Spark framework.

The remaining of this chapter is organized as follows: In Section 3.1, we present the related work of parallel data mining algorithms that are based on a big data framework. Section 3.2 introduces our proposed approach, SCPSO. Section 3.3 presents the experimental evaluation as well as the results. Finally, Section 3.4 presents our conclusions.

## 3.1. Related Work

A scalable design and implementation of traditional data mining algorithms has recently received attention by researchers to overcome the inefficiency of traditional algorithms for managing and analyzing big data. During the last decade, many of the big data frameworks have been proposed such as MapReduce and Spark, to capture the characteristics of big data which are

Volume, Velocity, Veracity and Variety [51]. In this section, we will present the related work of parallel data mining algorithms that are based on a big data framework.

In [52], the authors proposed a parallelized version of the K-means clustering algorithm using the MapReduce framework. In their proposed approach, the K-means algorithm is formulated as a MapReduce job to find cluster centroid points. In the MapReduce job, the Map function assigns each data point to the closest centroid, while the Reduce function updates the centroids. The MapReduce job is repeated until the stop condition is met. The experimental results revealed that the algorithm can process the large data efficiently on a cluster of nodes.

The parallelization and scalability of a common and effective fuzzy clustering algorithm named Fuzzy C-Means (FCM) algorithm was proposed in [53]. The algorithm was parallelized using the MapReduce paradigm. A validity analysis was conducted in order to show that the implementation works correctly achieving competitive purity results compared to state-of-the art clustering algorithms. Furthermore, a scalability analysis was conducted to demonstrate the performance of the parallel FCM implementation with increasing number of computing nodes used.

In [54], the authors proposed a parallel K-means clustering algorithm based on the Apache Spark framework to overcome the limitations of the K-means algorithm that is provided by the Spark MLIB library. The main work of the algorithm is to perform the distance computation between data points and the centroids on the worker nodes, and the centroid update on the master node. The algorithm was tested with real data sets and the results showed the effectiveness and efficiency of the parallel K-mean algorithm.

Yang and Li in [55] proposed a parallel ant-colony clustering algorithm using the MapReduce framework. In their work, heterogeneous big data is automatically decomposed into clusters using the ant colony algorithm based on data semantic. The algorithm was tested with big data sets and the experimental results demonstrate that the algorithm achieved good accuracy with good efficiency.

In [56], the authors proposed a parallelized version of the particle swarm optimization clustering algorithm using the MapReduce framework (MR-CPSO). In MR-CPSO, the data points are divided into clusters by taking the minimum distances between data points and the cluster centroids. The updating of the particle centroids and the fitness function evaluation in MR-CPSO are carried out through three modules. In the first module, the particle centroids are updated using

Map and Reduce functions. The second module uses the result of the first module, which are the particle centroids, to evaluate the fitness value of each particle using Map and Reduce functions. In the third module, the results from the first and second modules are merged to form a single new swarm for the next iteration. In addition to that, the Best Global and Best Local positions are updated. The scalability of the algorithm was evaluated with synthetic data sets and the results showed the effectiveness and efficiency of the algorithm to process big data sets.

In [57], the author used a PSO clustering algorithm running on the MapReduce platform to cluster streaming twitter data. Twitter data was pre-processed through three phases; tokenizing, stemming, and filtering by removing stop words, before applying the PSO clustering algorithm. The experimental results showed that the parallel PSO algorithm outperformed K-means in terms of the F-measure, and the parallel PSO scaled very well when increasing the dimensionality and size of the data. Another work is reported in [58] whereby the authors proposed an intrusion detection model using the MapReduce-based PSO clustering algorithm to analyze the network traffic. The experiments were conducted with a real traffic network data set and the experimental results revealed that the model achieved good detection rate with a very low rate of false alarms. In addition, the speedup of the model is very close to the linear speedup especially for large data sets.

A new MapReduce-based artificial bee colony clustering algorithm is proposed in [59]. The artificial bee colony (ABC) method was introduced to imitate the foraging behavior of honey bees. In ABC, the colony consists of the employee and onlooker bees that cooperate to find the best food source. ABC is used to solve the data clustering problem by finding the optimal centroids when minimizing the sum of distances between data point and the cluster centroids. The authors proposed a model for ABC clustering using MapReduce in order to handle large data sets. In this model, the two main operations, the updating of centroids and the fitness evaluation, are adapted as a MapReduce job. In the MapReduce job, the Map function computes the distance between the data point and the centroids, then generate a new key-value pair whereby the value is the minimum distance. After that, the Map function emits the key-value pairs to the Reduce function. The Reduce function computes the average distance after aggregating the values with the same key. The experiments were performed on real and synthetic data sets in order to measure the

performance and scalability of ABC clustering. The results revealed the effectiveness and efficiency of the MapReduce-based ABC clustering algorithm.

The grey wolf optimizer (GWO) is one of the most recent swarm intelligence methods that simulates the hunting behavior of grey wolves. In [60], the authors proposed a new algorithm for data clustering using enhanced GWO and MapReduce called MR-EGWO. The MR-EGWO was tested with real and synthetic data sets and compared with four clustering algorithms that are based on MapReduce; parallel K-Means, parallel K-PSO, MapReduce based artificial bee colony optimization (MR-ABC), and the dynamic frequency based parallel k-bat algorithm (DFBPKBA). The results showed that MR-EGWO outperformed the four algorithms in terms of the F-measure, and it also achieved significant speedup performance for large data sets.

In [27], a classification algorithm based on particle swarm optimization (PSO) was proposed by Falco et al. to solve classification problem. The main idea of the PSO classification algorithm is to find the optimal centroid for each target label in a data set using the basic PSO algorithm and then assign each data instance in a testing data set to the closest centroid. Because of inefficient performance of PSO classification algorithms to handle the big data sets.in this chapter, we proposed a parallel version for the PSO classification algorithm using the Apache Spark framework, which is further referred to as SCPSO. In addition, we investigated the scalability and the performance of SCPSO. To the best of our knowledge, this is the first work that implements the PSO classification algorithm using the Apache Spark framework. The aim is to shows how the PSO classification takes advantage of Apache Spark to work on very large data sets to achieve high accuracy and scalability.

### 3.2. Spark-Based PSO Classification Algorithm (SCPSO)

The SCPSO algorithm is based on PSO to find the optimal centroid for all target classes in a data set. In SCPSO, each particle's position and velocity are encoded as vectors as follows:

$$\vec{p_j} = \{p_j^{c_1}, p_j^{c_2}, \ldots, p_j^{c_i}\} \tag{3.1}$$

$$\vec{v_j} = \{v_j^{c_1}, v_j^{c_2}, \ldots, v_j^{c_i}\} \tag{3.2}$$

where $p_j^{c_i}$ and $v_j^{c_i}$ are the position and the velocity vector for particle $j$'s class label $c_i$, respectively, which are represented in an N-dimensional space as follows:

$$p_j^{c_i} = \{p_1, p_2, \ldots, p_n\} \tag{3.3}$$

$$v_j^{c_i} = \{v_1, v_2, \ldots, v_n\} \tag{3.4}$$

where $p_n$ and $v_n$ are the real values of the position and velocity, respectively, for dimension $n$. In addition, each particle has the following attributes:

- Particle Identification Number ($ParticleID$)
- Current Fitness ($FV$)
- Best Centroids Vector that has been discovered so far during the journey of particle ($PBest$)
- Best fitness that has been discovered so far during the journey of particle

Furthermore, SCPSO keeps track of the best centroids vector ($GBest$) and the best group fitness which are achieved by any particle.

To evaluate the fitness of each particle, two fitness functions are used. The first fitness function (F1) computes the fitness of particle $j$ using Equation 3.5 by taking the average of the sum of all Euclidean distances (Equation 3.6) between the current centroid vector of class label $c_i$ ($\vec{p_j}^{c_i}$) and the data instances that belong to class label $c_i$ according to the training data set [27].

$$F1(j) = \frac{1}{D} \sum_{i=1}^{D} d\left(\vec{x}_i, \vec{p_j}^{c_i}\right) \tag{3.5}$$

where $D$ is the number of data instances in a training data set, $\vec{x}_j$ is a data instance vector that belongs to class label $c_i$ according to a training data set, and $\vec{p_j}^{c_i}$ is the current centroid vector for class label $c_i$. It should be noted here that each data instance for each dimension (feature) is normalized within [0.0,1.0] using min-max normalization, and that the sum of those distances is

divided by N (total number of features in data set), and thus the computed distance is also within the range of [0.0,1.0].

$$d\left(\vec{x},\vec{y}\right)=\sqrt{\sum_{i=1}^{n}\left(x_i-y_i\right)^2} \tag{3.6}$$

For the second fitness function $F2$, the fitness of particle $j$ is computed as follows, which is a linear combination of two fitness values:

$$F2(j)=\frac{1}{2}(F1(j)+F\psi(j)) \tag{3.7}$$

The first fitness is computed using the first fitness function $F1$ (Equation 3.5) and the second fitness is computed using the $F\psi$ fitness function. $F\psi$ computes the fitness in two stages. In the first stage, all data instances in a training data set are assigned to the closest centroid. The second stage calculates the percentage of incorrectly classified instances of a training data set [27]:

$$F\psi\left(j\right)=\ \frac{1}{D}\sum_{i=1}^{D}\delta(\vec{x}_i) \tag{3.8}$$

$$\delta(\vec{x}_i)=\begin{cases}1 & if \quad c_{predicted}(\vec{x}_i)\neq c_{actual}(\vec{x}_i)\\ 0 & otherwise\end{cases} \tag{3.9}$$

where $D$ is the number of data instances in a training data set, $c_{actual}$ is the actual outcome of $\vec{x}_i$, and $c_{predicted}$ is the predicted outcome of $\vec{x}_i$.

The particle position updating and fitness evaluation at each iteration are needed to be adapted in order to apply the classification task on a large data set. In SCPSO, these two operations are performed in two stages, as shown in Figure 3.1. In the first stage, the driver program sends the tasks to the executors. Besides, all particles are sent to the executors via a broadcast variable through a cluster manager. Then, each executor reads a part of data records that is encapsulated in an RDD. The pseudo code of each executor for fitness function F1 and F2 evaluation are shown in Algorithm 3.1 and 3.2, respectively. It should be noted here that the executors read a part of data instances only once and cache these data instances in their memory for the next iterations.

In Algorithm 3.1, the executor extracts the data instance vector $x$ from RDD and the position vector $p$ that belongs to the same class label of $x$ from the particle. Then, the distance between $p$ and $x$ is calculated using Equation 3.6. After that, $ParticleID$ and the calculated distance are added to the accumulator variable $Accumulator_{F1}$. This process is repeated over all data instances in RDD.

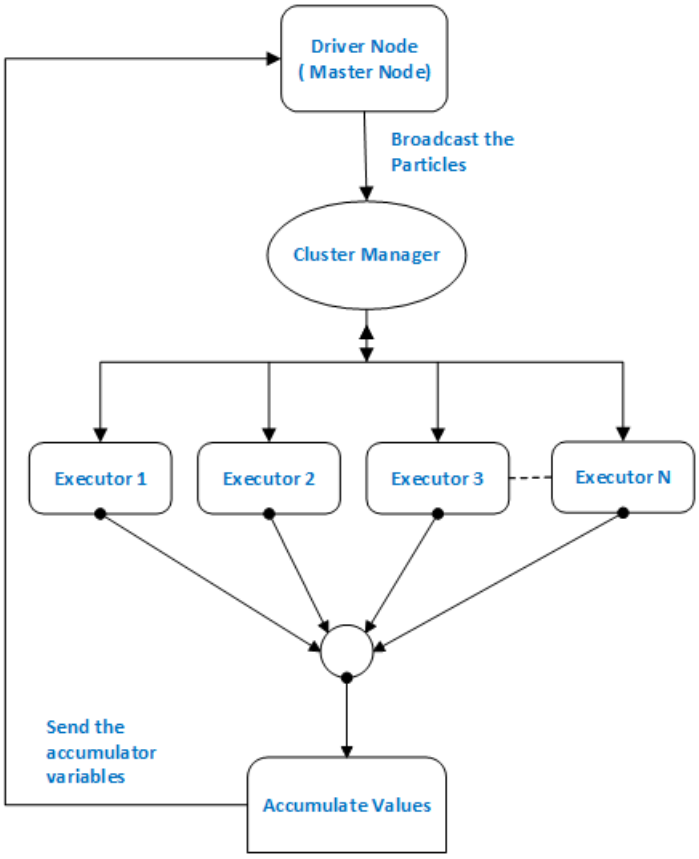



Figure 3.1. SCPSO Architecture

In Algorithm 3.2, the executor extracts data instance vector $x$ from RDD and computes the fitness in two steps. The first step is the same as the previous one and the result is added to the accumulator variable $Accumulator_{F1}$. In the second step, the data instance $x$ is assigned to the class label whose position is the closest to that data instance and then the predicted class label is compared with the actual class label of $x$. If the predicted class label is not the same as the actual class label, then the value is 1.0, and $ParticleID$ are added to the accumulator variable $Accumulator_{F2}$. This process is repeated over all data instances in RDD.

**Algorithm 3.1** Fitness Function 1 Evaluation

```
for each data instance x in RDD do
    for each particle j in Swarm do
        - Extract centroid vector p from particle j which belongs to the same class label of x
        - Compute the distance between x and p using Equation (3.6)
        - Add ParticleID and the distance to accumulator Accumulator_F1
    end for
end for
```

After the executors finish their work, the second stage starts by sending the accumulator variables $Accumulator_{F1}$ or/and $Accumulator_{F2}$ to the driver program to update the fitness $FV$ and the position vector $p$ of the particles. The portion of pseudo code of the driver program is shown in Algorithm 3.3.

**Algorithm 3.2** Fitness Function 2 Evaluation

```
for each data instance x in RDD do
    for each particle j in Swarm do
        // FV1
        - Extract centroid vector p from particle j which belongs to the same class label of x
        - Compute the distance between x and p using Equation (3.6)
        - Add ParticleID and the distance to accumulator Accumulator_F1

        // FVψ
        - Assign x to closest centroid
        if  Assigned_Class(x) != Actual_Class(x) then
            - Add 1.0 and ParticleID to accumulator Accumulator_F2
        end if
    end for
end for
```

The previous operations are repeated until the maximum number of iterations is reached. Then, each data instance in a testing data set is assigned to the class label whose centroid is the closest using Equation 3.6.

### 3.3. Experiments and Results

In this section, we will start by presenting the data sets that are used in our experiments, then describe the execution environment of our experiments. Finally, describe the experiments and the results. We will focus on the scalability measurements; speedup and scaleup for the SCPSO evaluation.

**Algorithm 3.3** Driver Program

Find the number of instances in a training data set $D$ ▷ Is executed only once during the application running.
**for** each particle $j$ in Swarm **do**
  **if** Fitness_Function = F1 **then**
    - Extract a value $V$ of particle $j$ from $Accumulator_{F1}$
    - Update $FV$ of the particle $j$ using Equation (3.5)
    - Update $PBest$ if necessary
  **else if** Fitness_Function = F2 **then**
    - Extract a value $V1$ of particle $j$ from $Accumulator_{F1}$
    - Extract a value $V2$ of particle $j$ from $Accumulator_{F2}$
    - Update $FV$ of the particle $j$ using Equation (3.7)
    - Update $PBest$ if necessary
  **end if**
**end for**
- Take the position vector of particle whose best $FV$ value as $GBest$
**for** each particle $j$ in Swarm **do**
  - Update particle velocity using Equation (1.1)
  - Update particle position using Equation (1.3)
**end for**

### 3.3.1. Data Sets

In our experiments, we used real data sets to evaluate the scalability, performance, and robustness of the SCPSO algorithm, which were taken from the UCI Machine Learning Repository[1], Canadian Institute for Cybersecurity data sets[2], and MOA Machine Learning for Streams[3]. Table 3.1 shows the data sets and their properties. All data sets in Table 3.1 were normalized using Min-Max normalization (Apache Spark MinMaxScaler) [61] to rescale the features to the range [0,1]. In addition, all data instances which have missing values were removed from the data sets.

Furthermore, we split the data set into two data sets whereby 80% of the instances are used for training and the remaining for testing.

### 3.3.2. Environment

We ran our experiments on the SDSC Dell Cluster with Intel Haswell Processors (COMET) operated by the San Diego Supercomputer Center at UC San Diego[4]. The SDSC-Comet cluster consists of 1948 nodes where each node has 24 Intel Xeon cores (2.5 GHz speed) and 128GB of DRAM. For the Apache Spark environment, we used Spark version 2.1, standalone cluster

[1] http://archive.ics.uci.edu/ml
[2] http://www.unb.ca/cic/datasets/index.html
[3] https://moa.cms.waikato.ac.nz/datasets/
[4] https://portal.xsede.org/sdsc-comet

Table 3.1. Properties of Data Sets

| Data set | #Instances | #Features | #Training Instances | #Testing Instances | File Size (MB) | #Class labels |
|---|---|---|---|---|---|---|
| HIGGS [62] | 10,809,619 | 28 | 8,620,812 | 2,188,807 | 5,208.878 | 2 |
| CICIDS2017 [63] | 1,223,895 | 70 | 973,650 | 250,245 | 923.898 | 2 |
| Skin | 245,057 | 3 | 196,000 | 49,057 | 14.092 | 2 |
| NSL-KDD [64] | 140,491 | 37 | 112,394 | 28,097 | 31.411 | 2 |
| Electricity | 45,312 | 8 | 36,250 | 9,062 | 3.348 | 2 |

manager, and Java Runtime 1.8 to implement the SCPSO algorithm. The parameter values of SCPSO algorithm except for the maximum number of iterations are as in [27]:

- maximum number of iterations = 300
- acceleration coefficient constants $c_1$ and $c_2 = 2.0$
- swarm size = 50
- velocity range [$v_{min}$ = -0.05, $v_{max}$ = 0.05]
- weight inertia range [$w_{min} = 0.4$, $w_{max} = 0.9$]

### 3.3.3. Evaluation Measures

To evaluate the scalability, performance, and robustness of SCPSO algorithm, we used the speedup [65] and scaleup [65] measures as well as classification accuracy.

The speedup measures the parallelization ability of the algorithm by taking the ratio of the running time on a single node to the running time on parallel nodes $n$. The speedup is calculated as follows, where the data set size is fixed while the number of nodes is increased in a certain ratio.

$$Speedup = \frac{T_1}{T_n} \tag{3.10}$$

where $T_1$ is the running time using a single node, and $T_n$ is the running time using $n$ nodes.

On the other hand, scaleup measures how the cluster of nodes are utilized efficiently by the parallel algorithm. The scaleup is calculated as follows, where the data set size and the number of nodes are increased by the same ratio.

$$Scaleup = \frac{T_{sn}}{T_{SN}} \tag{3.11}$$

Table 3.2. Accuracy

| Data set | SCPSO-F1 | SCPSO-F2 |
|---|---|---|
| HIGGS | 52.99 | **61.16** |
| CICIDS2017 | 80.00 | **92.69** |
| Skin | 91.01 | **94.35** |
| NSL-KDD | 86.86 | **90.02** |
| Electricty | 53.37 | **75.89** |

where $T_{sn}$ is the running time for the data set with size $s$ using $n$ nodes. $T_{SN}$ is the running time for the data size $S$ using the $N$ nodes, where $S$ is 2-fold of data size $s$ and $N$ is 2-fold of node $n$.

For the robustness evaluation of a model, we used the classification accuracy measure, which is calculated as follows:

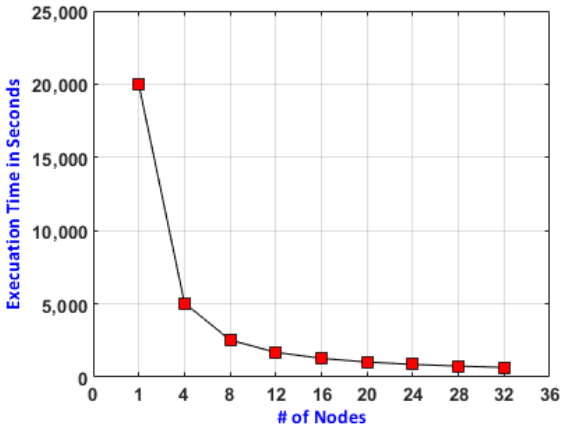
$$Accuracy = (\frac{\#InstancesCorrectlyClassified}{\#Instances}) \times 100 \tag{3.12}$$

### 3.3.4. Results

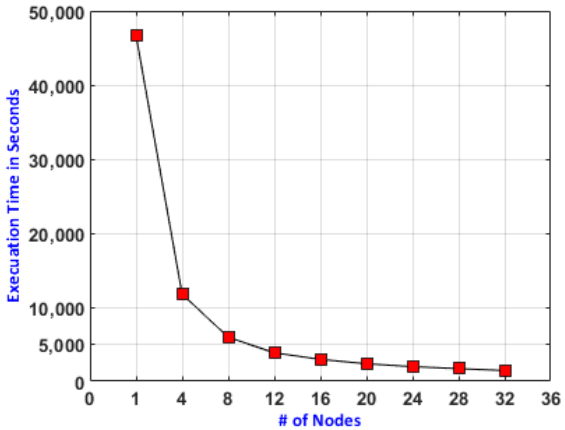
To evaluate the robustness of the SCPSO algorithm, we ran the SCPSO algorithm using two fitness functions F1 and F2 on all data sets in Table 3.1. Table 3.2 shows the accuracy that was achieved by SCPSO-F1 and SCPSO-F2. From the results, we can see that the performance of SCPSO-F2 outperformed the performance of SCPSO-F1 for all data sets in terms of accuracy, where SCPSO-F2 obtained 61.16%, 92.69%, 94.35%, 90.02% and 75.89% compared to the accuracy values obtained by SCPSO-F1, which were 52.99%,80.00%, 91.01%, 86.86% and 53.37%, for HIGGS, CICIDS2017, Skin, NSL-KDD, and Electricity data sets, respectively.

For the scalability evaluation of the SCPSO algorithm, we used the data sets which have more than 1,000,000 instances. Based on the data sets in Table 3.1, HIGGS and CICIDS2017 data sets contain 10,809,619 and 1,223,895, respectively. We performed two experiments using the HIGGS and CICIDS2017 data sets to evaluate the speedup and scaleup of the SCPSO algorithm.

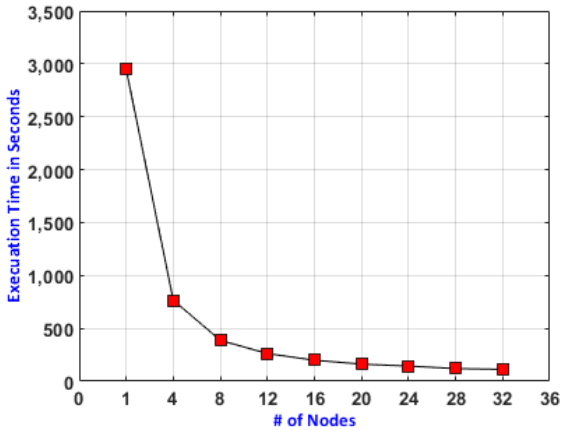
In first experiment, we ran the SCPSO algorithm using two fitness functions F1 and F2 on the HIGGS and CICIDS2017 data sets on the COMET cluster. In each run, the number of nodes is increased by a factor of 4. In addition, we reported the running time in seconds and speedup of the SCPSO algorithm for the HIGGS and CICIDS2017 data sets as shown in Figures 3.2 and 3.3.



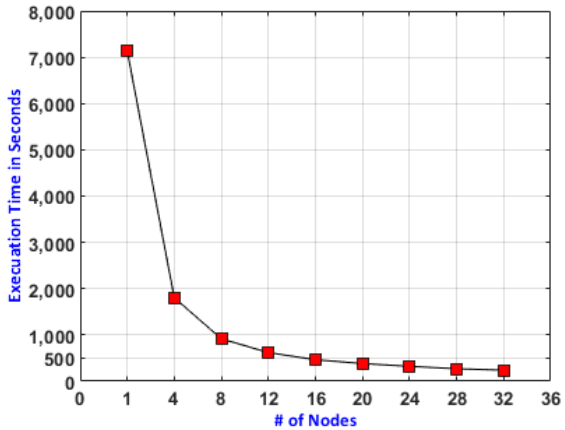

(a) SCPSO-F1 HIGGS




(b) SCPSO-F2 HIGGS

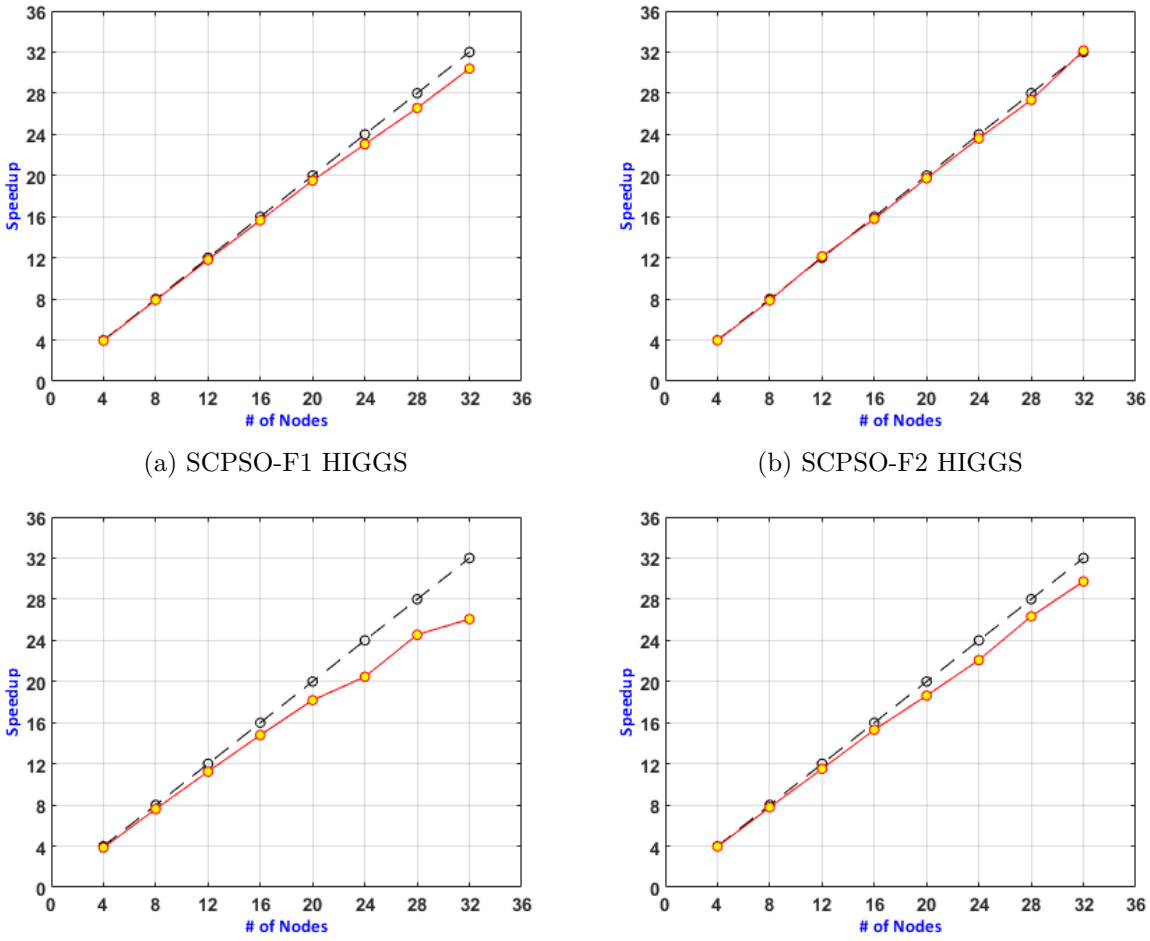



(c) SCPSO-F1 CICIDS2017




(d) SCPSO-F2 CICIDS2017

Figure 3.2. SCPSO-F1 and SCPSO-F2 Running Times




(a) SCPSO-F1 HIGGS

(b) SCPSO-F2 HIGGS

(c) SCPSO-F1 CICIDS2017

(d) SCPSO-F2 CICIDS2017

Figure 3.3. SCPSO-F1 and SCPSO-F2 Speedup; Black Dashed Line Represents Linear Speedup and Red Straight Line Represents Speedup of SCPSO

From Figure 3.2, we can easily see that the running time measured in seconds (s) of SCPSO-F1 is better than the running time of SCPSO-F2 on both, the HIGGS and CICIDS2017 data set, over all nodes. The reason for this is because fitness function F2 computes the fitness in two steps and thus takes longer. In Figures 3.2a and 3.2c the running time of SCPSO-F1 for the HIGGS and CICIDS2017 data sets using 32 nodes were 657s and 113s, respectively, compared to the running time using a single node which were 19,975s and 2,946s for HIGGS and CICIDS2017, respectively. For Figures 3.2b and 3.2d, the running time of SCPSO-F2 for the HIGGS and CICIDS2017 data sets using a single node were 46,703s and 7,136s, respectively, which decreased to 1,453s for HIGGS and 240s for CICIDS2017 using 32 nodes. A significant conclusion that we can draw from Figures 3.2a-3.2d is that the running time of SCPSO-F1 and SCPSO-F2 decreases approximately linearly when the number of nodes are increased.

On other hand as shown in Figure 3.3, we can note that the speedup of SCPSO-F2 is better than the speedup of SCPSO-F1 on the HIGGS and CICIDS2017 data sets. For example, using 32 nodes, the speedup of SCPSO-F2 for HIGGS and CICIDS2017 were 32.14 and 29.73, respectively, compared to the speedup of SCPSO-F1, which were 30.4 and 26.07 for HIGGS and CICIDS2017, respectively. This is because $F2$ is more complex, thus, the utilization using the Spark framework is higher compared to the less complex $F1$ function. This means that the parallelization is more effective on $F2$. In Figure 3.3a, the speedup results of SCPSO-F1 achieved with 4 to 20 nodes using the HIGGS data set were approximately the linear speedup, then the speedup drifts a little away from the linear speedup. In Figure 3.3c, SCPSO-F1 achieved linear speedup results with 4 to 12 nodes using the CICIDS2017 data set, then the speedup results drift away from the linear speedup. In Figure 3.3b and 3.3d, SCPSO-F2 achieved approximately linear speedup results using the HIGGS data set. For the CICIDS2017 data set, the speedup results of SCPSO-F2 using 4 to 16 nodes were approximately similar to the linear speedup, then the speedup drifts a little away starting from 20 nodes.

Finally, we can conclude from Figures 3.3a-3.3d that both versions of SCPSO achieved a significant speedup where the speedup of SCPSO-F2 is almost identical to the linear speedup, and the speedup of SCPSO-F1 is quite close to the linear speedup.

For the second experiment, we measured the scaleup of SCPSO-F1 and SCPSO-F2 to see how the SCPSO-F1 and SCPSO-F2 algorithms utilize the cluster of nodes efficiently, as shown in

Figure 3.4. We can note from Figures 3.4a to 3.4d that the scaleup of SCPSO-F1 has almost a constant ratio ranging between 0.923 and 1.0 for HIGGS, and between 0.981 and 1.0 for CICIDS2017. The scaleup of SCPSO-F2 is a little better than the scaleup of SCPSO-F1. The scaleup of SCPSO-F2 also has almost a constant ratio with a range of 0.976 to 1.011 for HIGGS, and a range of 0.998 to 1.018 for CICIDS2017.

Overall, the SCPSO algorithm becomes more scalable as the data set size increases achieving a significant speedup.

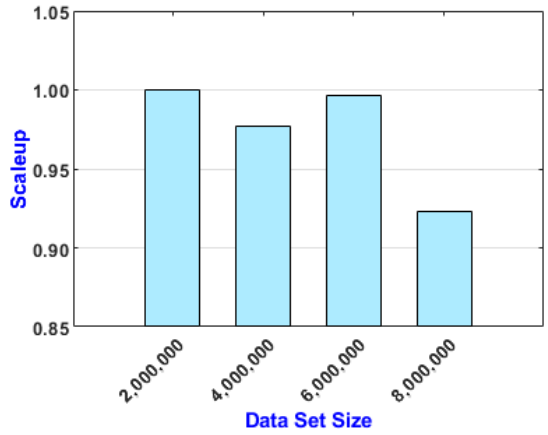
### 3.4. Summary

In this chapter, we designed and implemented a parallel particle swarm optimization classification algorithm (SCPSO) using the Apache Spark framework to overcome the inefficiency of PSO classification of handling big data sets. The SCPSO algorithm uses the basic PSO algorithm to find the optimal centroid for each target label in a data set, and then assigns each unlabeled data instance in a testing data set to the closest centroid. Two versions of SCPSO, SCPSO-F1 and SCPSO-F2, have been proposed based on a different fitness function. SCPSO-F1 and SCPSO-F2 were tested with real data sets to evaluate their scalability and robustness.

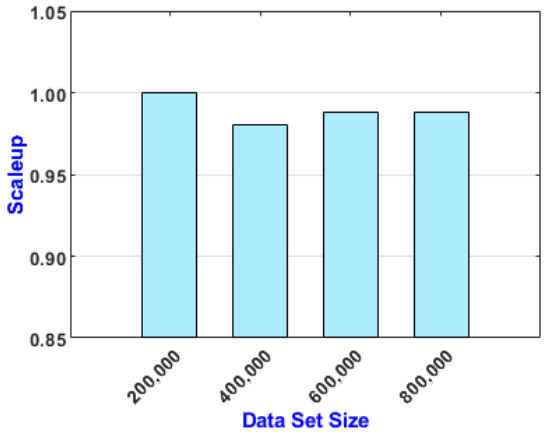
The scalability analysis for SCPSO-F2 revealed that the speedup of SCPSO-F2 is almost identical to the linear speedup, and SCPSO-F2 scales very well with increasing data set sizes. For SCPSO-F1, the scalability analysis showed that the running time of SCPSO-F1 is better than the running time of SCPSO-F2, and the speedup of SCPSO-F1 is very close to the linear speedup.

In term of accuracy of predictive model, the performance of SCPSO-F2 outperformed the performance of SCPSO-F1 on all tested data sets.

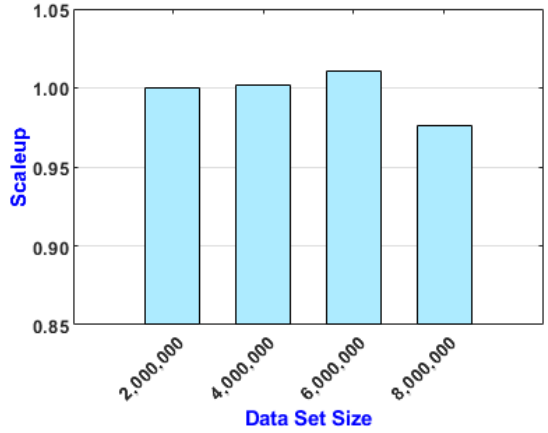
Overall, the experimental results showed that SCPSO-F1 and SCPSO-F2 can be efficiently parallelized with the Apache Spark framework running on cluster of nodes and the performance of SCPSO-F1 and SCPSO-F2 almost follow the ideal speedup with increasing data set sizes.



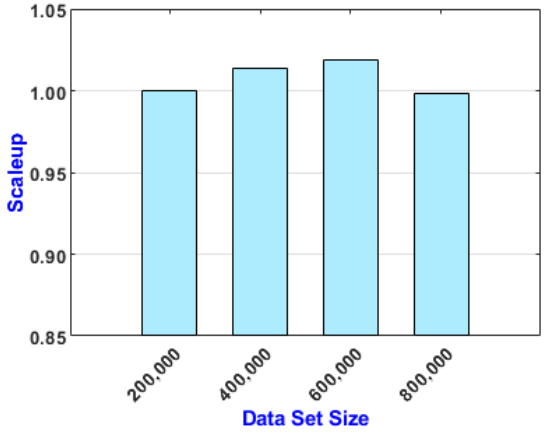

(a) SCPSO-F1 HIGGS




(b) SCPSO-F1 CICIDS2017




(c) SCPSO-F2 HIGGS




(d) SCPSO-F2 CICIDS2017

Figure 3.4. SCPSO-F1 and SCPSO-F2 Scaleup

# 4. A COST-SENSITIVE CENTROID-BASED DIFFERENTIAL EVOLUTION CLASSIFICATION ALGORITHM APPLIED TO CANCER DATA SETS

Nowadays, the collected or generated data for some real-life applications such as in the Medical domain and Intrusion Detection, are typically imbalanced. Imbalanced data sets consist of date where one class-label (minority) includes significantly fewer instances compared to other class labels. The misclassification of the minority class-label could be costly in some circumstances. Therefore, the extraction of valuable information from this kind of data poses a challenge to the scientific community. During the last decades, the researchers proposed a centroid-based classification algorithm using differential evolution (CDE) to solve data classification. However, CDE shows an inefficient performance especially when applied to imbalanced binary data sets. In this chapter, we propose a cost-sensitive version of CDE based on a new objective function in order to overcome this drawback. We are using four cancer data sets that are imbalanced namely Breast, Lung, Uterus, and Stomach. Furthermore, we analyzed and investigated the performance of our proposed version of CDE for predicting the survivability of cancer patients compared to the performance of the current variants of CDE. Moreover, we compared the performance of our proposed version of CDE with the performance of five cost-sensitive machine learning algorithms. The experimental results demonstrate that our proposed version of CDE improves the performance of CDE when applied to imbalanced binary data sets. Furthermore, the performance of our proposed CDE algorithm outperformed the performance of the current variants of CDE on all data sets in terms of Area Under Curve and G-mean.

The remaining sections in this chapter are as follows: Section 4.1 provides an overview of existing works in cancer survivability analysis as well as solving the classification problem using different optimization methods. In Section 4.2, we presents the differential evoluation classification algorithm. Section 4.3 illustrates our proposed approach. Section 4.4 explains the data set and preprocessing. In Section 4.5, we present the experiments as well as the results. Finally, Section 4.6 presents the summary.

### 4.1. Related Work

In this section, we will present research work that is related to solve the data classification task using nature-inspired optimization algorithms.

De Falco et al. [30] proposed a new centroid-based differential evolution classification algorithm method to classify hand-segmented image parts automatically. In this work, the optimal centroids of all target labels are found by minimizing the sum of the Euclidean distances between the data instances in a training data set and the centroid of the actual target label that the data instance belongs to. Then, the optimal centroids are used to classify the instances in a testing data set according to the Euclidean distance. Comparing the approach with ten machine learning algorithms, the DE predictor outperformed nine of these algorithms in terms of misclassification rate applied to the hand-segmented image parts data set.

Another work found in [31], Luukka and Lampinen proposed an approach based on the DE method and principal component analysis (PCA) for solving the classification task. The authors analyzed and studied the performance of the DE classifier using five clinical data sets that are related to Heart disease after applying PCA to these data sets. The experimental results revealed that the performance of the DE classifier is improved by applying PCA to the data sets first. Moreover, the computational time of the DE classifier is reduced because the dimensionality of the data is decreased using PCA. Other works related to solving data classification using DE can be found in [66, 20].

De Falco et al. [27] introduced a new classification algorithm based on the Particle Swarm Optimization method (PSO). The main goal is the same as in the previous works, which is finding the optimal centroids for all labels in a training data set. Three versions of this algorithm were proposed based on different fitness functions. Using thirteen benchmark data sets taken from the UCI Machine Learning Repository [48], the third version achieved better performance than the first and second version, and outperformed five out of the nine comparison machine learning algorithms according to the averaged classification error rate.

In [28], a new classification algorithm based on artificial bee colony (ABC) was proposed to find the optimal centroids by minimizing the sum of all Euclidean distances between the current centroid position of a class label and the data instances that it belongs to. Compared to the second

version of the PSO-based classifier [27] and nine classification algorithms, the experimental results revealed that the ABC-based classifier achieved the best performance in 6 out of 13 data sets in terms of the classification error rate. Moreover, the ABC-based classifier ranked second according to the averaged classification error rate over all data sets.

Firefly Algorithm (FA) is a stochastic search method that belongs to the swarm intelligence family. FA imitates the flashing lights of fireflies, which were introduced by Yang [67]. In [29], the authors proposed the classification algorithm based on FA using the same fitness function as in [27, 28, 30]. Using thirteen data sets, the authors studied and investigated the performance of the FA-based classifier compared to PSO and ABC classifiers and nine traditional classifiers. The experimental results showed that the FA-based classifier obtained the best classification error rate in eight data sets compared to other classifiers. Moreover, the FA-based classifier achieved the best averaged classification error rate. Other research related to centroid-based classification algorithms using nature-inspired algorithms can be found in [68, 69].

The existing variants of CDE have shown efficient to do the data classification task on the balanced binary data sets [30, 31]. However, these variants suffer from handling imbalanced binary data sets. In this chapter, we propose a cost-sensitive version of CDE to address the previous drawback. We introduce a new objective function to handle the imbalanced binary data set by minimizing the misclassification cost instead of the misclassification error. To the best of our knowledge, this is the first work that implements the cost-sensitive version of CDE and applies it to data sets in the medical area in particular on cancer data sets. The aim is to study and investigate the capability of the cost-sensitive-based CDE on handling real-world, imbalanced, and binary data sets.

### 4.2. Differential Evoluation Classifier (CDE)

In [30, 31], the authors proposed variants of a centroid-based DE classification algorithm (CDE) based on different objective functions. The main idea of CDE is to find the optimal centroid of each class label in a training data set and then assign the instances of the unseen data set to the closest centroid.

In CDE, each individual is formed as follows [30]:

$$\vec{x}_i = \{\vec{v}^{c_1}, \vec{v}^{c_2}, \ldots, \vec{v}^{c_n}\} \tag{4.1}$$

Here, $c_1, c_2, \ldots, c_n$ are the class labels in a training data set and $\vec{v}^{c_n}$ is the centroid vector of class label $c_n$, which is a $D$-dimensional vector. The centroid vectors of the initial population are initialized randomly in the $D$-dimensional problem space. After that, the initial population goes through repeated processes of mutation, crossover, and selection to improve the initial population as described in Section 1.2.2.

In our work, the scaling factor $F$ is a random value within the range [0.5, 1.0] using Equation 4.2 [70].

$$F = 0.5 * (1 + rand(0, 1)) \tag{4.2}$$

Futhermore, the crossover value CR is computed using Equation 4.3 [70], which begins from $CR_{max} = 1.0$ and then its value linearly decreases with increasing numbers of generations until it reaches $CR_{min} = 0.5$. At the beginning, the $CR$ value is close to 1.0 which means that most of the target vector elements are replaced by the elements of the mutant vector. But at later generations, the $CR$ value will be decreased linearly, which could lead to the trail vector to inherit more elements from the target vector [70]. It should be noted here that each element in the trail vector should be within the range [0,1].

$$CR_G = CR_{max} - \left( (CR_{max} - CR_{min}) \frac{G}{G_{max}} \right) \tag{4.3}$$

Here, $G$ is the current generation and $G_{max}$ is the maximum number of generations. For the fitnees evaluation, three types of objective functions were proposed to tackle data classification. The first objective function ($F_1$) evaluates the fitness of an individual by computing the rate of misclassification after assigning all instances to the closest centroid according to the shortest distance [31]. Mathematically, the $F_1$ function is described as follows :

$$F_1(\vec{x}_{i,G}) = \frac{1}{N} \sum_{j=1}^{N} \delta(\vec{I}_j) \tag{4.4}$$

$$\delta(\vec{I}_j) = \begin{cases} 1 & if \quad \hat{y} \neq y \\ 0 & otherwise \end{cases} \tag{4.5}$$

where $N$ is the total number of instances in a data set, $\hat{y}$ is the predicted class label of instance $\vec{I}_j$, and $y$ is the actual class of $\vec{I}_j$.

The second objective function ($F_2$) computes the fitness of an individual by taking the average of the sum of all Euclidean distances between the current centroid of class label $c_j$ and the instances in a training data set that belong to class label $c_j$ according to the training data set [30]. Mathematically, the $F_2$ function is described as follows :

$$F_2(\vec{x}_{i,G}) = \frac{1}{N}\sum_{i=1}^{N} d\left(\vec{I}_{i,c_j}, x_{c_j}\right) \tag{4.6}$$

$$d\left(\vec{a}, \vec{b}\right) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{4.7}$$

The third objective function ($F_3$) taken from [27], which is the best fitness function that was reported in [27] to tackle data classification. $F_3$ is a linear combination of two values computed by $F_1$ and $F_2$. Mathematically, the $F_3$ function is described as follows:

$$F_3(\vec{x}_{i,G}) = \frac{1}{2}(F_1(\vec{x}_{i,G}) + F_2(\vec{x}_{i,G})) \tag{4.8}$$

### 4.3. Proposed Approach: Cost-Sensitive Differential Evolution Classifier

In our work, we propose a cost-sensitive version of CDE in order to cope with imbalanced binary data sets by minimizing the misclassification cost instead of the misclassification error. We propose a new objective function $F_{cost}$, which is a minimization objective function. In $F_{cost}$, first each class label should be assigned a misclassification cost. Then, $F_{cost}$ computes the fitness level of the individual vector $\vec{x}_{i,G}$ in two steps. In the first step, all instances in a training data set are assigned to the closest centroid according to the Euclidean distance. After that, the second step is carried out by summing over the misclassification cost of all instances that are misclassified as follows:

$$F_{cost}(\vec{x}_{i,G}) = \sum_{j=1}^{N} \psi_{cost}(\vec{I}_j) \tag{4.9}$$

$$\psi_{cost}(\vec{I_j}) = \begin{cases} Cost^{+} & if \quad \hat{y} \neq y \quad and \quad y = + \\ Cost^{-} & if \quad \hat{y} \neq y \quad and \quad y = - \\ Otherwise & 0 \end{cases} \tag{4.10}$$

where $Cost^{+}$ is the misclassification cost of the positive class label, $Cost^{-}$ is the misclassification cost of the negative class label, $N$ is the total number of instances in the data set, $\hat{y}$ is the predicted class label of instance $\vec{I_j}$, and $y$ is the actual class label of $\vec{I_j}$.

### 4.4. Medical Application: Data Set and Data Set Preparation

In our work, we used the latest version of the SEER data set (released in April 2018) [71, 72], which has been collected from various registries in the United States by the Surveillance, Epidemiology, and End Results (SEER) program of the National Cancer Institute (NCI) since 1973. The SEER data set is a public data set, which contains information of cancer incidences in the US that cover 34.6% of the US population [73].

In the SEER data set, each record represents the information of cancer incidence for one patient stored as fixed-length text. We developed a preprocessing tool using Java SDK 1.8 to extract the sub data sets of breast, lung, uterus, and stomach cancer incidences from the SEER data. The chosen features (variables) for these data sets are the same that have been used in previous research [74, 75, 76, 77] as shown in Table 4.1 [71] besides using three features, which are Vital Status Recode, Cause Of Death, and Survival Months to tag an instance. It should be noted here that the cancer incidences that were extracted for our experiments have been diagnosed since 2004, because some of the SEER features such as Tumor Size, Tumor Extension, and Lymph Nodes, are just reported for cancer incidences that have been diagnosed since 2004.

The survivability of cancer patients is determined by the rule that is shown in Figure 4.1 [74, 75, 76]. According to that rule, the patient who lives at least five years (60 months) since cancer diagnosis; this patient is tagged as "Survived". The patient who died within five years after cancer diagnosis, and cause of death is the same type of cancer that was diagnosed at the beginning; this patient is tagged "Not Survived". Otherwise, a record is ignored.

Table 4.1. The Description of Chosen Features

| # | **Feature Name** | **Desription** |
|---|---|---|
| 1 | Race/Ethnicity | - |
| 2 | Gender[1] | - |
| 3 | Marital Status at DX | The marital status of the patient at cancer diagnosis. |
| 4 | Age of Diagnosis | The patient's age at cancer diagnosis. |
| 5 | Primary Site[2] | The initial site of the primary tumor. |
| 6 | Histologic Type ICD-O-3 | The structure of the primary tumor. |
| 7 | Behavior Code ICD-O-3 | Type of primary tumor; Benign or Malignant. |
| 8 | Grade | Describes the spreading speed of the tumor and its form. |
| 9 | Regional Nodes Positives | The exact number of lymph nodes that contain metastases. |
| 10 | Regional Nodes Examined | The exact number of examined or removed lymph nodes. |
| 11 | CS Tumor Size | The size of the tumor. |
| 12 | CS Extension | Describes the extension of tumor. |
| 13 | CS Lymph Nodes | Provides information about involvement of lymph nodes. |
| 14 | CS Mets at DX | Metastasis. |
| 15 | RX Summ-Surg Prim Site | Describes the surgical routine in the primary site that removes or/and destroys involved tissues. |
| 16 | Summary stage 2000 (1998+) | Describes the stage of the cancer's spread. |
| 17 | Sequence Number-Central[3] | Provides description of the number and sequence of all malignant, in situ, benign, and borderline primary tumors that occurred during the lifetime of a patient. |

[1] This feature was only extracted for the stomach and lung cancer data sets.
[2] This feature is used to identify the type of cancer.
[3] This feature was only extracted for the uterus and lung cancer data sets.

Table 4.2. Properties of Data Sets

| **Data set** | **Survive (-)** | **Not Survive (+)** | **Total** |
|---|---|---|---|
| Breast Cancer | 271,972 (87.67%) | 38,272 (12.33%) | 310,244 |
| Lung Cancer | 23,898 (18.82 %) | 103,092 (81.18%) | 126,990 |
| Uterus Cancer | 23,337 (76.16%) | 7,303 (23.84%) | 30,640 |
| Stomach Cancer | 4,169 (27.65%) | 10,909 (72.35%) | 15,078 |

**if** SM $\geq$ 60 and VSR=" alive " **then**
    tag the patient as "Survived"
**else if** SM < 60 and COD=" Type of cancer " **then**
    tag the patient as "Not Survived"
**else**
    Ignore this record
**end if**

Figure 4.1. Rule of the Survivability of Cancer Patients. VSR: Vital Status Recode, SM: Survival Months, and COD: Cause of Death

For the patients who have more than one record in the SEER data set, the most recent record of a patient was extracted. Concerning the records which have missing values in the chosen features, we decided to exclude these records.

Table 4.2 shows the properties of the breast, uterus, lung and stomach cancers data sets after preprocessing. In Table 4.2, we reported the number of instances and distribution percentage of each class label, and the total number of instances in each data set. From Table 4.2, we can easily see that all data sets are highly imbalanced. For example, in the breast cancer data set, 87.67% of instances belong to class label "Survive" while the remaining 12.33% belonging to class label "Not Survive". All data sets are normalized using Min-Max normalization before applying the CDE algorithm.

### 4.5. Experiments and Results

In our work, we conducted two experiments to evaluate the robustness and performance of the cost-sensitive CDE algorithm for predicting the survivability of cancer patients. In the first experiment, we ran the CDE algorithm using the $F_{cost}$ objective function to assess the impact of $F_{cost}$ on the performance of CDE compared to the existing fitness functions ($F_1$,$F_2$, and $F_3$).

For the second experiment, we compared the cost-sensitive CDE algorithm with five cost-sensitive Machine Learning (ML) algorithms according to their performance in predicting the survivability of cancer patients.

For the classifier's performance evaluation, the Area Under Curve (AUC) and Geometric Mean (G-mean) measures are used to evaluate the performance and robustness of the classification algorithms among other evaluation measures. The reason for that is that all data sets are imbalanced; thus accuracy, misclassification rate, or recall and precision measures do not reflect the accurate performance of the classification algorithms.

AUC is the most important and popular measure in medical applications, which summarizes the performance of a classifier into a single value. AUC is the calculated area under the Receiver Operating Characteristic (ROC) curve, which is calculated as follows [78, 79, 80]:

$$AUC = \frac{1 + TP_{rate} - FP_{rate}}{2} \tag{4.11}$$

G-mean is another measure to assess a classifier's performance on a highly imbalanced data set, which is sensitive to the True rate of the minority class label [81, 82, 83]. This measure is calculated as follows:

$$G - mean = \sqrt{sensitivity \times specificity} \tag{4.12}$$

In our experiments, all data sets in Table 4.2 were split into two data sets; 80% of the data set was used as the training data set, and the remaining 20% as the testing data set.

It should be noted here that the misclassification cost of the class labels are specified empirically starting with one, then the cost is incremented until finding the best misclassification cost. Table 4.3 shows the best misclassification cost of the "Not Survive (+)" and "Survive (-)" class labels that were found in each data set.

Moreover, the parameters of the CDE algorithm were taken from [70], except the maximum number of generations, which are:

- Maximum number of generations = 200

Table 4.3. Misclassification Cost of Survive (-) and Not Survive (+) Class Labels for Each Data Set

| Data set | Survive (-) | Not Survive (+) |
|---|---|---|
| **Breast Cancer** | 1.0 | 7.0 * |
| **Lung Cancer** | 4.0 * | 1.0 |
| **Uterus Cancer** | 1.0 | 3.0 * |
| **Stomach Cancer** | 2.0 * | 1.0 |

* Minority class label.

- Population size $NP = 10$ * (Data Set Dimensionality)
- Crossover range [$\text{CR}_{\text{min}} = 0.5$, $\text{CR}_{\text{max}} = 1.0$]
- Scaling factor $F$ = a random value in range [0.5,1.0]

### 4.5.1. First Experiment

To evaluate the impact of our proposed objective function ($F_{cost}$) on the performance of the CDE algorithm when applied to imbalanced binary data sets, we compared its performance with the performance of three variants of CDE that are based on $F_1$,$F_2$, and $F_3$. Four variants of CDE are introduced based on the objective function, which are abbreviated as follows: $CDE - F_{cost}$, $CDE - F_1$, $CDE - F_2$ and $CDE - F_3$. We performed 25 independent runs for $CDE - F_{cost}$, $CDE - F_1$, $CDE - F_2$, and $CDE - F_3$ on all data sets given in Table 4.2 and reported the G-mean and AUC results. Furthermore, the G-mean and AUC results obtained by $CDE - F_{cost}$ have been compared statistically with the G-mean and AUC results obtained by $CDE - F_1$, $CDE - F_2$, and $CDE - F_3$ using the Wilcoxon Signed-Rank test at the 5% significance level. Table 4.4 and 4.5 show the average of the G-mean and AUC results (25 independent runs), respectively, obtained by $CDE - F_{cost}$, $CDE - F_1$, $CDE - F_2$, and $CDE - F_3$ for each data set. In addition, we reported the standard deviation within brackets in both tables.

From the results in Table 4.4, we can see that the performance of $CDE - F_{cost}$ statistically outperforms the performance of $CDE - F_1$, $CDE - F_2$, and $CDE - F_3$ in terms of G-mean for all data sets, where the p-value is 1.2290e-05 for all. The best G-mean results on the Breast, Lung, Uterus, and Stomach cancer data sets were obtained by $CDE - F_{cost}$, which were 79.60%, 83.87%, 83.97%, and 79.49%, respectively. In terms of AUC, the results in Table 4.5 revealed that

Table 4.4. G-Mean Results Achieved by $CDE - F_1$, $CDE - F_2$, $CDE - F_3$, and $CDE - F_{cost}$

| | $CDE - F_{cost}$ | $CDE - F_1$ | $CDE - F_2$ | $CDE - F_3$ |
|---|---|---|---|---|
| **Breast Cancer** | **79.60%** [±**0.10%**] | 61.51% [±0.46%] | 77.61% [±0.04%] | 61.78% [±0.43%] |
| **Lung Cancer** | **83.87%** [±**0.09%**] | 72.62% [±0.88%] | 80.76% [±0.05%] | 73.03% [±0.67%] |
| **Uterus Cancer** | **83.97%** [±**0.17%**] | 79.81% [±0.63%] | 83.03% [±0.04%] | 80.14% [±0.49%] |
| **Stomach Cancer** | **79.49%** [±**0.37%**] | 74.79% [±0.49%] | 74.04% [±0.07%] | 74.96% [±0.48%] |
| **Average** | **81.73%** | 72.18% | 78.86% | 72.48% |

$CDE - F_{cost}$ obtained the best AUC on all data sets compared to $CDE - F_1$, $CDE - F_2$, and $CDE - F_3$, where the AUC results were 79.61%, 83.93%, 83.98%, and 79.60% for Breast, Lung, Uterus, and Stomach cancer data set, respectively. Furthermore, the performance of $CDE - F_{cost}$ statistically outperforms the performance of $CDE - F_1$, $CDE - F_2$, and $CDE - F_3$ on all data sets according to the p-value of 1.2290e-05 for all. Figures 4.2 and 4.3 show the distribution of the G-mean and AUC results (from 25 independent runs) that were obtained by $CDE - F_1$, $CDE - F_2$, $CDE - F_3$, and $CDE - F_{cost}$.

To draw a conclusion over all data sets, the G-mean and AUC results are averaged over all data sets, as shown in the last row in Table 4.4 and 4.5. From these results, we can see that $CDE - F_{cost}$ achieved the best averaged G-mean and AUC compared to $CDE - F_1$, $CDE - F_2$, and $CDE - F_3$. Moreover, the previous results revealed another significant conclusion that is that the $F_{cost}$ objective function improves CDE's performance when applied to imbalanced binary data set.

### 4.5.2. Second Experiment

In this experiment, we compare the performance of the cost-sensitive CDE, which is $CDE - F_{cost}$, in terms of G-mean and AUC with the performance of cost-sensitive classification algorithms. Five ML algorithms were employed that are cost-sensitive classification algorithms, which are Naive Bayes [84], Decision Tree C4.5 (J48) [85], Logistic Regression [86], RBF network [87], and IBk (5

Table 4.5. AUC Results Achieved by $CDE - F_1$, $CDE - F_2$, $CDE - F_3$, and $CDE - F_{cost}$

| | $CDE - F_{cost}$ | $CDE - F_1$ | $CDE - F_2$ | $CDE - F_3$ |
|---|---|---|---|---|
| **Breast Cancer** | **79.61%** [±**0.10%**] | 68.40% [±0.26%] | 77.64% [±0.04%] | 68.56% [±0.25%] |
| **Lung Cancer** | **83.93%** [±**0.09%**] | 74.97% [±0.62%] | 80.81% [±0.05%] | 75.18% [±0.48%] |
| **Uterus Cancer** | **83.98%** [±**0.16%**] | 80.89% [±0.47%] | 83.05% [±0.04%] | 81.10% [±0.40%] |
| **Stomach Cancer** | **79.60%** [±**0.34%**] | 76.32% [±0.35%] | 74.21% [±0.08%] | 76.41% [±0.35%] |
| **Average** | **81.78%** | 75.15% | 78.93% | 75.31% |

nearest neighbors) [88]. For the cost-sensitive classification algorithms, the cost matrix should be provided by assigning the misclassification penalty for each class label. We used the same misclassification cost given in Table 4.3 for the five cost-sensitive classification algorithms. For running the cost-sensitive classification algorithm, the Waikato Environment for Knowledge Analysis (WEKA) tool version 3.6 was used [49, 50], which is one of the most popular tools for running data mining tasks. Furthermore, the default setting provided by the WEKA was used for the five ML algorithms.

Table 4.6 and 4.7 show the G-mean and AUC obtained by the five cost-sensitive classification algorithms and $CDE - F_{cost}$, respectively, for the given data sets. It can be seen from the results in Table 4.6 and 4.7 that $CDE - F_{cost}$'s performance outperforms the performance of all cost-sensitive classification algorithms in 3 out of 4 data sets in terms of G-mean and AUC. $CDE - F_{cost}$ achieved the best G-mean on the Breast, Lung, and Uterus cancer data sets, where the G-mean results were 79.60%, 83.87%, and 83.97%, respectively. Additionally, the best AUC results on the Breast, Lung, and Uterus cancer data sets were obtained by $CDE - F_{cost}$, where the AUC results were 79.61%, 83.93%, and 83.98%, respectively. For the Stomach cancer data set, $CDE - F_{cost}$'s performance outperforms the performance of 3 out of 5 cost-sensitive classification algorithms in terms of G-mean and AUC. The best G-mean and AUC results for the stomach cancer data set was achieved by the cost-sensitive Decision Tree (C4.5) algorithm.

Table 4.6. G-Mean Results for Five Cost-sensitive Classification Algorithms and $CDE - F_{cost}$

| | $CDE - F_{cost}$ | **Naïve Bayes** | **Decision Tree (C4.5)** | **Logistic Regression** | **RBF Network** | **IBK** |
|---|---|---|---|---|---|---|
| **Breast Cancer** | **79.60% [±0.10%]** | 69.34% | 75.73% | 79.33% | 76.15% | 75.85% |
| **Lung Cancer** | **83.87% [±0.09%]** | 82.67% | 80.11% | 83.66% | 82.99% | 79.88% |
| **Uterus Cancer** | **83.97% [±0.17%]** | 81.99% | 83.31% | 83.78% | 81.10% | 81.43% |
| **Stomach Cancer** | 79.49% [±0.37%] | 74.27% | **81.12%** | 79.65% | 78.42% | 75.50% |
| **Average** | **81.73%** | 77.07% | 80.07% | 81.61% | 79.66% | 78.17% |

Finally, we summarized the results by taking the average of the G-mean and AUC results of each algorithm over all data sets as shown in the last row in Table 4.6 and 4.7. From the average results, we can deduce that $CDE - F_{cost}$ achieved the best averaged G-mean and AUC results, which are 81.73% and 81.78%, respectively, outperforming all cost-sensitive classification algorithms.

### 4.6. Summary

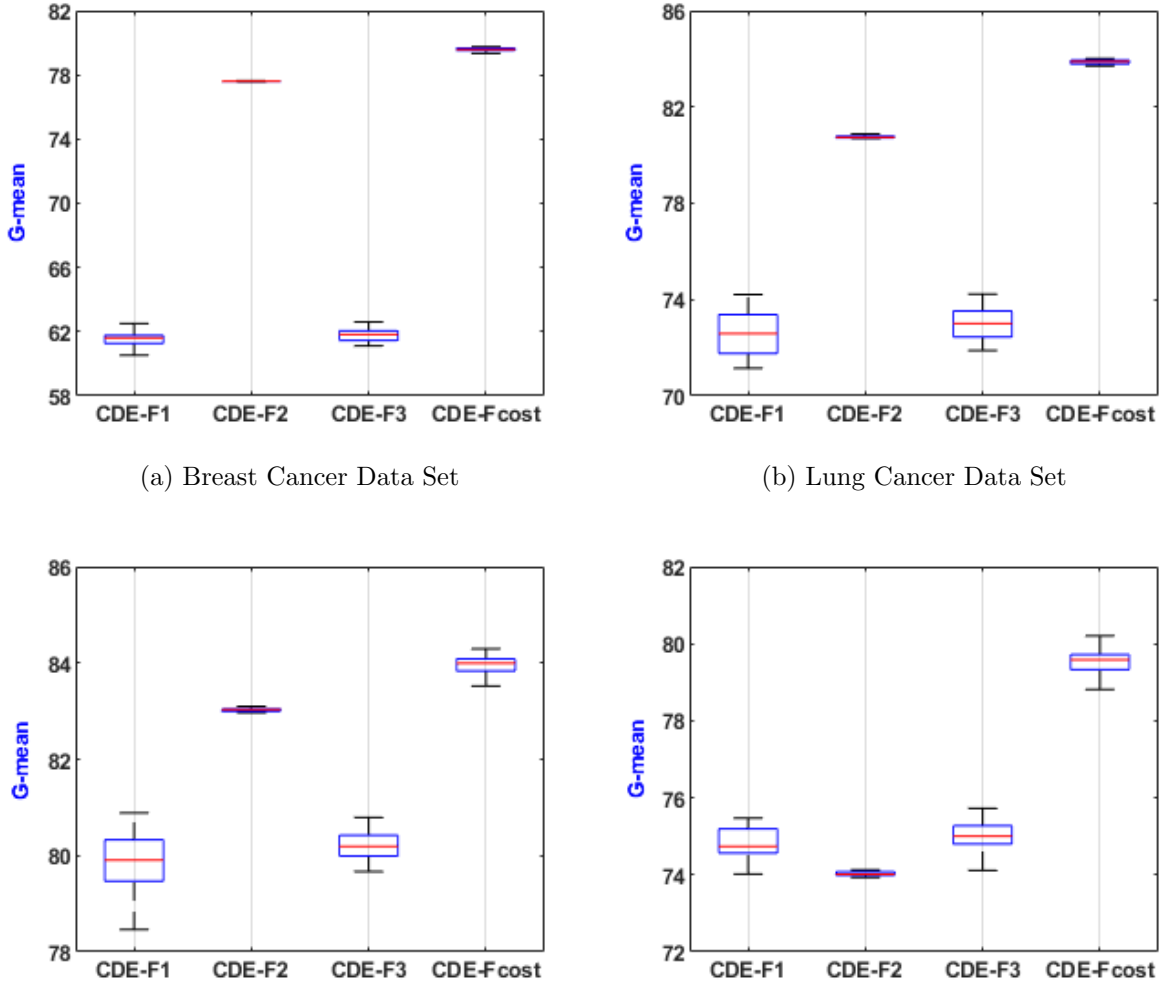
In this work, we proposed the cost-sensitive differential evolution ($CDE - F_{cost}$) based on a new objective function $F_{cost}$. Two experiments were conducted using four real medical data sets extracted from the SEER data to assess the performance of $CDE - F_{cost}$ starting by evaluating the impact of the objective function $F_{cost}$ on the performance of the centroid-based differential evolution classification algorithm ($CDE$) and then comparing the performance of the $CDE - F_{cost}$ with the performance of the five cost-sensitive classification algorithms with respect to the G-mean and AUC measures.

The findings revealed that $CDE$ efficiently handles highly imbalanced binary data sets using the $F_{cost}$ objective function. Additionally, $F_{cost}$'s performance outperformed the performance of three existing objective functions ($F_1$, $F_2$, and $F_3$) with respect to G-mean and AUC. Furthermore, $CDE - F_{cost}$ performed best in 3 out of 4 cancer data sets compared to the five cost-sensitive clas-

Table 4.7. AUC Results for Five Cost-sensitive Classification Algorithms and $CDE - F_{cost}$

| | $CDE - F_{cost}$ | **Naïve Bayes** | **Decision Tree (C4.5)** | **Logistic Regression** | **RBF Network** | **IBK** |
|---|---|---|---|---|---|---|
| **Breast Cancer** | **79.61% [±0.10%]** | 71.97% | 76.34% | 79.39% | 76.82% | 75.85% |
| **Lung Cancer** | **83.93% [±0.09%]** | 82.87% | 80.19% | 83.77% | 83.01% | 80.36% |
| **Uterus Cancer** | **83.98% [±0.16%]** | 82.17% | 83.37% | 83.80% | 81.31% | 81.60% |
| **Stomach Cancer** | 79.60% [±0.34%] | 75.39% | **81.14%** | 79.76% | 78.42% | 75.59% |
| **Average** | **81.78%** | 78.10% | 80.26% | 81.68% | 79.89% | 78.35% |

sification algorithms. For the Stomach cancer data set, $CDE - F_2$'s performance outperformed the performance of three out of the five cost-sensitive classification algorithms. Overall, the experimental results showed that $CDE - F_{cost}$ can be successfully used to cope with highly imbalanced binary data sets. In addition, it is competitive compared to all cost-sensitive classification algorithms.



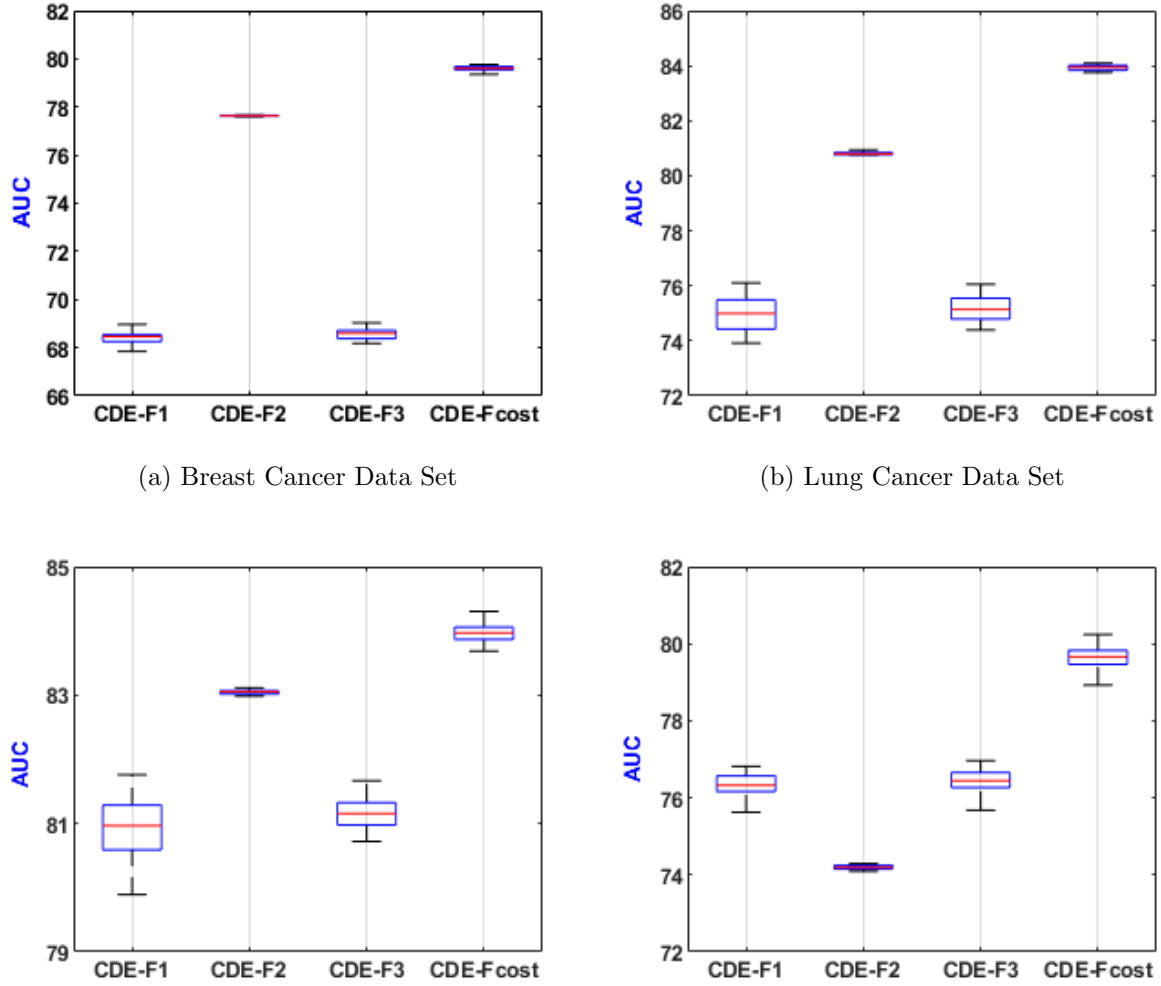

(a) Breast Cancer Data Set

(b) Lung Cancer Data Set

(c) Uterus Cancer Data Set

(d) Stomach Cancer Data Set

Figure 4.2. Box Plots of the G-Mean Results Obtained by $CDE - F_1$, $CDE - F_2$, $CDE - F_3$, and $CDE - F_{cost}$ for Breast, Lung, Uterus, and Stomach Cancer Data Sets. Red Bar Inside the Box Represents the Median; Whiskers Above and Below the Box Represents Maximum and Minimum Values, Respectively.




(a) Breast Cancer Data Set

(b) Lung Cancer Data Set

(c) Uterus Cancer Data Set

(d) Stomach Cancer Data Set

Figure 4.3. Box Plots of the AUC Results Obtained by $CDE-F_1$, $CDE-F_2$, $CDE-F_3$, and $CDE-F_{cost}$ for Breast, Lung, Uterus, and Stomach Cancer Data Sets. Red Bar Inside the Box Represents the Median; Whiskers Above and Below the Box Represents Maximum and Minimum Values, Respectively.

# 5. PERFORMANCE EVALUATION OF A COST-SENSITIVE DIFFERENTIAL EVOLUTION CLASSIFIER USING SPARK

Nowadays, the amount of data that has been collected or generated in many sectors has been growing exponentially because of the rapid development of technologies such as the Internet of Things (IoT). Additionally, the nature of this data is imbalanced. The need for extracting valuable information for decision support from this data poses a challenge to the scientific community to find a solution to cope with large imbalanced data. A cost-sensitive differential evolution classification algorithm showed itself as efficient for handling highly imbalanced data set. However, this algorithm shows inefficient performance when applied to big data set. Furthermore, this algorithm lacks to scale with data size increases. In this chapter, we design and implement a parallel version of the cost-sensitive differential evolution classifier using the Apache Spark framework (SCDE). The aim is to handle large and binary imbalanced data. The main idea of the algorithm is to find the optimal centroid for each target label using differential evolution by minimizing the total misclassification cost and then assign unlabeled data points to the closest centroid. Our experiments include a real data set that is based on intrusion detection in order to evaluate our algorithm's scalability and performance. The experimental results show that SCDE efficiently handles imbalanced binary data and scales very well with data size increases. Moreover, the speedup and scaleup results that are obtained by SCDE are close to optimal.

The remainder of this chapter is organized as follows: Section 5.1 provides an overview of existing works in parallel data mining algorithms using big data frameworks. Section 5.2 illustrates our proposed approach - SCDE. Section 5.3 describes the data set and preprocessing. In Section 5.4, we present the scalability and performance analysis of SCDE. Finally, Section 5.5 presents our conclusions.

## 5.1. Related Work

Recently, designing and implementing scalable solutions for traditional ML algorithms or data mining algorithms that are based on optimization methods have received recent attention

from the scientific community. In particular, the inefficiency of these algorithms for handling and analyzing big data in order to extract valuable information is investigated. The researchers have proposed a scalable solution to accommodate the rapid growth of data and run these algorithms in a suitable time using a high-performance cluster of nodes [89]. In this section, we will start presenting the existing parallel solutions of these algorithms based on a big data framework. Additionally, the parallel implementation of the optimization methods are also described.

In [59], a new ABC clustering algorithm based on the MapReduce model (MR-ABC) was proposed. MR-ABC aims to find the optimal centroids by minimizing the sum of the Euclidean distances. MR-ABC carries out fitness level evaluation by launching a MapReduce job. During the MapReduce job, the Map function creates a new key-value pair whereby the value is the minimum computed distance between the data point and the centroids. Then, emitting the key-value pairs to the Reduce function to calculate the average distance by aggregating the values with the same key. MR-ABC was experimented using real and synthetic data sets, and the analysis of the results showed the effectiveness and robustness of the MR-ABC for solving the clustering task.

Another work found in [90], a Spark-based artificial bee colony for clustering was proposed. The idea of clustering is the same as MR-ABC [59], but the authors used the Apache Spark framework to run the algorithm. The algorithm evaluates the fitness of bees (individuals) in a parallel manner by distributing the RDDs to the worker nodes along with the individuals. The algorithm was tested using the KDDCUP99 data set to evaluate its effectiveness. The experimental result analysis showed that the algorithm obtained good accuracy and achieved almost linear speedup.

A novel clustering algorithm using enhanced grey wolf optimizer that is run using the MapReduce framework (MR-EGWO) and was introduced in [60]. MR-EGWO starts by dividing the data set into partitions that are distributed among the Hadoop nodes. Then, the MapReduce job is launched to create a set of key/value pairs using the Map function. After that, the set of key/value pairs is used by the Reduce function to collect values based on a key. MR-EGWO was tested using real and synthetic data sets. The results showed that MR-EGWO outperformed four MapReduce-based clustering algorithms according to the F-measure and also obtained significant speedup results.

In [91], the authors proposed a parallel version of DE using the MapReduce (MR) paradigm in order to improve the running time of the optimization method when solving a large-scale problem.

In this work, the fitness evaluation is carried out through the Map and Reduce phases. Nevertheless, the experimental results showed that the exhaustive I/O disk operations during the shuffling and sorting phases are reducing the parallelization performance. Two variant parallelization approaches of DE based on Apache Spark, a master-slave and an island-based, have been proposed in [92] to overcome the drawback of the previous work. The master-slave and island-based approaches were experimented on the AWS cloud using synthetic and real biology-inspired benchmarks. The results revealed that both approaches show good scalability with increasing numbers of nodes. Other works related to the parallel implementation of the optimization method can be found in [93],[94],[95],[96].

The authors in [97] proposed a parallel implementation of the DE clustering algorithm using the MapReduce framework. DE clustering is carried out through three levels. First, a MapReduce job is launched to carry out the mutation and crossover operations for generating new offspring. Then, the current population and offsprings are evaluated using the fitness function by running the MapReduce job. After that, the selection operation is performed to improve the current population. The proposed approach was tested using 18 real gene expression data sets, and the experimental results achieved by the proposed approach are found to be more reliable compared to K-means and PSO MapReduce.

A Spark-based clustering algorithm using particle swarm optimization (S-PSO) was proposed in [98]. In this work, a new strategy was introduced to improve the quality of the clustering by running the k-means algorithm in the final stage. S-PSO runs through repeated three steps which are fitnes.s evaluation, personal and global best updating, and position and velocity updating. The fitness evaluation, and position and velocity updating steps are run in a parallel manner using the map and reduce functions. After S-PSO is completed, the global best position is taken as an initial centroid for the k-means algorithm that runs in parallel. Using real and synthetic data sets, S-PSO achieved good performance and scalability results. Other works for implementing PSO clustering using Apache Spark can be found in [99], [100].

Another work found in [90], a Spark-based artificial bee colony for clustering was proposed. The idea of clustering is the same as MR-ABC, but the authors used the Apache Spark framework to run the algorithm. The algorithm evaluates the fitness of bees (individuals) in a parallel manner by distributing the RDDs to the worker nodes along with the individuals. The algorithm was tested

using the KDDCUP99 data set to evaluate its effectiveness. The experimental result analysis showed that the algorithm obtained good accuracy and achieved almost linear speedup.

### 5.2. Cost-Sensitive Differential Evolution Classifier Based on Spark

In Chapter 4, a cost-sensitive differential evolution classification algorithm was proposed that can efficiently handle highly imbalanced binary data sets. However, this algorithm shows inefficient performance to cope with the big data sets. In this work, we proposed a scalable design for the cost-sensitive differential evolution classifier based on Apache Spark (SCDE) in order to handle big data sets. The SCDE algorithm is based on differential evolution to find the optimal centroid of each class label in a training data set. Then, each instance in a testing data set is assigned to the closest centroid based on the Euclidean distance. SCDE starts with a predefined number of individuals $NP$ that form the initial population $Pop_{initial}$. Each individual $\vec{x}_i$ is encoded as follows:

$$\vec{x}_i = \{\vec{v}_{c_1}, \vec{v}_{c_2}, \ldots, \vec{v}_{c_n}\}, i = \{1, 2, \ldots, NP\} \tag{5.1}$$

where $\vec{v}_{c_n}$ is a $d$-dimensional centroid vector of class label $c_n$.

Furthermore, each individual has an individual identification number ($IndividualID$) and the fitness value $FL$. Each centroid vector of individual $\vec{x}_i$ in the initial population $Pop_{initial}$ is randomly initialized in the $d$-dimensional problem space. After initialization, the individuals are evaluated using an objective function $F$ to measure their fitness as follows:

$$F(\vec{x}_i) = \sum_{j=1}^{N} F\psi_{cost}(\vec{I}_j) \tag{5.2}$$

$$F\psi_{cost}(\vec{I}_j) = \begin{cases} Cost^{+} & if \quad \hat{y} \neq y \quad and \quad y = + \\ Cost^{-} & if \quad \hat{y} \neq y \quad and \quad y = - \\ Otherwise & 0 \end{cases} \tag{5.3}$$

here, $Cost^{+}$ is the misclassification cost of the positive class label, $Cost^{-}$ is the misclassification cost of the negative class label, $N$ is the total number of instances in the data set, $\hat{y}$ is the predicted class label of instance $\vec{I}_j$, and $y$ is the actual class label of $\vec{I}_j$. The fitness level of the individual vector $\vec{x}_i$ is evaluated in two steps. In the first step, all instances in a training data set are assigned

to the closest centroid according to the Euclidean distance, Equation 5.4. After that, the second step is carried out by summing the misclassification cost of all instances that are misclassified.

$$d\left(\vec{a},\vec{b}\right) = \sqrt{\sum_{k=1}^{n}(a_k - b_k)^2} \tag{5.4}$$

Then, $Pop_{initial}$ goes through a repeated evolution process to improve $Pop_{initial}$ as described in Section 1.2.2.

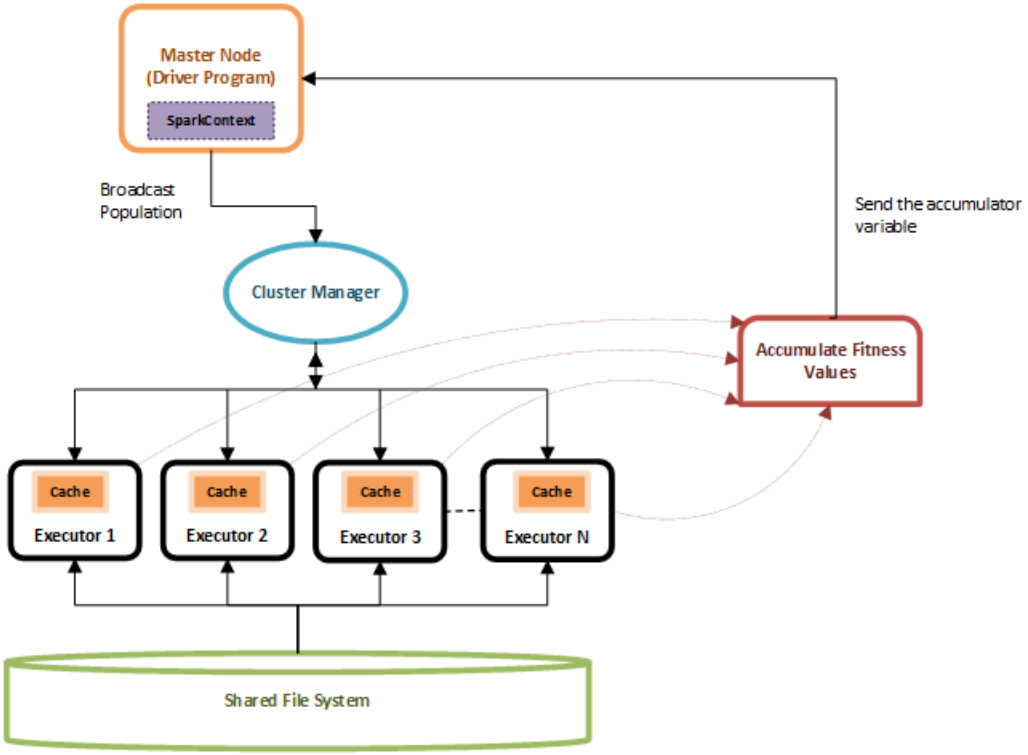



Figure 5.1. SCDE Architecture

In our work, the scaling factor $F$ is a random value chosen using Equation 5.5 [70]. In addition, the crossover rate $CR$ value is linearly decreasing with increasing numbers of generations (Equation 5.6) [70]. The aim is to explore the entire space at the beginning by replacing most of the target vector elements with the mutant vector elements. But at later generations, the $CR$ value will linearly decrease, thus, more elements will inherit from the target vector. This leads to

explore the interior space comprehensively [70]. It should be noted here that each element in the trail vector should be within the range [0,1].

$$F = 0.5 * (1 + rand(0, 1)) \tag{5.5}$$

$$CR = CR_{max} - \left( (CR_{max} - CR_{min}) \frac{G}{G_{max}} \right) \tag{5.6}$$

Here, $G$ is the current generation and $G_{max}$ is the maximum number of generations.

The operations for generating a new population in the following generation $G+1$, besides the fitness evaluation, need to be adapted in order to work on large data sets. For example, suppose the data set contains 100 million examples with two class labels and the population size is 20, thus, the cost-sensitive DE classifier needs to compute 100,000,000 $\times$ 2 $\times$ 20 = 4,000,000,000 distance values during one iteration. Since the fitness evaluation of the individuals requires more computational time and consumes much more memory space compared to the operations (mutation, crossover, and selection) for generating a new population in generation $G+1$. In SCDE, the fitness evaluation of the individuals is carried out across a cluster of worker nodes while the operations for generating a new population for $G + 1$ is performed in the master node.

Figure 5.1 shows the architecture of SCDE. In Figure 5.1, the driver program starts creating a directed acyclic graph (DAG) for the RDD operations and sends the tasks to the executors. In addition, the driver program sends the current population with or without their offspring as a broadcast variable to the executors through a cluster manager for fitness evaluation. After that, the executors starts reading the part of the RDD to work on. In this stage, each executor starts extracting one instance vector $I$ at a time from the RDD. Then, for each individual, $I$ is assigned to the closest centroid based on the Euclidean distance. If the class label of the assigned centroid does not match with the actual class label $cl$ of $I$, then $IndividualID$ and the cost of misclassifying $cl$ are added to the accumulator variable $Accumulator_{FV}$ as shown in Algorithm 5.1. It should be noted here that the data instances are read only once by the executors and are cached in executors' memory for the next iterations.

After the executors finished, the accumulator variable $AccumulatorFV$ is sent to the driver program. In the driver program (master node), the generation of the new population for next

**Algorithm 5.1** Fitness Function Evaluation

```
for each data instance I in RDD do
    for each individual in Broadcast Variable do
        Assign I to closest centroid
        if Assigned_Class(I) != Actual_Class(I) then
            Add MisclassificationCost and IndividualID to accumulator Accumulator_FV
        end if
    end for
end for
```

generation $G + 1$ starts with updating the fitness of all individuals in the current population *Pop*. Then, *Pop* sequentially goes through the mutation, crossover, and selection operations.

The previous operations are repeated until the maximum number of generation is reached.

### 5.3. Data Set and Environment

In our work, we used a new and reliable intrusion data set, which is CICIDS2017 [63, 101], to evaluate the performance and scalability of SCDE. CICIDS2017 was generated by the Canadian Institute for Cybersecurity, which contains the most recent types of attacks captured during five days starting from July 3, 2017. The CICIDS2017 data set consists of eight CSV files, which are the network traffic analysis results that were collected using the CICFlowMeter tool [101]. Each CSV file contains one type of attack along with normal traffic (Benign). Besides, each record has 78 features describing the behavior of 25 users. For our experiments, we focused on four types of attacks which are Brute force, Botnet, Port Scan, and Web Attack that were taken from CICIDS2017. Table 5.1 shows the data sets including the number of features, the total number of instances in each data set as well as the number of instances and the distribution percentage of each class label. From the table we can easily see that three out of four data sets are highly imbalanced. For example, Botnet attack represents only 01.02% of instances in the Botnet data set.

It should be noted here that the instances that have a missing value were removed. Furthermore, we performed supervised feature selection on the data sets using WEKA [49, 50] to reduce the number of features as shown in Table 5.1, and then applied the Min-Max normalization technique to normalize all data sets.

The Brute Force and Web Attack data sets are multi-labels data sets. Therefore, we decided to transform both data sets into binary data sets by mapping various attack types to one attack. In the Brute Force data set, “SSH-Patator” and “FTP-Patator” labels are mapped to one label

Table 5.1. Properties of Data Sets

| Data set | # Features | Label | # Instances | Percentage | Total |
|---|---|---|---|---|---|
| **Botnet** | 4 | BENIGN (Normal) | 188,955 | 98.98% | 190,911 |
| | | Botnet Attack * | 1,956 | 01.02% | |
| **Brute Force** | 3 | BENIGN (Normal) | 431,813 | 96.90% | 445,645 |
| | | Brute Force Attack * | 13,832 | 03.10% | |
| **Web Attack** | 4 | BENIGN (Normal) | 168,051 | 98.72% | 170,231 |
| | | Web Attack * | 2,180 | 01.28% | |
| **Port Scan** | 5 | BENIGN (Normal) | 127,292 | 44.49% | 286,086 |
| | | Port Scan Attack | 158,804 | 55.51% | |

* A minority class label.

named "Brute Force". For the Web Attack data set, "Web Attack-Sql", "Web Attack-XSS", and "Web Attack-Brute Force" labels are mapped to the "Web Attack" label.

The experiments were run on the SDSC Dell Cluster with Intel Haswell Processors (COMET) operated by the San Diego Supercomputer Center at UC San Diego[1]. The SDSC-Comet cluster consists of 1,948 nodes that have 24 Intel Xeon cores (2.5 GHz speed) and 128GB of DRAM. We implemented and run the SCDE algorithm using Java Runtime 1.8, Spark version 2.1, and the standalone cluster manager.

## 5.4. Experiments and Results

In this section, we will start by describing the measurements that were used to evaluate the performance and scalability of SCDE and also the parameter setting of SCDE. Then, the experiments and the results are explained. We will focus in particular on the scalability for SCDE including speedup and scaleup.

### 5.4.1. Evaluation Measures

In this work, we used various measures to assess the robustness and scalability of the SCDE algorithm. To assess the performance and effectiveness of SCDE, we used the Detection Rate, False Alarm Rate, Geometric Mean (G-mean), and Area under Curve (AUC) measures. The following are the descriptions of these measures:

[1]https://portal.xsede.org/sdsc-comet

– True positive (TP): The number of attack instances that are detected correctly.

– True negatives (TN): The number of normal behavior instances that are classified correctly.

– False positives (FP): The number of normal behavior instances that are classified as attacks.

– False negatives (FN): The number of attack instances that are undetected.

– Detection Rate: The ratio of detecting the attacks is calculated as follows:

$$\frac{TP}{TP + FN} \tag{5.7}$$

– False Alarm Rate: The ratio of the wrong prediction of normal behavior is calculated as follows:

$$\frac{FP}{TN + FP} \tag{5.8}$$

– G-mean: This metric is one of the essential measures to evaluate the performance of a classifier on a highly imbalanced data set [102], which is calculated as follows:

$$G - mean = \sqrt{sensitivity \times specificity} \tag{5.9}$$

– AUC: This metric is another essential measure to evaluate the performance of the classifier on a highly imbalanced data set [102], which is the area under the curve between TPR and FPR.

For the scalability evaluation, we used the following measures:

– Speedup [65]: Is a metric that measures the parallelization capability of the application, which is the ratio of the computation time on a single node $T_1$ to the computation time on $P$ nodes $T_p$. The speedup is calculated as follows:

$$Speedup = \frac{T_1}{T_p} \tag{5.10}$$

Here, the number of nodes $P$ is increased in a certain ratio while the data set size is fixed.

Table 5.2. Misclassification Cost of the Class Labels for Each Data Set

| **Data set** | **Class Label** | **Misclassification Cost** |
|---|---|---|
| Botnet | BENIGN | 1.0 |
| | Botnet Attack * | 97.0 |
| Brute Force | BENIGN | 1.0 |
| | Brute Force Attack * | 31.0 |
| Web Attack | BENIGN | 1.0 |
| | Web Attack * | 76.0 |
| Port Scan | BENIGN * | 1.2 |
| | Port Scan Attack | 1.0 |

* Minority class label.

– Scaleup [65]: Is a metric that measures how the cluster of nodes are utilized efficiently by the parallel algorithm, which is calculated as follows:

$$Scaleup = \frac{T_{sp}}{T_{SP}} \tag{5.11}$$

Here, $T_{sp}$ is the running time for the data set size $s$ using the $p$ nodes. $T_{SP}$ is the running time for the data size $S$ using the $P$ nodes, where $S$ is 2-fold of data size $s$ and $P$ is 2-fold of node $p$.

For the experiments, the parameters of SCDE were taken from [70], except the maximum number of generations, which are:

- Maximum number of generations = 200
- Population size $NP$ = 10 * (Data Set Dimensionality)
- Crossover range [$CR_{min}$=0.5, $CR_{max}$ = 1.0]
- Scaling factor $F$: a random value in range [0.5, 1.0]

In addition, the misclassification cost of the class labels are empirically identified as shown in Table 5.2.

Table 5.3. SCDE's Performance Results

| Data set | Detection Rate | False Alarm Rate | G-mean | AUC |
|---|---|---|---|---|
| **Botnet** | 99.81% [±0.11%] | 15.57% [±0.02%] | 91.80% [±0.06%] | 89.16% [±0.43%] |
| **Brute Force** | 100.00% [±0.0] | 00.89% [±0.0] | 99.56% [±0.0] | 99.54% [±0.0] |
| **Web Attack** | 88.14% [±2.50%] | 06.99% [±1.55%] | 90.52% [±0.71%] | 89.74% [±1.81%] |
| **Port Scan** | 99.22% [±0.01%] | 00.94% [±0.07%] | 99.14% [±0.04%] | 99.02% [±0.21%] |

Table 5.4. Duplicate Data Sets

| Data set | Duplication Rate | Size |
|---|---|---|
| 20T | 20 times | 8,912,900 |
| 40T | 40 times | 17,825,800 |
| 60T | 60 times | 26,738,700 |

### 5.4.2. Performance Analysis

For the performance experiment, we performed 25 independent runs for SCDE on the data sets in Table 5.1 to evaluate the performance and effectiveness of SCDE. Table 5.3 shows the average results achieved by SCDE for all data sets in terms of detection rate, false alarm rate, G-mean, and AUC. Furthermore, the standard deviation is given within the brackets. From Table 5.3, we can observe that SCDE achieved the highest detection rate and the lowest false alarm rate as well as the best G-mean and AUC results in the Brute Force data set compared to the other data sets. In the Port Scan data set, SCDE achieved excellent results for the detection rate and false Alarm rate, G-mean, and AUC where the results were 99.22%, 0.94%, 99.14%, and 99.02%, respectively. For the Botnet and Web Attack data sets, SCDE achieved very good results according to the detection rate and false Alarm rate, G-mean, and AUC as shown in Table 5.3. Overall, SCDE can detect attacks efficiently with low false alarm rate.

### 5.4.3. Scalability Analysis

Since the data sets in Table 5.1 are too small to measure the speedup of SCDE on a large cluster of nodes, we decided to duplicate the largest data set in Table 5.1, which is the Brute Force data set by replicating the original data set 20, 40, and 60 times. Table 5.4 shows the sizes of the Brute Force data sets after duplication.

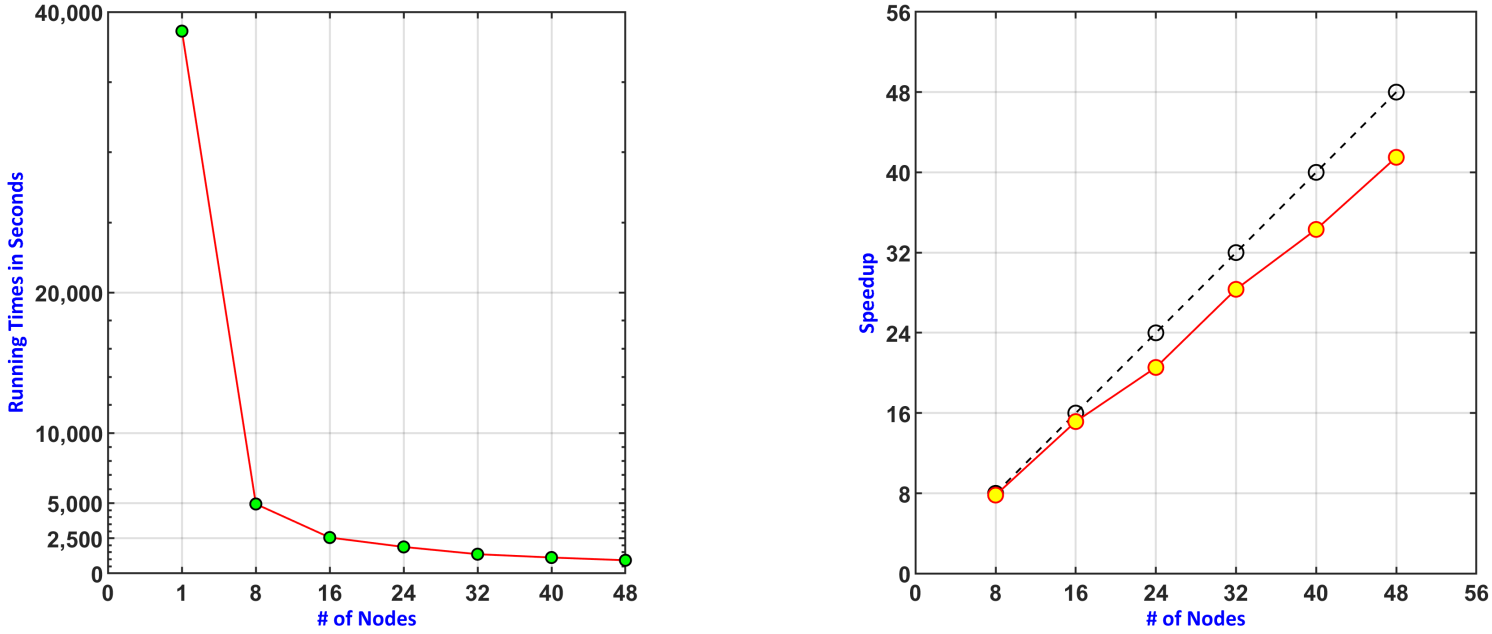
In the speedup measurement experiment, we ran SCDE on all duplicate data sets for up to 48 nodes on the COMET cluster. For each run, the number of nodes is increased by multiples of eight while the data size is kept fixed. we reported the running time in seconds and the speedup of SCDE for all duplicate data sets as shown in Figures 5.2 to 5.4. The black dashed line in Figures 5.2b, 5.4b, 5.4b represents the linear speedup.

From Figures 5.2a, 5.3a, and 5.4a, we can easily observe that the running time of SCDE for all data sets is decreasing approximately exponentially. For example, the running time of SCDE for the 20T, 40T, and 60T data sets using 48 nodes is reduced to 931s, 1,758s, and 2,676s, respectively, compared to the running time of SCDE using one node which are 38,633s, 76,761s, 122,219s for the 20T, 40T, and 60T data sets, respectively.

In the speedup evaluation, we can see from Figure 5.2b that the speedup results for 8 and 16 nodes using the 20T data set are approximately the linear speedup. The speedup values then drift away a little from the linear speedup starting from 24 nodes, where the speedup results for 24, 32, 40, and 48 nodes using the 20T data set are 20.55, 28.34, 34.31, and 41.50, respectively.

In Figure 5.3b , the speedup results using the 40T data set are almost optimal for 8, 16, 24, and 32 nodes where the speedup results are 7.61, 14.89, 22.25, 29.72, respectively. For 40 and 48 nodes, the speedup results are 36.31, and 43.66, which are very close to the linear speedup.

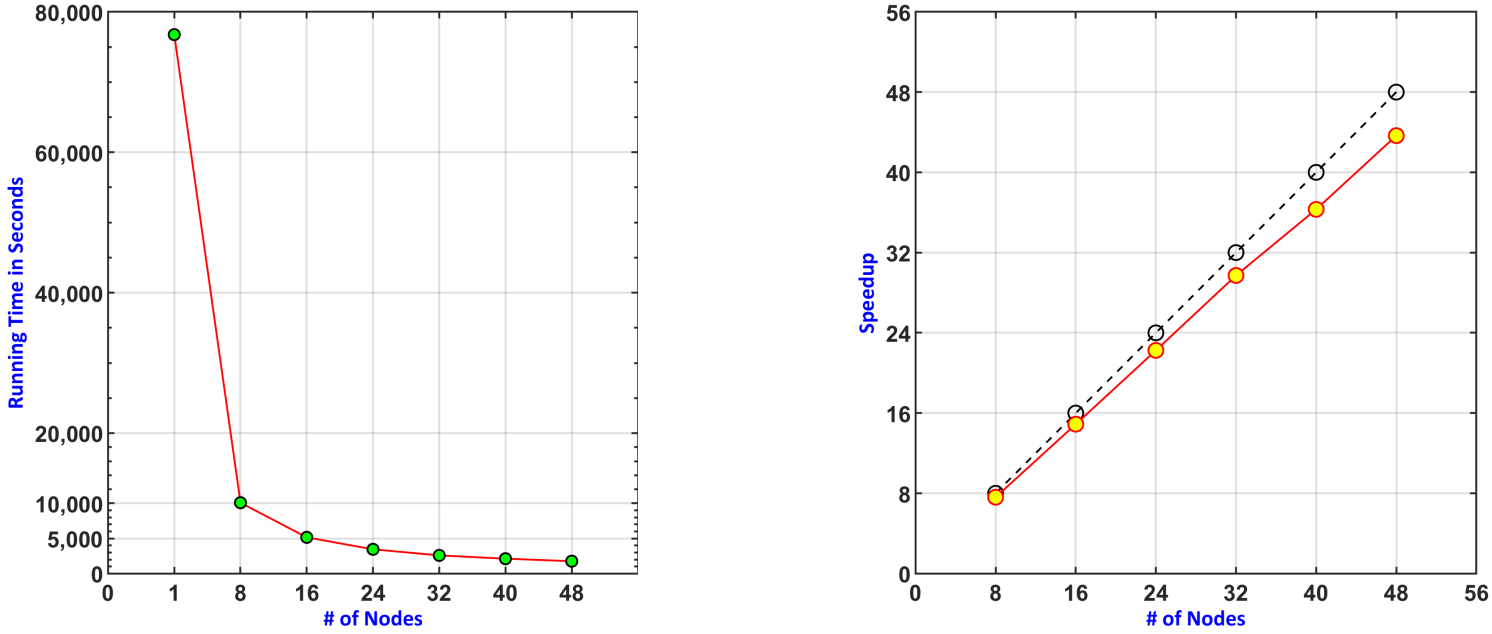
For the 60T data set, the speedup results are almost identical to the linear speedup where the speedup results for 8, 16, 24, 32, 40, and 48 nodes are 7.96, 15.62, 22.56, 30.09, 38.30, and 45.67, respectively as shown in Figure 5.4b. Overall, we can conclude from the previous results that the running time of SCDE is significantly improved with increasing numbers of nodes. Furthermore, the speedup achieved by SCDE is close to the linear speedup.




(a) Running Time

(b) Speedup

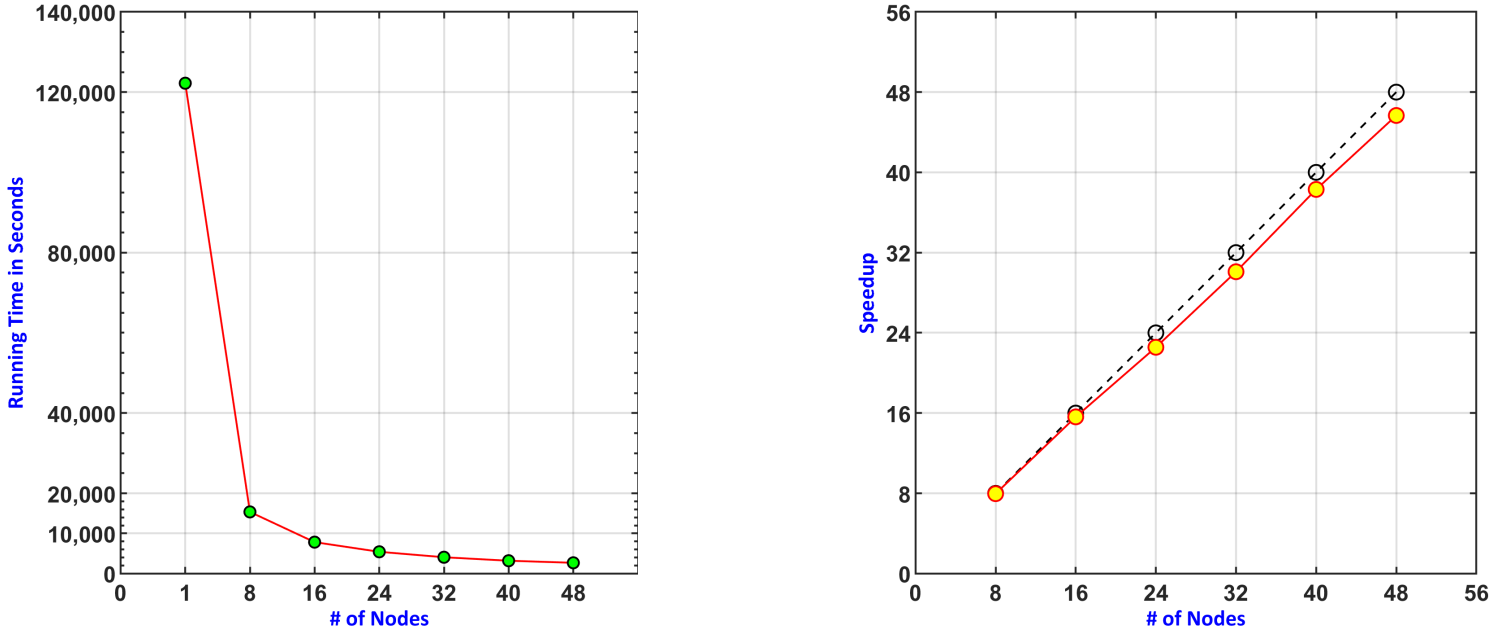
Figure 5.2. Running Time and Speedup Using the 20T Data Set




(a) Running Time

(b) Speedup

Figure 5.3. Running Time and Speedup Using the 40T Data Set




(a) Running Time

(b) Speedup

Figure 5.4. Running Time and Speedup Using the 60T Data Set

For the Scaleup evaluation experiment, we used the Brute Force data set to run SCDE on the COMET cluster to measure the capability of SCDE on utilizing the cluster of nodes efficiently. For this experiment, we doubled the data set size and number of nodes for each run starting from 445,645 records for data size and 2 nodes. Figure 5.5 shows the scaleup results achieved by SCDE. From this figure, SCDE shows good scaleup results that have approximately a constant ratio ranging between 0.967 and 1.0. Moreover, the results are close to 1.0 which is the optimal value.

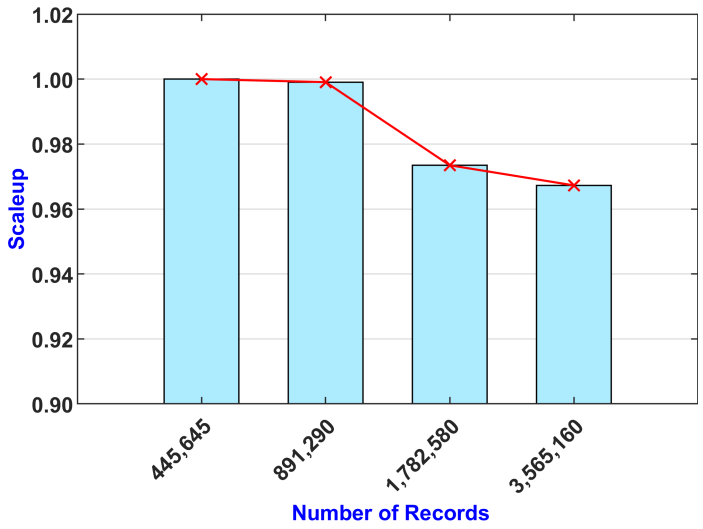



Figure 5.5. SCDE Scaleup

## 5.5. Summary

In this chapter, we designed and implemented a scalable cost-sensitive differential evolution (SCDE) algorithm using Apache Spark to solve the classification task for imbalanced and large data. SCDE is based on DE to find the optimal centroid vector for each class label by minimizing the total misclassification cost. SCDE assigns each instance in a data set to the closest centroid.

The experiments used real intrusion detection data sets to evaluate SCDE's performance and scalability. The experimental results revealed that SCDE efficiently detects an attack with a low false alarm rate. Furthermore, the scalability analysis showed that the improvement factor of SCDE's running times are very close to the optimal values for most data sets, and SCDE scales efficiently with data size increases.

Overall, we can conclude that SCDE is robust and efficient for handling imbalanced binary large data and achieved good speedup and scaleup results that are close to optimal.

# 6. CONCLUSION AND FUTURE WORK

The classification algorithms based on nature-inspired methods have been demonstrated as efficient and robust to solve classification tasks. In this study, we focused on two of the nature-inspired algorithms which are particle swarm optimization (PSO) and differential evolution (DE). The PSO and DE classification algorithms have been efficiently used for data classification. However, these algorithms suffer from their inability of scaling with a growth in data size. Moreover, these algorithms lack in handling imbalanced data.

In this dissertation, we designed scalable solutions using the Apache Spark framework in order to address the inefficient performance of the PSO and DE algorithms when applied to large data. The aim was to run these algorithms on large data in a reasonable time while maintaining the solution quality. Besides, we enhanced the performance of these algorithms.

The main reason for choosing Apache Spark over Hadoop MapReduce is that Apache Spark has low latency by caching data or objects across worker nodes. Moreover, Apache Spark carries out the interactive and iterative jobs more efficiently than Hadoop MapReduce and provides rich APIs that can help the developers to program an application easily.

In this dissertation, we first proposed a new approach to improve the performance of the PSO-based classification algorithm. The main idea of the proposed approach is to find the optimal centroid using PSO, then the standard deviation for each target label is computed based on that centroid. After that, each instance in a test data set is classified using the normal distribution probability density function and the probability of each target label. The proposed approach was tested using ten data sets. The experimental results showed that the performance of the PSO-based classification algorithm is good, i.e., achieving a low misclassification rate.

Secondly, we proposed two parallel versions of the PSO-based classification algorithm using Apache Spark (SCPSO) based on different fitness functions. The scalability and performance of both versions were studied and analyzed using real large data sets. The experimental results showed that Apache Spark is an efficient and robust framework to design and implement a scalable solution for the PSO-based classification algorithm. Moreover, the results demonstrate that the proposed

algorithm utilizes a cluster of nodes efficiently and achieves good scalability results in terms of speedup and scaleup.

Thirdly, a cost-sensitive variant of the DE classification algorithm was proposed by introducing a new objective function in order to handle imbalanced binary data sets. We analyzed and investigated the capability of the cost-sensitive DE classifier using four cancer data sets (imbalanced) namely Breast, Lung, Uterus, and Stomach. The experiments showed that the cost-sensitive DE classification algorithm can efficiently handle highly imbalanced binary data sets and its performance outperformed the performance of current variants of the DE classification algorithm in terms of Area Under Curve (AUC) and G-mean. Moreover, its performance is competitive compared to the performance of five cost-sensitive classification algorithms concerning AUC and G-mean.

Finally, we designed and implemented a parallel version of a cost-sensitive differential evolution classifier using the Apache Spark framework (SCDE). The aim was to handle imbalanced binary data sets and take advantage of Apache Spark to work on large data sets to achieve high performance and scalability. The idea of the algorithm is to find the optimal centroid of each class label in a training data set by minimizing the total misclassification cost and then assigning unlabeled data points to the closest centroid. Our experiments included a real data set based on intrusion detection in order to evaluate our algorithm's scalability and performance. The experimental results showed that the scalable design of the cost-sensitive differential evolution classifier scales very well with data size increases besides achieving good scalability results in terms of speedup and scaleup.

Our future work aims to apply the SCPSO and SCDE algorithms to other real-world applications with very large data sets (terabyte size) using hundreds of nodes and conduct comprehensive experiments to investigate the utilization of the resources used. Furthermore, we will apply a cost-sensitive function to new optimization algorithms such as artificial bee colony and bat algorithm and conduct a comprehensive performance study. Moreover, finding a solution to cope with imbalanced multi-label data set is another future topic of investigation.

**Acknowledgment**

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562.

# REFERENCES


[1] Saed Sayad. *Real time data mining*. Self-Help Publishers, 2011.

[2] Mohammed Zaki and Wagner Meira. *Data mining and analysis fundamental concepts and algorithms*. Cambridge University Press, 2014.

[3] Parul Agarwal and Shikha Mehta. Nature-inspired algorithms: state-of-art, problems and prospects. *International Journal of Computer Applications*, 100:14–21, 2014.

[4] BV Babu and Leenus Jehan. Differential evolution for multi-objective optimization. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 4, pages 2696–2703. IEEE, 2003.

[5] Kang-Ping Wang, Lan Huang, Chun-Guang Zhou, and Wei Pang. Particle swarm optimization for traveling salesman problem. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693)*, volume 3, pages 1583–1585. IEEE, 2003.

[6] Swagatam Das, Ajith Abraham, and Amit Konar. Automatic clustering using an improved differential evolution algorithm. *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans*, 38:218–237, 2008.

[7] Sandra Paterlini and Thiemo Krink. Differential evolution and particle swarm optimisation in partitional clustering. *Computational statistics & data analysis*, 50:1220–1247, 2006.

[8] Shih-Wei Lin, Kuo-Ching Ying, Shih-Chieh Chen, and Zne-Jung Lee. Particle swarm optimization for parameter determination and feature selection of support vector machines. *Expert systems with applications*, 35:1817–1824, 2008.

[9] Carlos Coello, Satchidananda Dehuri, and Susmita Ghosh. *Swarm Intelligence for Multi-objective Problems in Data Mining*. Springer Berlin, 2013.

[10] Konstantinos Parsopoulos and Michael Vrahatis. *Particle swarm optimization and intelligence: advances and applications*. IGI Global, 2010.

[11] Marco Dorigo and Stutzle Thomas. *Ant colony optimization.* MIT Press, 2004.

[12] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[13] Rainer Storn and Kenneth Price. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359, 1997.

[14] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence.*, pages 69–73. IEEE, 1998.

[15] Nateri Madavan. On improving efficiency of differential evolution for aerodynamic shape optimization applications. In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 4622. American Institute of Aeronautics and Astronautics, 2004.

[16] Paolo Rocca, Giacomo Oliveri, and Andrea Massa. Differential evolution as applied to electromagnetics. *IEEE Antennas and Propagation Magazine*, 53:38–49, 2011.

[17] Uday Chakraborty. *Advances in differential evolution.* Springer-Verlag Berlin Heidelberg, 2008.

[18] Swagatam Das, Amit Konar, and Uday Chakraborty. Two improved differential evolution schemes for faster global search. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 991–998. ACM, 2005.

[19] Deepak Dawar and Simone A. Ludwig. Effect of strategy adaptation on differential evolution in presence and absence of parameter adaptation: An investigation. *Journal of Artificial Intelligence and Soft Computing Research*, 8:211–235, 2018.

[20] David Koloseni. *Differential evolution based classification with pool of distances and aggregation operators.* PhD thesis, Lappeenranta University of Technology, 2015.

[21] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A

fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[22] Apache Hadoop- MapReduce. `https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html` , Accessed on Feb-10-2019.

[23] Sumita Kumar. *Apache Spark for Java Developers*. Packt Publishing Limited, 2017.

[24] Spark 2.1.0 documentation. `https://spark.apache.org/docs/2.1.0/`, , Accessed on Feb-10-2019.

[25] Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia. *Learning Spark: Lightning-Fast Big Data Analysis*. OReilly, 2015.

[26] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Peng, and Joshua Huang. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1:145–164, 2016.

[27] Ivanoe De Falco, Antonio Cioppa, and Ernesto Tarantino. Facing classification problems with particle swarm optimization. *Applied Soft Computing*, 7:652–658, 2007.

[28] Dervis Karaboga and Celal Ozturk. A novel clustering approach: Artificial bee colony (abc) algorithm. *Applied soft computing*, 11:652–657, 2011.

[29] Jayavelu Senthilnath, Subbarama Omkar, and V Mani. Clustering using firefly algorithm: performance study. *Swarm and Evolutionary Computation*, 1:164–171, 2011.

[30] Ivanoe De Falco, Antonio Cioppa, and Ernesto Tarantino. Automatic classification of hand-segmented image parts with differential evolution. In *Workshops on Applications of Evolutionary Computation*, pages 403–414. Springer, 2006.

[31] Pasi Luukka and Jouni Lampinen. A classification method based on principal component analysis and differential evolution algorithm applied for prediction diagnosis from clinical emr heart data sets. In *Computational Intelligence in Optimization*, pages 263–283. Springer, 2010.

[32] Chen Qiu, Liangxiao Jiang, and Ganggang Kong. A differential evolution-based method for class-imbalanced cost-sensitive learning. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.

[33] DW Van der Merwe and Andries Engelbrecht. Data clustering using particle swarm optimization. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, pages 215–220. IEEE, 2003.

[34] Mahamed Omran, Andries Petrus Engelbrecht, and Ayed Salman. Particle swarm optimization method for image clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, 19:297–321, 2005.

[35] Xiaohui Cui, Thomas Potok, and Paul Palathingal. Document clustering using particle swarm optimization. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 185–191. IEEE, 2005.

[36] Ruochen Liu, Xiaojuan Sun, Licheng Jiao, and Yangyang Li. A comparative study of different cluster validity indexes. *Transactions of the Institute of Measurement and Control*, 34:876–890, 2012.

[37] Ibrahim Aljarah and Simone A. Ludwig. A new clustering approach based on glowworm swarm optimization. In *2013 IEEE Congress on Evolutionary Computation*, pages 2642–2649. IEEE, 2013.

[38] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes university, engineering faculty, computer ..., 2005.

[39] Mete Çelik, Derviş Karaboğa, and Fehim Köylü. Artificial bee colony data miner (abc-miner). In *2011 International Symposium on Innovations in Intelligent Systems and Applications*, pages 96–100. IEEE, 2011.

[40] Rafael Parpinelli, Heitor Lopes, and Alex Freitas. Data mining with an ant colony optimization algorithm. *IEEE transactions on evolutionary computation*, 6:321–332, 2002.

[41] Peter Clark and Tim Niblett. The cn2 induction algorithm. *Machine learning*, 3:261–283, 1989.

[42] Prakash Shelokar, Valadi Jayaraman, and Bhaskar Kulkarni. An ant colony approach for clustering. *Analytica Chimica Acta*, 509:187–195, 2004.

[43] Kelly Robbins, Wensheng Zhang, Joseph Bertrand, and Romdhane Rekaya. The ant colony algorithm for feature selection in high-dimension gene expression data for disease classification. *Mathematical medicine and biology: a journal of the IMA*, 24:413–426, 2007.

[44] Nicholas Holden and Alex Freitas. Web page classification with an ant colony algorithm. In *International Conference on Parallel Problem Solving from Nature*, pages 1092–1102. Springer, 2004.

[45] Allen Chan and Alex Freitas. A new ant colony algorithm for multi-label classification with applications in bioinfomatics. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 27–34. ACM, 2006.

[46] Hidetomo Ichihashi, Katsuhiro Honda, Akira Notsu, and Keichi Ohta. Fuzzy c-means classifier with particle swarm optimization. In *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*, pages 207–215. IEEE, 2008.

[47] Min Chen and Simone A. Ludwig. Particle swarm optimization based fuzzy clustering approach to identify optimal number of clusters. *Journal of Artificial Intelligence and Soft Computing Research*, 4:43–56, 2014.

[48] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. `http://archive.ics.uci.edu/ml`, University of California, Irvine, School of Information and Computer Sciences.

[49] Ian Witten, Eibe Frank, Mark Hall, and Christopher Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[50] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11:10–18, 2009.

[51] Wissem Inoubli, Sabeur Aridhi, Haithem Mezni, Mondher Maddouri, and Engelbert Nguifo. An experimental survey on big data frameworks. *Future Generation Computer Systems*, 86:546–564, 2018.

[52] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In *IEEE International Conference on Cloud Computing*, pages 674–679. Springer, 2009.

[53] Simone A. Ludwig. Mapreduce-based fuzzy c-means clustering algorithm: implementation and scalability. *International journal of machine learning and cybernetics*, 6:923–934, 2015.

[54] Bowen Wang, Jun Yin, Qi Hua, Zhiang Wu, and Jie Cao. Parallelizing k-means-based clustering on spark. In *Advanced Cloud and Big Data (CBD), 2016 International Conference on*, pages 31–36. IEEE, 2016.

[55] Jie Yang and Xiaoping Li. Mapreduce based method for big data semantic clustering. In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, pages 2814–2819. IEEE, 2013.

[56] Ibrahim Aljarah and Simone A. Ludwig. Parallel particle swarm optimization clustering algorithm based on mapreduce methodology. In *Nature and biologically inspired computing (NaBIC), 2012 fourth world congress on*, pages 104–111. IEEE, 2012.

[57] Akhilesh Chunne, Uddagiri Chandrasekhar, and Chetan Malhotra. Real time clustering of tweets using adaptive pso technique and mapreduce. In *Communication Technologies (GCCT), 2015 Global Conference on*, pages 452–457. IEEE, 2015.

[58] Ibrahim Aljarah and Simone A. Ludwig. Mapreduce intrusion detection system based on a particle swarm optimization clustering algorithm. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 955–962. IEEE, 2013.

[59] Anan Banharnsakun. A mapreduce-based artificial bee colony for large-scale data clustering. *Pattern Recognition Letters*, 93:78–84, 2017.

[60] Ashish Kumar Tripathi, Kapil Sharma, and Manju Bala. A novel clustering method using enhanced grey wolf optimizer and mapreduce. *Big data research*, 14:93–100, 2018.

[61] Apache Spark - MinMaxScaler. `https://spark.apache.org/docs/2.1.0/ml-features.html#minmaxscaler`. Accessed on May-13-2019.

[62] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[63] Iman Sharafaldin, Arash Lashkari, and Ali Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *4th International Conference on Information Systems Security and Privacy (ICISSP)*, pages 108–116, 2018.

[64] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, pages 53–58. IEEE, 2009.

[65] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing, Second Edition*. Addison-Wesley, 2003.

[66] Pasi Luukka and Jouni Lampinen. Differential evolution classifier in noisy settings and with interacting variables. *Applied Soft Computing*, 11(1):891–899, 2011.

[67] Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[68] Rohollah Omidvar, Amin Eskandari, Narjes Heydari, Fatemeh Hemmat, and Sara Esmaeili. Data clustering using by chaotic sspco algorithm. *Majlesi Journal of Electrical Engineering*, 11(2), 2017.

[69] Roozbeh Razavi-Far, Vasile Palade, and Enrico Zio. Invasive weed classification. *Neural Computing and Applications*, 26(3):525–539, 2015.

[70] Ajith Abraham, Swagatam Das, and Amit Konar. Document clustering using differential evolution. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1784–1791. IEEE, 2006.

[71] Surveillance, Epidemiology, and End Results (SEER) Program (www.seer.cancer.gov) Research Data (1973-2015), National Cancer Institute, DCCPS, Surveillance Research Program, released April 2018, based on the November 2017 submission.

[72] Seer data. `https://seer.cancer.gov/data/` , Accessed on Dec-10-2018.

[73] Overview of the seer program. `https://seer.cancer.gov/about/overview.html` , Accessed on Jan-10-2019.

[74] Abdelghani Bellaachia and Erhan Guven. Predicting breast cancer survivability using data mining techniques. *Age*, 58:10–110, 2006.

[75] Elham Pour. *Stage-Specific Predictive Models for Cancer Survivability.* PhD thesis, The University of Wisconsin-Milwaukee, 2016.

[76] Elham Pour and Rohit Kate. Stage-specific survivability prediction models across different cancer types. In *AMIA Annual Symposium Proceedings*, page 1421. American Medical Informatics Association, 2017.

[77] Dursun Delen, Glenn Walker, and Amit Kadam. Predicting breast cancer survivability: a comparison of three data mining methods. *Artificial intelligence in medicine*, 34:113–127, 2005.

[78] Alaa Tharwat and Thomas Gabel. Parameters optimization of support vector machines for imbalanced data using social ski driver algorithm. *Neural Computing and Applications*, pages 1–14, 2019.

[79] Emel Kızılkaya Aydogan, Mihrimah Ozmen, and Yılmaz Delice. Cbr-pso: cost-based rough particle swarm optimization approach for high-dimensional imbalanced problems. *Neural Computing and Applications*, pages 1–19, 2018.

[80] Michela Antonelli, Pietro Ducange, and Francesco Marcelloni. An experimental study on evolutionary fuzzy classifiers designed for managing imbalanced datasets. *Neurocomputing*, 146:125–136, 2014.

[81] Urvesh Bhowan, Mark Johnston, Mengjie Zhang, and Xin Yao. Reusing genetic programming for ensemble selection in classification of unbalanced data. *IEEE Transactions on Evolutionary Computation*, 18(6):893–908, 2014.

[82] Alaa Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 2018.

[83] Chong Zhang, Kay Chen Tan, Haizhou Li, and Geok Soon Hong. A cost-sensitive deep belief network for imbalanced classification. *IEEE transactions on neural networks and learning systems*, (99):1–14, 2018.

[84] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann, 1995.

[85] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[86] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. 95(1-2):161–205, 2005.

[87] David Broomhead and David Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2(3), 1988.

[88] David Aha, Dennis Kibler, and Marc Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[89] Samiya Khan, Kashish Shakil, and Mansaf Alam. Big data computing using cloud-based technologies, challenges and future perspectives. *CoRR*, abs/1712.05233, 2017.

[90] Yanjie Wang and Quan Qian. A spark-based artificial bee colony algorithm for large-scale data clustering. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1213–1218. IEEE, 2018.

[91] Chi Zhou. Fast parallelization of differential evolution algorithm using mapreduce. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1113–1114. ACM, 2010.

[92] Diego Teijeiro, Xoán Pardo, Patricia González, Julio Banga, and Ramón Doallo. Implementing parallel differential evolution on spark. In *European Conference on the Applications of Evolutionary Computation*, pages 75–90. Springer, 2016.

[93] Changshou Deng, Xujie Tan, Xiaogang Dong, and Yucheng Tan. A parallel version of differential evolution based on resilient distributed datasets model. In *Bio-Inspired Computing-Theories and Applications*, pages 84–93. Springer, 2015.

[94] Andrew McNabb, Christopher Monson, and Kevin Seppi. Parallel pso using mapreduce. In *2007 IEEE Congress on Evolutionary Computation*, pages 7–14. IEEE, 2007.

[95] Goutham Miryala and Simone A. Ludwig. Comparing spark with mapreduce: Glowworm swarm optimization applied to multimodal functions. *International Journal of Swarm Intelligence Research (IJSIR)*, 9:1–22, 2018.

[96] Long Cui. Parallel pso in spark. Master's thesis, University of Stavanger, Norway, 2014.

[97] Meroua Daoudi, Soumiya Hamena, Zakaria Benmounah, and Mohamed Batouche. Parallel diffrential evolution clustering algorithm based on mapreduce. In *2014 6th International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, pages 337–341. IEEE, 2014.

[98] Mariem Moslah, Mohamed Ben HajKacem, and Nadia Essoussi. Spark-based design of clustering using particle swarm optimization. In *Clustering Methods for Big Data Analytics*, pages 91–113. Springer, 2019.

[99] Matthew Sherar and Farhana Zulkernine. Particle swarm optimization for large-scale clustering on apache spark. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2017.

[100] Kannan Govindarajan, David Boulanger, Vivekanandan Kumar, and Kinshuk. Parallel particle swarm optimization (ppso) clustering for learning analytics. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 1461–1465. IEEE, 2015.

[101] University of new brunswick - canadian institute for cybersecurity. `https://www.unb.ca/cic/datasets/ids-2017.html` , Accessed on Feb-10-2019.

[102] Josephine Akosa. Predictive accuracy: A misleading performance measure for highly imbalanced data. In *Proceedings of the SAS Global Forum*, 2017.