

POLITECNICO DI TORINO

Corso di Laurea magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

A classification algorithm based on Spark



Relatore

Prof Baralis Elena Maria

Prof Paolo Garza

Prof Cagliari Luca

Candidato

Abegaz Michael Wondimu

DEC 2017

ABSTRACT

With the development of the information technology, big data becomes a very popular word everywhere. Many companies have already captured value from big data and profit from it because of its advancement in its analytical technology.

The goal of this dissertation is to implement and analyze a classification algorithm called WAODE prediction algorithm by using spark on a distributed platform.

Application of machine learning algorithms like WAODE on big data sets often requires high amount of processing power and memory. Including such resources on single computer is usually a headache. Scientists come up with a new Idea called distributed computing which uses the computing power of many computers to alleviate this problem.

In this project a library is implemented that applies the WAODE algorithm, it accepts a training dataset and predict the class label of new unlabeled data. 10-fold cross validation is also implemented so that we can find which algorithm offers the best result.

ACKNOWLEDGEMENTS

My special gratitude goes to Prof. Paolo Garza, for the remarks and engagement through the making of my thesis, for making sure that I was provided with all the necessary material and equipment, for granting me to have the access to the Big Data server. I am also very thankful to my supervisor Cagliero Luca, whose professional guidance and follow-up during my project makes me understand and acquire a lot of knowledge in the subject. Finally I would like to thank all of those who has been with me and made this to happen.

Table of Contents

ABSTRACT.....	2
ACKNOWLEDGEMENTS.....	3
1. INTRODUCTION	8
2. BACKGROUND OF BIG DATA ANALYSIS.....	10
2.3. How Big Data Analysis Helps Companies (Placeholder1)	13
2.4. What makes analyzing Big Data difficult	14
2.5. Usage of Big Data Analytics	14
2.6. Advantages Big Data Analytics	15
2.7. Overview of Big Data Techniques	15
2.7.1 Classification.....	16
2.7.2 Clustering.....	16
2.7.3 Collaborative Filtering	17
3. CLASSIFICATION	18
3.1 Naive Bayes (NB).....	18
3.2.5. Weighted Averaged N Dependence Estimators (WAnDE)	19
4. DISTRIBUTED SYSTEMS	20
4.1. Hadoop.....	20
4.2. Apache Spark	21
5. WAOE, AnDE and WAnDE IMPLEMENTATION	23
5.1. Preprocessing.....	23
5.2. Interface	24
5.3. WAOE Implementation Process	24
5.4. AnDE Implementation Process	32
5.5. WAnDE Implementation Process	39
6. EVALUATION AND RESULTS.....	46
A. Experiment Dataset	46
B. Test Environment.....	47
C. Evaluation methodology	48
D Experimental Results	51
6.2. Conclusions and Future work	58
Bibliography	59

List of Tables

Table 1 UCI Datasets used for the testing the accuracy of the algorithms	46
Table 2 Polito Data Mining Cluster Environment	47
Table 3 . Measuring the accuracy of WAODE, AnDE, WAnDE estimators by using 10-fold cross validation methodology on selected UCIDatasets	51
Table 4 Average accuracy results of the (WAODE, A2de, WAnDE) versus WAODE centralized version (Rapid Miner) on selected UCIDatasets.....	52
Table 5 Average accuracy comparison of WOADE Distributed implementation vs WOADE centralized (Rapid Miner version).	53
Table 6 Average accuracy comparison of WAnDE with respect to 3 different M-estimator Implementations.....	54
Table 7 Average accuracy comparison of WAnDE with respect to 3 different Laplace Estimator Implementations	55
Table 8 Summery of experimental results of WAODE Implementations	56
Table 9 Summery of Experimental results for AnDE and WAODE	57
Table 10 Summery of experimental results for WAnDE implementation.....	57

List of Figures

Figure 1 Big data and analysis tools.....	10
Figure 2 Importance of Big Data analysis	14
Figure 4 Classification Example	16
Figure 5 Clustering Algorithm Example	16
Figure 6 Collaborative Filtering Example	17
Figure 9 Map Reduce Steps	21
Figure 10 Machine learning algorithms.....	22
Figure 11 7 Sample Log file showing 10-fold cross validation accuracy result for a dataset.....	50

Acronyms

AODE	Averaged One Dependence Estimator
WAODE	Weightedly Averaged One Dependence Estimator
AnDE	Averaged N Dependence Estimator
A2DE	Averaged 2 Dependence Estimator
WAnDE	Weighted Averaged One Dependence Estimator
GPS	Geographical Positioning System
NB	Naïve Bayes
RDD	Resilient Distributed Dataset
IP	Mutual Information
HDFS	Hadoop Distributed File System
ME	M Estimation
LE	Laplace Estimation

Chapter 1

1. INTRODUCTION

Big data is an Information Technology term which refers to high volume, velocity, variety. That means big data is a huge dataset with fast generated data and different types of data. Many companies like IBM and Google have already realized that there is huge potential value in big data. They all profit by advanced analyzing tools or innovating new business model to capture value from big data.

Big data analytics helps companies to make more informed business decisions by enabling Data Scientist, predictive modelers and other analytics professionals to analyze large volumes of transaction data.

Common datamining tools like Rapid Miner, Weka and other text analysis tools can analyze Big Data. We can also use visualization tools like “I2” to analyze and benefit from it. But the major drawback of this kinds of tools is that they are incapable of analyzing unstructured data or semi structured data. In addition, most current systems are Realtime which process continuous arrival of data F.eg social network websites and online Banking transactions. As a result, they need big processing power and resources.

Due to this challenges Hadoop related tools started to emerge in to the market of Big Data analysis. Big companies start to shift to such kind of Tools like **Spark** to process their big chunk of data. Today spark related tools became the main part of data analysis framework that can be used by developers and analysts. (bigdataanalytic, n.d.)

The Main purpose of this thesis is to implement a predictive analysis technique which is called WAODE on the distributed environment by using Spark and Hadoop. WAODE is a probabilistic classification model which is made to improve the accuracy of its predecessor algorithms like Naïve Bayes and AODE by weakening their attribute independence assumption and similar attribute weight assumption.

Today as we are in the information age Big Data is produced in vast quantity, speed and type from many things around us which we use for our day to day life F.eg Social media, our Smart phones, GPS sensors as well as Industrial productions. To process this vast amount of data we need very good analysis technology with high processing speed.

Thus, it is important to use Distributed computing which is the simultaneous use of multiple computing resources to solve a computational problem. Specially this is advantageous in machine learning and data analysis which uses repeated application of an algorithm to learn the data and give a precise prediction. (bigdataanalytic, n.d.)

The previous implementation of this algorithm was made using sequential programming which makes it insufficient when dealing with Big Data. As a result, the main objective of this Thesis is to implement WAODE algorithm by using Spark technology which will make it suitable for analyzing Big Data by increasing parallelism and efficiency. Since Efficient parallel and concurrent implementation techniques are needed to meet the scalability and performance requirements entailed by scientific data analyses. Challenges such as scalability and resilience to failure are already being addressed at the lower layer.

One type of distributed computing application is predictive analysis. Predictive analytics brings together advanced analytics capabilities spanning ad-hoc statistical analysis, predictive modeling, data mining, text analytics, entity analytics, optimization, real-time scoring, machine learning and more. (bigdataanalytic, n.d.)

Apache Hadoop is the pioneer in Big Data technology and it is the base framework for many Big Data technologies. As of today, it covers huge percentage of Big Data market. However, the need for faster processing and result requires in memory processing and data storage capabilities. Apache spark comes with new technology which allows it to process and store intermediate data in memory. Due to this in memory capability spark could process data up to 100 percent faster than apache Hadoop. as a result, spark becomes the favorable technology for faster processing requirements. (technopedia, n.d.)

Thus, the main contribution of this Thesis is to implement a predictive algorithm which is WAODE by using Spark to make it suitable for analyzing Big Data by increasing parallelism and efficiency. (scholar)

Any new Big Data technology usually must address two big issues in Big Data world.

The first challenge is **Diversity**.

● **Most Internet companies generate petabytes of data daily which was in terabytes few years ago.**

Besides textual data audio and Video streaming data comprises most part of the data acquired by internet companies. This includes video and audio with different formats, images, longitudinal information acquired by sensors and from our mobile phone. Among this only 20 % is a relational data. Big Data usually have 3 characters namely “volume”, “velocity” and “variety”.

Volume describes the relative size of data to the processing capability. According to Moors law today a large number may be 20 terabytes, but in 12 months 100 terabytes may constitute big data. To solve the volume problem, we need a technology that could store huge amount of data in elastic fashion as well as technologies with distributed querying capability that can generate Intime and meaningful information from our Big Data.

Velocity is the frequency at which data is produced, collected and shared. The ability to analyze, detect patterns, to identify relationships between different data’s generally any real-time analysis should be tuned with fast emerging data from sensors and click systems. It also creates diversified real-time analysis to engage users based on location, activity, profile etc.

Variety refers to the production of different types of data from Devices, sensors etc. in addition to relational data. This unstructured data such as videos, speech and language etc. make it more difficult to store and process by using relational storage systems. This kind of diversified data requires a new type analysis methods besides distributed storage and access mechanisms. (Heidelberg)

The second challenge is the **Richness** of analytics.

Big Data technologies require a new kind of techniques and combination to produce a valuable information form data stored in our servers which is very huge and diverse.

since we are in the information age, the range of tools we have for big data analysis today are too few to make a good analysis. we are obliged to make deep analysis relative to the amount of data we have. commonly used techniques like statistical analysis, geographical analysis, time line analysis etc. are not enough to make good decision making with our current business needs and engaging customers.

the emergence Big Data in varied and vast quantity lets companies and researchers to analyze their big data just like Big companies do. Furthermore, the strain that Big Data places on our network, infrastructure and server make it inevitable to outsource our Big Data analysis on the cloud.

By using cloud distributing computing we could possibly make real time analysis, sentiment analysis, entity extraction as well as other complex analysis to get intelligence and knowledge without any cost or hustle for the infrastructure or big clusters needed. In fact, we are also free from worrying about the administration and management costs for this clusters. This and other new approaches let researcher and companies in the domain to mine through huge dataset in much faster and efficient way. (Heidelberg)

2.2 What is Big Data Analysis

It is the method of discovering hidden knowledge and pattern by collecting and processing large dataset commonly known as Big Data. It helps companies and analysts to better understand the hidden knowledge with in the large amount of data as it also helps decision making process by identifying key data in the business and forecasting future outcomes.

The intent of big data analysis is knowledge which is found by analyzing the data. It has the goal of making organizations excel in their decision making by untapping unforeseen information. The primary source of input for the data analysis is data from enterprise programs and systems like business reports, log files, email exchanges, customer research data, call detail records etc. this and other sets of data's can be examined by professionals to make predictions, forecasting, classifying and in other decision-making areas in the business world.

The explosion of data in recent years makes relational databases less usable since Big Data requires a lot of features which cannot be fulfilled by this traditional system for example continuous arrival of data from sensors and internet of things may require large storage facility and variation of Big Data which can consist different formats, language may not fit well with the existing data types and formats. (webopedia, n.d.)

As a result, many big companies turned their face to Hadoop based systems which can analyze data across many clusters. Hadoop ecosystem including many technologies such

as yarn, spark, hive, pig became the preferred tools by most analysts for analyzing Big Data of varied types. (prabatech, n.d.)

2.3. How Big Data Analysis Helps Companies (Placeholder1)

Tom Davenport a lead researcher at UC Berkley made a research to understand how Big Data analysis helps companies. According to the study big data analysis integrates organizations data and helps to identify hidden opportunities that will result in good business decisions and prediction as well as high profit, Increased efficiency and minimized wastage of resources. According to the companies which participated in the research, they got benefits in the following 3 aspects. (prabatech, n.d.)

Cost reduction: The emergence of cloud based distributed analytics minimized the huge costs of memory and management. The need for large storage facility and software systems can be easily outsourced to cloud based systems.

Faster decision making: Sparks In-memory processing capability and additional features like supporting various data sources helps companies to analyze information quickly and make a faster decision on business problems.

New products and services: According to Davenport's research Big Data analysis techniques like Association matrix helped companies measure customer satisfaction level and identify customer expectations easily. hence, providing newer products and services.



Figure 2 Importance of Big Data analysis

2.4. What makes analyzing Big Data difficult

The primary challenge of Big Data analysis is the volume and variety of the collected data by different units in the organization. Different units in the company could have different formats or types of data according to their business functionality. This challenge requires dividing data into different domains to understand the collected data from different places and different sub systems. (researchIjcaOnline)

Another big data challenge is the mechanism of processing this unstructured data to make it more structured and responsive for data analysis algorithms. often this involves several stages into which the data can be cleaned, transformed and analyzed to give useful information. This large volume of data is typically difficult to process using traditional data storage and analysis tools. (SasInsights, n.d.)

2.5. Usage of Big Data Analytics

Studies by DataMation shows that recent advancement of Big Data analysis and its application in organizational data allows huge transformation and breakthrough in

today's modern world. Big Data analysis helped scientists to identify human genes which are related with certain diseases easily. Predict criminal activities and vulnerable areas, Redirect advertisements based on user's activity and tendency etc.

Another good application of Big Data comes from research made by orange mobile which is a leading mobile network provider in France. The analysis involves 2 billion records of customer data belonging to 2.5 million subscribers. The customer data contains all exchanged text and voice messages except the subscriber information. The research result shows very good information on how to contain disease by using cellular data and identifies where peoples will go after emergencies. this and other information gathered from deep analysis of the research helped to shape the health infrastructures and development projects according to the research outcome. (webopedia, n.d.)

2.6. Advantages Big Data Analytics

The main purpose of Big Data projects arises from the need to address biggest questions in day today business environment. The adoption of Big Data analysis systems enables enterprises to boost profit, improve efficiency, enhance operations and provide good customer services and avoid risk.

Online survey from different organizations in the Big Data industry shows that they make use of Big Data analytics to target customer, improve production quality and helps them compete with other companies on similar sector. Obviously, the application of data analytics on certain areas made great difference regarding efficiency and minimizing cost. The research showed that most of the organizations will use data analytics to speed up production and reduce complexity. (webopedia, n.d.)

2.7. Overview of Big Data Techniques

Machine learning is a technique of data analysis which uses algorithms that iteratively learn from data to generate analytical model of the problem. machine learning has the power to identify hidden knowledge without being explicitly programmed where to look. In general, there are two broad categories of machine learning: supervised and unsupervised

Supervised algorithms need both input and output data provided in advance to make the analytical model of the problem. While Unsupervised algorithms do not have the outputs of the data in advance as a result Unsupervised algorithm work by making good analysis of the input data to provide output of their own.

The Three main categories of machine learning are Clustering, Classification, and Collaborative Filtering.

2.7.1 Classification

Classification machine learning algorithms belong to a class of supervised learning. This algorithm can label an input data which was not previously labeled. Some applications of classification are categorization of input emails as social, important and advertisement on Gmail application is one of the popular example of classification

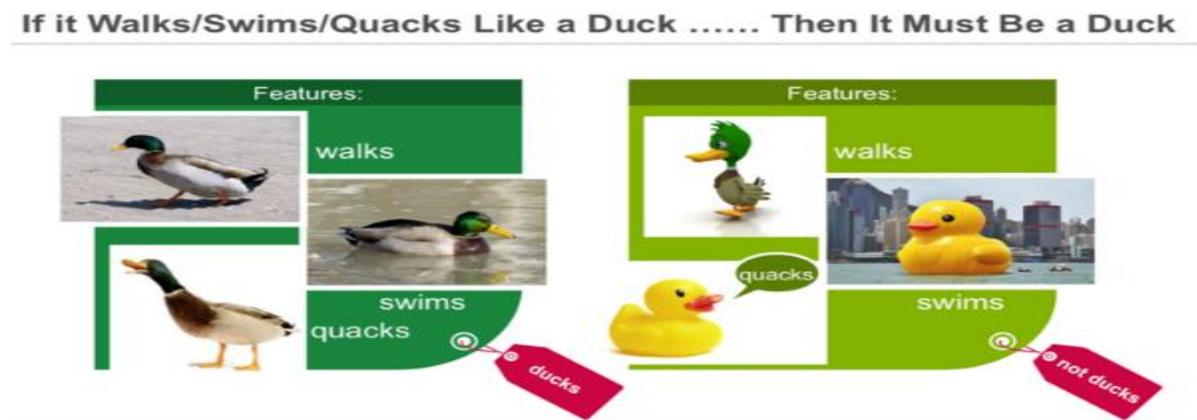


Figure 3 Classification Example

2.7.2 Clustering

Clustering algorithms group inputs in to set of categories based on their similarities. Clustering groups inputs with similar characters together and dissimilar inputs will have different groups. Clustering is widely used in technology and different science areas f.eg Text categorization, Fraud detection, grouping of users based on their activities, categorizing search results on popular search engines etc.

clustering algorithms do not know the output in advance nor they have input labels in advance



Figure 4 Clustering Algorithm Example

2.7.3 Collaborative Filtering

Also known as recommender systems. This set of algorithms use association learning to recommend specific items usually based on past preferences from similar customers. Collaborative filtering algorithms take preference data from customers and create analytical model that will be used to recommend future users. This set of techniques are mostly used on online shopping websites like amazon. For example, previous shopping data may show an association between purchase of suits and purchase of ties, based on this result amazon could recommend an incoming user to purchase ties if he is already purchasing suit. It is also widely used on google advertisements to show targeted ads.

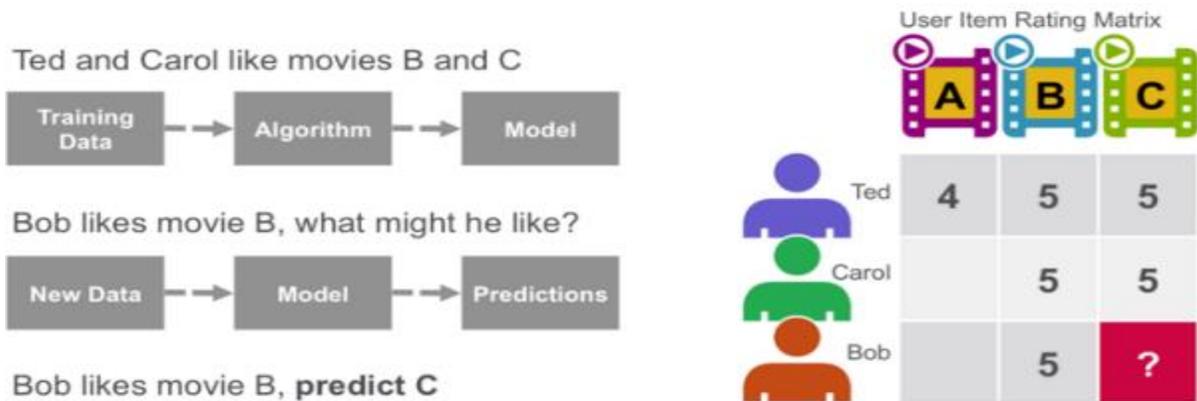


Figure 5 Collaborative Filtering Example

Chapter 3

3. CLASSIFICATION

We make prediction throughout our life from weather forecasting to crop production, money exchange and so many others with different accuracy. Thus, prediction helps to conclude what will happen in the future starting from current situations or past events. There are various things which should be predicted professionally to avoid risk or to increase profit for example stock exchanges, natural disasters etc. The following are popular classification algorithms which the thesis paper concerns

3.1 Naive Bayes (NB)

Naïve Bayes is a classification method which is based on the Bayesian theorem is simple and very efficient which also helped it to be widely applied in bigdata technology. This classification technique analyses the relationship between each attribute and the class for each instance to derive a conditional probability for the relationship between the attribute values and the class. The opinion behind Naïve Bayes for classification is a simple.

In training stage, the probability table for each class is computed firstly by counting the frequency of the class (how frequently it occurs in the dataset). This is called the prior probability $P(c)$. (springer)

Assuming $A_i, I = 1, 2, \dots, n$, which are n attributes with values $a_i, I = 1$ up to n respectively. These attributes will be used together to predict the class label c of the class attribute C . Hence, the NB classifier can be formed as:

$$\text{Argmax } P(c) P(a_1, a_2, \dots, a_n | c)$$

Secondly, Attribute probability table will be created, the algorithm computes the probability for the instance a given a and class c , $P(a_i | c)$. With the assumption that all attributes are independent the probability will be the product of the probabilities of each single attribute. The probabilities are estimated from the frequencies of the instances in the training set.

Thus, Naïve Bayes classifier will be formed according to the following formula

$$\underset{c \in C}{\text{argmax}} P(c) \prod_{i=1}^n P(a_i | c)$$

Naïve Bayes is known to be extremely efficient. However, the assumption that all attributed are independent with each other makes it less usable since in most real-life scenarios attributes could be dependent on each other.

3.2.5. Weighted Averaged N Dependence Estimators (WAnDE)

It has been proven that the predictive accuracy of AnDE can be improved with the introduction Weight on the AnDE (Averaged n Dependence estimators) estimators. WAnDE retains most of the characteristics like computational complexity. It shows that similar approaches could be applied on to improve the performance.

For notational convenience, we define

$$X_{\{i,j,\dots,q\}} = (x_i, x_j, \dots, x_q)$$

For example, $x_{\{2,3,5\}} = (x_2, x_3, x_5)$

WAnDE aims to use

$$WAnDE(Y, X) = \sum_{S_n} W_s P(Y, X_s) P(X|Y, X_s) / W_{tot}$$

where S_n indicates all subsets of size n of the set $\{1, \dots, a\}$.

Chapter 4

4. DISTRIBUTED SYSTEMS

4.1. Hadoop

It's a "software library" that gives users the ability to process "large data sets across clusters of computers using simple programming models." In other words, it gives companies the capability to gather, store and analyze huge sets of data. (Hadoop, n.d.)

Advantages of using Hadoop

- Hadoop is a pioneer in distributed systems which makes it more stable than other newer systems.
- It is open source and platform independent since it is made in java.
- Hadoop is hardware independent and it detects errors at application level rather than on lower levels.
- It has many functionalities which helps to perform Distributed tasks easily.
- It manages to distribute data across clusters while also using CPU parallelism offered by the underlying hardware.

By implementing Hadoop, users gain access to an amazing number of tools and resources that allow them to truly personalize their big data experience to fit whatever their business needs may be. (Hadoop, n.d.)

The project includes these modules:

Hadoop Common: The core Hadoop utilities that help users to perform many functionalities

YARN: The main module that integrates and manages the clusters. It also schedules tasks.

HDFS: The file system behind the Hadoop which manages access to Distributed data

Hadoop MapReduce: Yarn based functionalities that help to process huge datasets in parallel. HDFS (Hadoop Distributed File System) which is based on GFS (Google File System) helps to transfer data in a faster way between nodes in the cluster in a fault tolerant manner. As a result, it does not stop functioning even if one node fails. HDFS allows parallel processing by partitioning data in small chunks across the cluster. It duplicates the same data on multiple computers which enables it to continue working even if some data is corrupted or damaged in case of failure

It can accommodate very large datasets up to petabytes which makes it ideal for Big-data applications. (apache, n.d.)

4.2. Apache Spark

Apache Spark is a general compute engine that offers fast data analysis on a large-scale dataset. Spark is built on HDFS but bypasses MapReduce and instead uses its own data processing framework. Common uses cases for Apache Spark include real-time queries, event stream processing, iterative algorithms, complex operations and machine learning.

Spark MapReduce Framework

MapReduce which is the most widely used technique to process data in parallel is born because of a research at Google back in the early 2000's. According to the research paper it is concluded that most parallel tasks for a distributed computing can be compiled as a Map and Reduce actions. Map actions usually apply same operation on each record of a dataset like filtering while the Reduce actions summarize the results which are computed in the early stages of Map. In fig 1 Input data is partitioned in to multiple parts and distributed across clusters. The Map stage begins parallel processing of the data by applying same tasks in each partition. The 2nd phase which is the reduce stage merges each part of the data across the cluster to give a single output. (mdpi, n.d.)

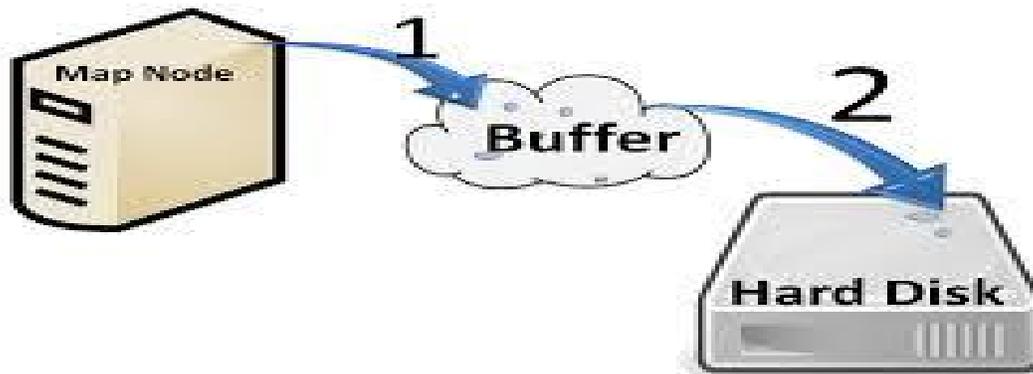


Figure 6 Map Reduce Steps

Map reduce is a very powerful platform in a way that programs based on it can run parallelly on thousands of computers. Programmers can use the advantage of MapReduce to run tasks parallelly on multiple nodes even if they don't have any knowledge of the underlying distributed system infrastructure.

Spark Platform

Spark is another kind of distributed programming platform developed at DataBricks. It processes most of Map Reduce tasks in memory as a result it is faster than Hadoop MapReduce which is disk based. Spark's core logical unit is called RDD (Resilient Distributed Datasets) which is parallelized across clusters to perform distributed tasks. Spark can be developed by using java, python or Scala programming languages. Spark's in-memory computation capability makes it ideal for processing iterative algorithms like WAODE, which would take large computational time if processed on Disk. Spark cluster works as a Master and Slave (Worker nodes). The worker nodes perform the Map reduce tasks parallelly while the Master node coordinates the tasks and allocates resources to the worker nodes. The master also performs error Management, scheduling and other tasks. (mdpi, n.d.)

Machine Learning with Spark

Machine learning is a method of data analysis that enables computers to get hidden information without being explicitly instructed. It learns incrementally to build an analytical model of the data. (mdpi, n.d.)

MLLIB

It is a set of readymade machine learning algorithms that perform classification, clustering as well as other important tasks. MLlib provides programming interfaces that help to use these algorithms easily.

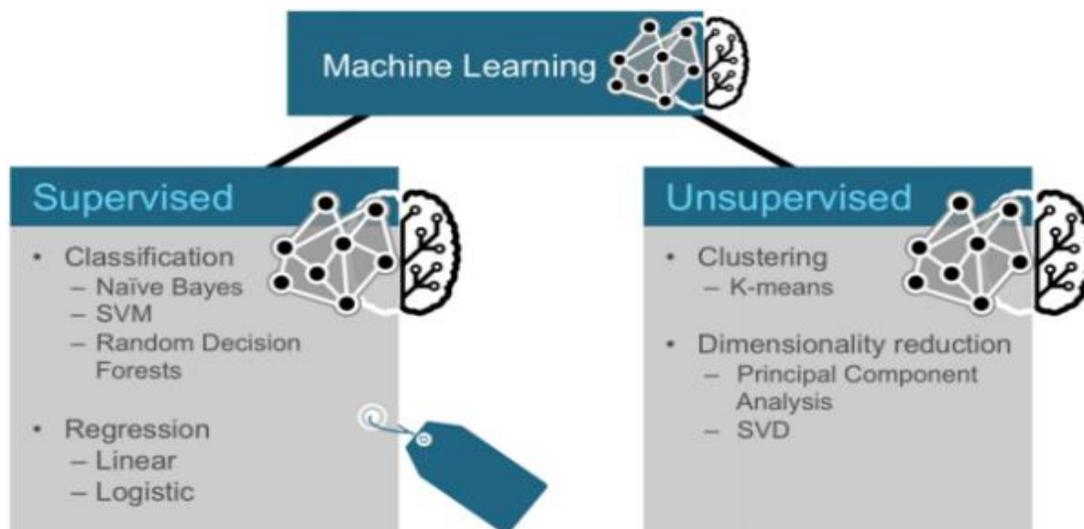


Figure 7 Machine learning algorithms

Chapter 5

5. WAODE, AnDE and WAnDE IMPLEMENTATION

WAODE (Weightedly averaged one dependence estimators) is experimentally tested to show high predictive accuracy on large datasets. However, implementing WAODE with common procedural programming makes it time consuming and impractical for very large datasets. overcoming this problem requires implementing the parallel version of the algorithm with technologies like **Spark**.

Historically, the possible ways to implement parallel programming was by using multiple core CPUs, however this kind technique is very complex programmatically and it is dependent of the number of CPUs on the computer. Another possible way of parallel implementation is by using Hadoop MapReduce platform which a pioneer in this kind of computing. Hadoop MapReduce can solve many of computational problems like partitioning, scheduling tasks and distributing parallel jobs across cluster etc. However, Hadoop MapReduce also have a major drawback of storing intermediate results and stages on disk. This makes the read write tasks to take large amount of time specially for Iterative algorithms like WAODE. (mdppi, n.d.)

Sparks parallel programming architecture which uses RDD is suitable for iterative algorithms in such a way that computation and intermediate results are stored in memory and we can obtain both good performance and high predictive accuracy. In this chapter we focus on the implementation of WAODE, AnDE and WAnDE by using spark.

5.1. Preprocessing

The **Main** method of the System first parallelizes the Input File by using *sparkContext.Textfile* method which will convert it into RDD format as a result it enables spark to process the data parallely across clusters. Furthermore, this step splits the Input-RDD in to tokens by based on the coma separator.

Finally, this RDD will be passed to the algorithm as a Training Data Set. Caching helps to store the generated RDD on memory so that it does not process the input file for every classification step.

Pseudocode for processing input dataset

1. *filePath = Location of the training data on disc*
2. *Input_RDD = sc.Textfile(filePath);*
3. *Training_RDD = RDD_input.Map(x → x.split(","))*

4. *Training_RDD.Cache ();*

5.2. Interface

The WAODE, AnDE and WAnDE libraries generally implement the **Estimator** Interface. As a result, they all inherit **CreateModel** and **Predict** Method which could perform the Training and Classifications stage accordingly.

These functions have public modifier which allows them to be accessed from outside objects.

```
WAODE_Estimator estimator = new WAODE_Estimator(sc)
  estimator.CreateModel ("/home/mike/AdultDataset.csv")
String class = estimator.Predict ("Private, ignore, HS-grad, Divorced
  Handlers-cleaners Not-in-family White Male -57")
```

```
AnDE_Estimator estimator = new AnDE_Estimator(sc)
  estimator.CreateModel ("/home/mike/nurseryDataset.csv")
String class = estimator.Predict ("usual, proper, complete,1, convenient, ")
```

```
WAnDE_Estimator estimator = new WAnDE_Estimator(sc)
  estimator.Train ("/home/mike/germanDataset.csv")
String class = estimator.Predict ("no-account, radio-tv, unknown, married-male, none,
  ignore, building-society")
```

5.3. WOADE Implementation Process

The WOADE algorithm is essentially a parallel algorithm like many machine learning algorithms, The WOADE algorithm can be divided into two stages which are namely Training and classification. At the first stage it iterates through all Attribute columns to calculate constants, weight values, frequency and probability estimates.

Finally, it saves the results as a model to use it for the 2nd stage of the algorithm.

In the second stage, which is the Classification stage WOADE predicts the class by averaging the aggregate of weighted one dependence classifiers to the Total weight value. These two stages clearly show that they can be implemented using Spark MapReduce platform.

The implementation of WAODE based on Spark map reduce platform is as follows

Constructor

The constructor of the WAODE library accepts the spark context as input parameter. It initializes the sparkContext, Hash Tables and Lists which will be used for storing intermediate data.

List for variables and HashTable to initialize

1. *Initialize memory for clsList*
2. *Initialize memory for WMap*
3. *Initialize memory for distItemsMap*
4. *Initialize memory for itemFrqMap*
5. *Initialize memory for jointProbMap*
6. *Initialize memory for tripleProbMap*

finally, it calls the **CreateModel** Method which uses to train the dataset accordingly

CreateModel

The CreateModel method of **WAODE** algorithm creates the classification model by accepting **Training Dataset** as an argument.

Pseudocode for Training Phase of WAODE Algorithm

1. ***function CreateModel (Training RDD)***
2. ***Input: Training RDD***
3. ***Output: joint frequency and joint Probability of each attribute with class***
4. $N = \text{Training_RDD.Count}$
5. $N\text{Broadcast} = \text{sc.broadcast}(N);$
6. $\text{DistinctItems_RDD} [i, \text{val}] = \text{Training_RDD.FlatMapToPair} (\text{funcPairDist}).$
 $\text{distinct} (). \text{cache} ()$
7. $V_Map ("i", v) = \text{DistinctItems_RDD.countByKey} ()$
8. $K = V\text{Map.get}("iclass")$
9. $v\text{MapBroadcast} = \text{sc.Broadcast}(V_Map)$
10. $k\text{MapBroadcast} = \text{sc.Broadcast}(K)$
11. $\text{DistinctItems_Map} = \text{DistinctItems_RDD} [i, \text{val}]. \text{groupByKey} (). \text{collectAsMap} ()$
12. $\text{cls-List} = \text{DistinctItems_Map.get}(i\text{-class})$
13. $\text{DistinctItems_RDD.unpersist}()$

 $// \text{Calculating Item Frequency and Probability}$

```

14. itemFrequencyRDD=Training_RDD.FlatMapToPair (funcPairSingle).
    reduceByKey(funcSumm)
15. jointFrequencyRDD = Training_RDD.flatMapToPair(funcPairJoint).
    reduceByKey(funcSumm)
16. tripleFrequencyRDD = Training_RDD.flatMapToPair(funcPairTriple).
    reduceByKey(funcSumm)
17. jointProbRDD = jointFrequencyRDD. MapToPair(funcJointProbablity)
18. tripleProbRDD = jointFrequencyRDD. MapToPair(funcTripleProbablity)

    // Collection of Results into HashMap

19. itemFrqMap = itemFrquencyRDD.collectAsMap()
20. jointFrqMap = jointFrquencyRDD.collectAsMap()
21. jointFrqMapBroadcast = sc. Broadcast(jointFrqMap)
22. jointProbMap = jointProbRDD.collectAsMap()
23. tripleProbMap = tripleProbRDD.collectAsMap()
24. jointProbMap = jointProbRDDNew.collectAsMap()
25. FOR I ← 1 TO m

        weightI ← calculateWeight(I)

        WeightMap.put (I, weightI)

    END FOR

26. Training_RDD.unpersist();
27. End    // Training completed

```

Description for WAODE CreateModel procedure

Line 1-3 CreateModel function accepts Training RDD as parameter and saves output in to HashMap.

Line 4 - 5: It calculates N which is the number of records in the dataset and saves it in to a broadcast variable so that it can be accessible from all nodes in the cluster.

Line 6: finds distinct items in every attribute node and maps it with its attribute index (since same items could exist in another attribute node)

Line 7: calculates V_i which is the number of distinct items in each attribute nodes

Line 8: calculates K which is the number of distinct classes in the class node.

Line 9: saves V_i in to broadcast variable so that it can be accessible form other nodes in the cluster.

Line 10: saves K in to broadcast variable so that it can be accessible form other nodes in the cluster.

Line 11: groups items with similar key to find distinct items with in the same attribute node and finally it collects them in to HashMap

Line 12: gets distinct classes from class attribute

Line 13: unloads the *DistinctItems_RDD* from memory since we don't need it anymore.

Line 14: Maps each item with 1 and finally adds them together to calculate Item frequency (the number of times an item appears in the dataset).

Line 15: Maps each combination of attribute and class with 1 and finally adds them together to calculate Joint frequency of Attribute and Class (The number of times an item and class combination appears in the dataset)

Line 16: Maps combination of 2 attributes I and J and class with 1 and finally adds them together to calculate Triple frequency which is the frequency of Two Attributes and Class (The number of times the combination of attribute item 1, attribute item 2 and class appears in the dataset)

Line 17: Calculates Joint probability from JointFrequency (a_i , class) by using implemented Probability functions (The probability functions could be M-Estimation or Laplace estimate).

Line 18: Calculates Triple probability (a_i , a_j , class) from Triple Frequency by using implemented Probability functions (The probability functions could be M-Estimation or Laplace estimate).

Line 19-24: Collects the calculated frequency and probability values in to HashMap. This helps to avoid calculating the values again for every prediction.

Line 25: Calculates the weight of each attribute node by using the implemented Weighting function. It finally saves the weight of each attribute node in a HashMap.

Line 26: Un Persists the Training dataset RDD to free up memory.

Line 27: End of procedure

Methods (Closures) used at the Training phase of WAODE

// Maps each attribute value with the index of attribute

1. *function funcPairDist (String x)*

FOR I ← 1 TO m

 New Tuple2 <" i", val>

END FOR

end

//Maps each attribute value and Index with One

2. *function funcPairSingle (String x)*

FOR I ← 1 TO m

 New Tuple2 <" i, val", 1>

END FOR

end

//Maps each attribute value, index and class with one

3. *function funcPairJoint (String x)*

FOR I ← 1 TO m

 New Tuple2 <" i, val, class", 1>

END FOR

end

//Maps the combination of 2 attribute values with class

4. *function funcPairTriple (String x)*

FOR I ← 1 TO m-1

 FOR I ← 1 TO m

 New Tuple2 <" i, val1, j, val2, class", 1>

 END FOR

END FOR

End

```

//Adds the 2 occurrences for attribute values
5. function funcSumm (Long val1, Long val2)
    return val1+ val2
end
//Joint probability for two attributes based on M-estimation
6. function funcJointProbablity (ai, c) M-estimation
    jointFrqMap (ai, c) + (1.0 / (Vi * K))) / (N +1)
    Return P
end
//Triple probability of 2 attributes and class based on M-estimation
7. function funcTripleProbablity (ai, aj, c)
    tripleFrqMap (ai, c) + (1.0 / Vj) / (jointFrqMap (ai, c) +1)
    Return P
end
//Calculates the weight of an attribute
8. function funcWeightI (i)
FOR EACH attribValue ai, classValue c ∈ AI*, C*
     $fa_i \leftarrow \text{itemFrequency\_Map}("a")$ 
     $fc \leftarrow \text{itemFrequency\_Map}("c");$ 
     $pa \leftarrow fa / N$ 
     $pc \leftarrow fc / N$ 
     $Nom \leftarrow \text{jointProbMap}("a, c")$ 
     $Den \leftarrow pa \times pc$ 
     $W_i \leftarrow W_i + \text{jointProbMap}("a, c") \times (\text{Log Nom} - \text{Log Den})$ 
Weight_Map (i, Wi)
End FOR
end

```

Predict Method

The predict method of the class accepts a vector data (which is the list of attribute values without a class label). The predict method returns the predicted class label for the given vector.

Pseudo code for Prediction Phase of WAODE algorithm.

1. *function CalculateArgMax ()*
2. **Input:** list of attribute tokens and class label
3. **Output:** Estimation result
4. $Est \leftarrow 0$
5. FOR I \leftarrow 1 TO m
6. $P_{mul} \leftarrow 1$
7. key \leftarrow “ a_i, c ”
8. $P(a_i, c) \leftarrow$ jointProbMap. Get(key)
9. For J \leftarrow 1 To n
10. If j Not Equals i
11. key \leftarrow “ a_i, a_j, c ”
12. $P(a_i, a_j, c) \leftarrow$ tripleProbMap. Get(key);
13. $P_{mul} \leftarrow P_{mul} \times P(a_i, a_j, c)$
14. End FOR
15. $Est \leftarrow Est + W_i \times P(a_i, c) \times P_{mul}$
16. END FOR
17. $Est \leftarrow Est / W_{sum}$
18. Return Est
19. end

Description of the WAODE Argmax procedure

Line 1-3: Argmax method accepts the attribute vector and class label as an input

Line 4: Initializes Est variable to 0

Line 5: Loops through all attribute values in the input vector

Line 8: lookup for the joint probability of attribute a_i and class c from HashMap

Line 9-12: loops through all attribute values except the parent attribute and lookup for the joint probability of attribute a_i, a_j and class termed as triple probability ($a_i, a_j, class$).

Line 13: aggregates the triple probabilities calculated in the previous steps for each attribute value combinations.

Line 15: multiplies the aggregate of triple probabilities with the Joint probability of the parent attribute and the Weight of the Parent Attribute (looks up from HashMap). This Estimate value will be summed up for each parent attribute.

Line 17-18: The Total estimate value will be averaged with the Total weight value to return the estimation for the given class label.

Line 19: This procedure will be repeated for each class label. The class label with the maximum estimation will be predicted for the given input vector

Line 20: end of procedure

5.4. AnDE Implementation Process

In this thesis project only one version of AnDE which is A2DE is implemented. where N refers to the number of dependent attributes with the class. Unlike WAODE which assumes 1 dependent attribute with the class the A2DE algorithm uses 2 dependent attributes. As a result, it is expected to increase the accuracy of the prediction.

The main purpose of implementing A2DE is to understand the characteristics of the algorithm with respect to the number of dependent attributes. The AnDE like other machine learning algorithms can be divided into two stages. In the first stage, which is the Training stage it calculates the constants, joint frequency values and base probability estimates. Finally, this data will be saved to be used as a classifier for the 2nd stage of the algorithm.

At the second stage, which is the Prediction AnDE predicts the class by averaging the aggregate of one dependence classifiers to the Total number of records which is N. AnDE compares the estimation for each class and takes the maximum to give the output. The following subtopics discuss how to implement these two stages by using spark.

The implementation of AnDE based on Spark map reduce platform is as follows.

Constructor

The constructor of the WOADE library accepts the spark context as a parameter. It initializes the sparkContext, Hash Tables and Lists which are used for storing data. which are namely.

- Initialize clsList variable
- Initialize HashMap WMap
- Initialize HashMap distItemsMap
- Initialize HashMap itemFrqMap
- Initialize HashMap jointProbMap
- Initialize HashMap tripleProbMap

finally, it calls the **CreateModel** Method which uses to train the dataset accordingly

CreateModel

The CreateModel method of **AnDE** algorithm creates the classification model and by accepting **InputRDD** as an argument.

Pseudo Code for Training Phase of AnDE Algorithm

1. **function CreateModel** (Training RDD)
2. **Input:** Training RDD
3. **Output:** joint frequency and joint probability values
4. $N = \text{Training_RDD.Count}$
5. $N\text{Broadcast} = \text{sc.broadcast}(N)$;
6. $\text{DistinctItems_RDD} (I, \text{distValsList}) = \text{Training_RDD.FlatMapToPair}(\text{funcPairDist}).\text{distinct}().\text{cache}()$.
7. $V_Map ("i", v) = \text{DistinctItems_RDD.countByKey}()$
8. $K = \text{VMap.get}("iclass")$
9. $v\text{MapBroadcast} = \text{sc.Broadcast}(V_Map)$
10. $k\text{MapBroadcast} = \text{sc.Broadcast}(K)$
11. $\text{DistinctItems_Map} = \text{DistinctItems_RDD}().\text{reduceByKey}(\text{funcCollect})$.
12. $\text{cls-List} = \text{DistinctItems_Map.get}(i\text{-class})$.
13. $\text{DistinctItems_RDD.unpersist}()$
14. $\text{itemFrequencyRDD} = \text{Training_RDD.FlatMapToPair}(\text{funcPairSingle}).\text{reduceByKey}(\text{funcSumm})$
15. $\text{jointFrequencyRDD} = \text{Training_RDD.flatMapToPair}(\text{funcPairJoint}).\text{reduceByKey}(\text{funcSumm})$
16. $\text{jointFrequencyRDDNew} = \text{Training_RDD.flatMapToPair}(\text{funcPairJointNew}).\text{reduceByKey}(\text{funcSumm})$
17. $\text{tripleFrequencyRDD} = \text{Training_RDD.flatMapToPair}(\text{funcPairTriple}).\text{reduceByKey}(\text{funcSumm})$
18. $\text{jointProbRDD} = \text{jointFrequencyRDD.MapToPair}(\text{funcJointProbablity})$
19. $\text{tripleProbRDD} = \text{jointFrequencyRDD.MapToPair}(\text{funcTripleProbablity})$
20. $\text{jointProbRDDNew} = \text{jointFrequencyRDD.MapToPair}(\text{funcJointProbablityNew})$
21. //Collect Results into Map
22. $\text{itemFrqMap} = \text{itemFrquencyRDD.collectAsMap}()$
23. $\text{jointFrqMap} = \text{jointFrquencyRDD.collectAsMap}()$
24. $\text{jointFrqMapBroadcast} = \text{sc.Broadcast}(\text{jointFrqMap})$
25. $\text{jointProbMap} = \text{jointProbRDD.collectAsMap}()$
26. $\text{tripleProbMap} = \text{tripleProbRDD.collectAsMap}()$
27. $\text{jointProbMapNew} = \text{jointProbRDDNew.collectAsMap}()$
28. end

Description of AnDE CreateModel procedure

Line 1-3 CreateModel function accepts Training RDD as parameter and saves output in to HashMap.

Line 4 - 5: It calculates N which is the number of records in the dataset and saves it in to a broadcast variable so that it can be accessible from all nodes in the cluster.

Line 6: finds distinct items in every attribute node and maps it with its attribute index (since same items could exist in another attribute node)

Line 7: calculates V_i which is the number of distinct items in each attribute nodes

Line 8: calculates K which is the number of distinct classes in the class node.

Line 9: saves V_i in to broadcast variable so that it can be accessible form other nodes in the cluster.

Line 10: saves K in to broadcast variable so that it can be accessible form other nodes in the cluster.

Line 11: groups items with similar key to find distinct items with in the same attribute node and finally it collects them in to HashMap

Line 12: gets distinct classes from class attribute

Line 13: unloads the *DistinctItems_RDD* from memory since we don't need it anymore.

Line 14: pairs each item with 1 and finally adds them together to calculate Item frequency (the number of times an item appears in the dataset).

Line 15: pairs each combination of 2 attribute values and class with 1 (a_i, a_j, class). finally adds them together to calculate Joint frequency of Attributes (i, j) and class (The number of times the items and class combination appears in the dataset)

Line 16: pairs combination of 2 attributes (a_i, a_j) with 1 and finally adds them together to calculate JointFrequency of two Attributes values (The number of times the combination of attribute item 1, attribute item 2 appears in the dataset)

Line 17: pairs combination of 3 attribute values a_i, a_j and a_z with class. finally adds them together to calculate the triple frequency of Attributes (i, j, z) and class (The number of times the items and class combination appears in the dataset)

Line 18: Calculates Joint probability (a_i, a_j, class) from JointFrequency of (a_i, a_j, class) by using implemented Probability functions (The probability functions could be M-Estimation or Laplace estimate).

Line 19: Calculates Triple probability ($a_i, a_j, a_z, \text{class}$) from Triple Frequency ($a_i, a_j, a_z, \text{class}$) by using implemented Probability functions (The probability functions could be M-Estimation or Laplace estimate).

Line 20: Calculates the Joint probability (a_i, a_j) from JointFrequency of (a_i, a_j) by using implemented Probability functions (The probability functions could be M-Estimation or Laplace estimate)

Line 22-27: Collects the calculated frequency and probability values in to HashMap. This helps to avoid calculating the values again for every prediction.

Line 28: end of procedure

Methods (Closures) used in the algorithm

// Maps each attribute value with the index

1. *function funcPairDist (String x)*

FOR I ← 1 TO m

New Tuple2 <" i", val>

END FOR

end

// Maps each attribute value and Index with One

2. *function funcPairSingle (String x)*

FOR I ← 1 TO m

New Tuple2 <" i, val", 1>

END FOR

end

// Maps each attribute value, index and class with one

3. *function funcPairJoint (String x)*

FOR I ← 1 TO m-1

FOR J ← 1 TO m

New Tuple2 <" i, val, j, val_J, class", 1>

END FOR

END FOR

END

// Maps the combination of attribute values with one

4. function funcPairJointNew (String x)

FOR I ← 1 TO m-1

FOR J ← 1 TO m

New Tuple2 <" i, val, j, val_J", 1>

END FOR

END FOR

End

// Maps combination of attribute values and class with 1

5. function funcPairTriple (String x)

FOR I ← 1 TO m-2

FOR J ← 1 TO m-1

FOR K ← 1 TO m

New Tuple2 <" i, val_i, j, val_j, z, val_z, class", 1>

END FOR

END FOR

END FOR

// Adds 2 occurrences of an attribute value

6. function funcSumm (Long val1, Long val2)

return val1 + val2

end

//Calculates joint probability based on M-estimation

7. function funcJointProbability (ai, c)

jointFrqMap (a_i, c) + (1.0 / (V_i * V_j * K)) / (N + 1);

Return P

// Calculates joint probability of 3 attributes based on M-estimation

8. function funcTripleProbability (ai, c)

tripleFreqMap (a_i, a_j, a_z, c) + (1.0 / V_z) / (jointFrqMap (a_i, a_j, c) + 1);

```

    Return P
end

//Calculates joint probability based on M-estimation
9. function funcJointProbablityNew (ai, c)
    jointFrqMap (ai, c) + (1.0 / (Vi *Vj)) / (N +1);
    Return P
end

```

Predict Method

The predict method of the object accepts an input vector (which is the list of attribute values without a class label). Estimates the class for the given predicate value based on the results from the Training phase.

Pseudo Code for Prediction phase of A2DE Algorithm

1. **function CalculateArgMax (att List, class)**
2. **Input:** list of attribute tokens and class label
3. **Output:** Probability Estimation for the given class label
4. $Est \leftarrow 0$
5. $P_{mul} \leftarrow 1$
6. FOR I \leftarrow 1 TO m
7. FOR J \leftarrow I+1 TO m
8. key \leftarrow "ai, aj, c"
9. $P(a_i, a_j, c) \leftarrow$ jointProbMap. Get(key)
10. For z \leftarrow 1 To m
11. If $z \neq I$ or $z \neq j$ then
12. key \leftarrow "ai, aj, az, c"
13. $P(a_i, a_j, c) \leftarrow$ tripleProbMap. Get(key);
14. $P_{mul} \leftarrow P_{mul} \times P(a_i, a_j, a_z, c)$
15. End FOR
16. END FOR
17. END FOR
18. $Est \leftarrow Est + (P(a_i, a_j, c) \times P_{mul})$
19. $Est \leftarrow Est / N$
20. Return Est

Description of AnDE Argmax Method

Line 1-3: Argmax method accepts attribute vector and class label as an input

Line 4-5: Initializes Est and P_{mul} variable

Line 6: Loops through all attribute values in the input vector

Line 7: For each parent attribute it loops through all other attribute values.

Line 9: It retrieves the joint probability of a_i , a_j and class from HashMap.

Line 11: Loops through all attributes for each combination of the parent attributes I and j.

Line 13: It calculates the triple probability for a_i , a_j and a_z with class label termed as

Line 14: aggregates the triple probabilities calculated in the previous steps.

Line 18: multiplies the aggregate of triple probabilities with the Joint probability of the parent attributes and. This Estimate value will be summed up for each parent attribute.

Line 19: The Total estimate value will be averaged with N value to return the estimation for the given class label.

Line 20: This procedure will be repeated for each class label. The class label with the maximum estimation will be predicted for the given input vector according to AnDE algorithm.

Line 21: End of procedure

5.5. WAnDE Implementation Process

Generally, WAnDE is a modification of AnDE by addition of Weighting function. The AnDE algorithm like WOADE is essentially a parallel algorithm. The AnDE algorithm can be divided into two stages. In the first stage, which is the Training stage it calculates the N, V and other constant values. In addition, it calculates joint frequency, triple frequency, joint probability, triple Probability and other constants. Finally, it saves the results to use it on the 2nd stage of the algorithm.

The implementation of AnDE based on Spark map reduce platform is as follows.

Constructor

The constructor of the WAnDE library accepts the spark context as a parameter. It initializes the sparkContext, Hash Tables and Lists which are used for storing data. which are namely.

Initialization of list clsList
Initialization of HashMap WMap
Initialization of HashMap distItemsMap
Initialization of HashMap itemFrqMap
Initialization of jointProbMap
Initialization of tripleProbMap

finally, it calls the **CreateModel** Method which uses to train the dataset accordingly

CreateModel

The CreateModel method of **WAnDE** algorithm creates the classification model by accepting **InputRDD** as an argument.

Pseudo Code for Training Phase of WAnDE Algorithm

1. $N = \text{Training_RDD.Count}$
2. $\text{DistinctItems_RDD} (I, \text{distValsList}) = \text{Training_RDD. FlatMapToPair} (\text{funcPairDist}). \text{distinct} (). \text{cache} ()$
3. $V_Map ("i", v) = \text{DistinctItems_RDD. countByKey} ()$
4. $K = VMap.get("iclass")$
5. $vMapBroadcast = sc.Broadcast(V_Map)$
6. $kMapBroadcast = sc.Broadcast(K)$
7. $\text{DistinctItems_Map} = \text{DistinctItems_RDD.} (). \text{reduceByKey} (\text{funcCollect}).$
8. $\text{cls-List} = \text{DistinctItems_Map. get}(i\text{-class}).$
9. $\text{DistinctItems_RDD.unpersist}()$

```

// Calculate Item Frequency and Probability
13. itemFrequencyRDD = Training_RDD. FlatMapToPair (funcPairSingle).
reduceByKey(funcSumm)

14. jointFrequencyRDD = Training_RDD.flatMapToPair(funcPairJoint).
reduceByKey(funcSumm)

15. jointFrequencyRDDNew =Training_RDD.flatMapToPair(funcPairJointNew).
reduceByKey(funcSumm)

16. tripleFrequencyRDD =Training_RDD.flatMapToPair(funcPairTriple).
reduceByKey(funcSumm)

17. jointProbRDD = jointFrequencyRDD. MapToPair(funcJointProbablity)

18. tripleProbRDD = jointFrequencyRDD. MapToPair(funcTripleProbablity)

19. jointProbRDDNew = jointFrequencyRDD. MapToPair(funcJointProbablityNew)
// Collect Results into Map
20. itemFrqMap = itemFrquencyRDD.collectAsMap()
22. jointFrqMap = jointFrquencyRDD.collectAsMap()
23. jointFrqMapBroadcast = sc. Broadcast(jointFrqMap)
24. jointProbMap = jointProbRDD.collectAsMap()
25. tripleProbMap = tripleProbRDD.collectAsMap()
26. jointProbMapNew = jointProbRDDNew.collectAsMap()
// Training Completed

```

Methods (closures) used in the WAnDE algorithm

// Maps each attribute value with the index

1. function funcPairDist (String x)

```
FOR I ← 1 TO m
  New Tuple2 <" i", val>
END FOR
END
```

// Maps each attribute value and Index with One

2. function funcPairSingle (String x)

```
FOR I ← 1 TO m
  New Tuple2 <" i, val", 1>
END FOR
END
```

// Maps each attribute value, index and class with one

3. function funcPairJoint (String x)

```
FOR I ← 1 TO m-1
  FOR J ← 1 TO m
    New Tuple2 <" i, val, j, valJ, class", 1>
  END FOR
END FOR
END
```

// Maps the combination of attribute values with one

4. function funcPairJointNew (String x)

```
FOR I ← 1 TO m-1
  FOR J ← 1 TO m
    New Tuple2 <" i, val, j, valJ", 1>
  END FOR
```

END FOR

END

// Maps combination of attribute values and class with 1

5. function funcPairTriple (String x)

FOR I ← 1 TO m-2

FOR J ← 1 TO m-1

FOR J ← 1 TO m

New Tuple2 <" i, val1, j, val2, z, val3, class", 1>

END FOR

END FOR

END FOR

END

// Adds 2 occurrences of an attribute value

6. function funcSumm (Long val1, Long val2)

return val1 + val2

END

// Calculates joint probability based on M-estimation

7. function funcJointProbablity (a_i, c)

jointFrqMap (a_i, c) + (1.0 / (V_i * V_j * K)) / (N + 1);

Return P

END

// Calculates joint probability of 3 attributes based on M-estimation

8. funcTripleProbablity (a_i, c)

tripleFreqMap (a_i, a_j, a_z, c) + (1.0 / V_z) / (jointFrqMap (a_i, a_j, c) + 1);

Return P

END

// Calculates joint probability based on M-estimation

9. funcJointProbabilityNew (a_i, c)

*jointFrqMap (a_i, c) + (1.0 / (V_i * V_j)) / (N + 1);*

Return P

END

Algorithm for Three types of Weighting Function Used in WAnDE

1. **function FuncWeight1(i, j)**
2. **Input:** index of two attributes
3. **Output:** The amount of information that is passing from I to j (weight of Att)
4. FOR EACH attribValue a_i, attribValue a_j, classValue c ∈ AI*, C*
5. *F_{ai} ← itemFrequency_Map (“a_i”) // frequency of attribute i*
6. *F_{aj} ← itemFrequency_Map (“a_j”) // frequency of attribute j*
7. *f_c ← itemFrequency_Map (“c”) // frequency of class*
8. *pa_i ← f_{ai} / N // probability of attribute*
9. *pa_j ← f_{aj} / N // probability of attribute*
10. *pc ← f_c / N // probability of class*
11. *Nom ← jointProbMap (“a_i, a_j, c”) // joint prob attributes and class*
12. *Den ← pa₁ x pa₂ x Pc // multiple of attribute probability and class*
13. *W ← W + jointProbMap (“a_i, a_j, c”) x (Log Nom - Log Den)*
14. *Weight_Map (i, W_i)*
15. *End FOR*
16. *End*

1. **function FuncWeight2 (I, j)**
2. **Input: index of two attributes**
3. **Output: The amount of information that is passing from I to j (weight of Att)**
4. FOR EACH attribValue ai, attribValue aj, classValue c $\in AI^*, C^*$
5. $F_{ai} \leftarrow \text{itemFrequency_Map}("a_i")$ // frequency of attribute
6. $F_{aj} \leftarrow \text{itemFrequency_Map}("a_j")$ // frequency of attribute
7. $f_c \leftarrow \text{itemFrequency_Map}("c");$ // frequency of class
8. $p_{ai} \leftarrow f_{ai} / N$ // probability of attribute
9. $p_{aj} \leftarrow f_{aj} / N$ // probability of attribute
10. $p_c \leftarrow f_c / N$ // probability of class
11. $Nom \leftarrow \text{jointProbMap}("a_i, a_j, c")$ // joint prob attributes and class
12. $Den \leftarrow \text{jointProbMap}("a_i, a_j") \times p_c$ // mul joint probability and class
13. $W \leftarrow W + \text{jointProbMap}("a_i, a_j, c") \times (\text{Log } Nom - \text{Log } Den)$
14. $\text{Weight_Map}(i, W_i)$
15. End FOR
16. End

28. **function FuncWeight3 (I, j)**
29. **Input: index of 2 Attributes**
30. **Output: The amount of information that is passing from I to j (weight of Att)**
31. FOR EACH attribValue ai, attribValue aj, classValue c $\in AI^*, C^*$
32. $F_{ai} \leftarrow \text{itemFrequency_Map}("a_i")$ // frequency of attribute
33. $F_{aj} \leftarrow \text{itemFrequency_Map}("a_j")$ // frequency of attribute
34. $f_c \leftarrow \text{itemFrequency_Map}("c");$ // frequency of class
35. $p_{ai} \leftarrow f_{ai} / N$ // probability of attribute
36. $p_{aj} \leftarrow f_{aj} / N$ // probability of attribute
37. $p_c \leftarrow f_c / N$ // probability of class
38. $Nom \leftarrow \text{jointProbMap}("a_i, a_j, c")$ // joint prob attributes and class
39. $Den \leftarrow \text{jointProbMap}("a_i, c") \times \text{jointProbMap}("a_j, c")$
40. $W \leftarrow W + \text{jointProbMap}("a1, a2, c") \times (\text{Log } Nom - \text{Log } Den)$
41. $\text{Weight_Map}(i, W_i)$
42. End FOR
43. end

Description of the WAnDE CreateModel Procedure

The Training phase of WAnDE resembles AnDE except that it calculates the Weight function for each attribute. The implementation of the WAnDE consists of three types of weight functions.

Predict Method

The predict method of the object accepts an input vector (which is the list of attribute values without a class label). Estimates the class for the given predicate value based on the results from the Training phase.

Pseudo Code for Prediction Phase of WAnDE Algorithm

1. **function CalculateArgMax** (*att List, class*)
2. **Input:** *attribute values and class label*
3. **Output:** *Probability Estimate for the class label*
4. $Est \leftarrow 0$
5. $P_{mul} \leftarrow 1$
6. **FOR** $I \leftarrow 1$ **TO** m
7. **FOR** $J \leftarrow I+1$ **TO** m
8. $key \leftarrow "a_i, a_j, c"$
9. $P(a_i, a_j, c) \leftarrow jointProbMap. Get(key)$
10. **For** $z \leftarrow 1$ **To** m
11. **If** $z! = I$ **or** $z! = j$ **then**
12. $key \leftarrow "a_i, a_j, a_z, c"$
13. $P(a_i, a_j, c) \leftarrow tripleProbMap. Get(key);$
14. $P_{mul} \leftarrow P_{mul} \times P(a_i, a_j, a_z, c)$
15. **End FOR**
16. **END FOR**
17. **END FOR**
18. $Est \leftarrow Est + (W_{ij} * P(a_i, a_j, a_z, c) \times P_{mul})$
19. $Est \leftarrow Est / W_{sum}$
20. **End**

Description of WAnDE Argmax Procedure

The implementation of WAnDE prediction phase resembles AnDE except that it multiplies the aggregate of the probability estimate with Weight value of Parent Attributes. Furthermore, it averages the total probability estimate with Total weight value to give final estimate.

Chapter 6

6. EVALUATION AND RESULTS

This chapter applies the approach introduced in chapter 6 to the realistic datasets found from UCI website. It begins by describing the experiment dataset followed by setup environment. Then presents evaluation of the accuracy by which the approach can estimate the classes of the input data attributes.

A. Experiment Dataset

25 UCI datasets were chosen to support the claim that the proposed approach performs as expected. UCI dataset is chosen as reference because the data were recorded for several real-life appliances and of its wide application as a standard test set by machine learning community for the empirical analysis of machine learning algorithms.

Dataset	N	A	K
Austral	690	14	2
Breast	699	10	2
Cleved	303	13	2
Crex	690	15	2
Diabetes	768	8	2
Glass	214	9	4
heart	270	13	2
hepati	155	19	2
horse	368	22	2
labor	57	16	2
mushroom	8124	22	2
nursary	12960	8	5
pima	768	8	2
segmentation	2310	19	7
sick	2801	29	3
sonar	209	60	3
soyabin-s.csv	47	35	4
hypo	3163	25	2
tic-tac	958	9	2
voting	435	16	2
wine	178	13	3
yeast	1484	8	10
zood	101	16	7
german	1001	20	3
ionod	352	34	3

Table 1 UCI Datasets used for the testing the accuracy of the algorithms

N: Number of records in the dataset, A: Number of attributes in the dataset

K: Number of distinct class values in the dataset

B. Test Environment

The presented accuracy tests and evaluations were done on Polito Datamining Cluster(dbdmgmtr.polito.it) which is a small cluster usually used for testing purpose, it contains 2 worker nodes and 1 master node. The server is running java version 1.6.0.3 and cloudera CDH 5.4.7 with spark version 1.3.0

The scalability testing is performed on the cluster of Big Data laboratory ([Big Data.polito.it](http://BigData.polito.it)). It consists of a group of worker nodes and a Master node used for running Big Data jobs. below is the specification of the Big Data cluster.

Hardware Architecture

30 worker nodes – With storage capacity 768 TB and 2TB Memory (8GB per computing core)

- 18 nodes Dell – With each node having maximum storage capacity 36 TB
- 12 nodes - With each node having maximum storage capacity 10 TB

3 Master nodes with the following characteristics

- 1 Master node DELL PowerEdge R620

Processor type	Intel processor E5-2630v2 6 cores, 2.6GHz
RAM	128 GB DDR3 with processor speed of 1600Mhz
2 Worker nodes DELL PowerEdge R720XD	
Processor type	Intel processor E5-2620v2 6 cores, 2.6GHz
RAM	96 GB DDR3 with processor speed of 1600Mhz

Table 2 Polito Data Mining Cluster Environment

Software architecture

Each node of the `dbdmgmtr@polito.it` and `Big Data@Polito.it` cluster runs a cloudera distribution on Linux Ubuntu (14.04.02 LTS). The cloudera distribution is based on the open source Apache Hadoop framework for Big Data distributed applications. The Apache Hadoop ecosystem data management like YARN, SPARK and HDFS (Hadoop Distributed File System).

C. Evaluation methodology

10-Fold cross validation is used for evaluating the accuracy of the algorithms. The `CrossValidator` module which is implemented in the project as described in chapter 6 validates a given algorithm and gives the average accuracy on a specified dataset.

```
CrossValidator cv = new CrossValidator (sc, inputfileRDD);  
Double avg = cv.kFold (10);
```

K -fold Cross Validation Implementation

The K-Fold cross validation methodology is the evaluation method we used to test our algorithms accuracy in the project. The method divides the data in to 10 partitions and uses each partition as a test set and the rest of the dataset is used as a training set. This process continues 10 times for each partition. Finally, the average of the 10 results will be taken as an accuracy for the dataset.

The constructor of `Cross Validator` class accepts 2 arguments. namely the spark context and an `Input-RDD`.

Interface

1. `Cross Validator Constructor`: accepts 2 arguments as an input the spark context an input `RDD`. it sets the input arguments to the `cross-Validator` object so that it will be used in the future.
2. `K fold method`: accepts 2 arguments the `K` value which in our case is 10 and the output `-Path` where it will store the results of the test. It returns the result

```
CrossValidator cv = new CrossValidator (sc, inputfileRDD);  
Double avg = cv.kFold (10);
```

Implementation

Pseudo Code for 10-fold Cross Validation Implementation

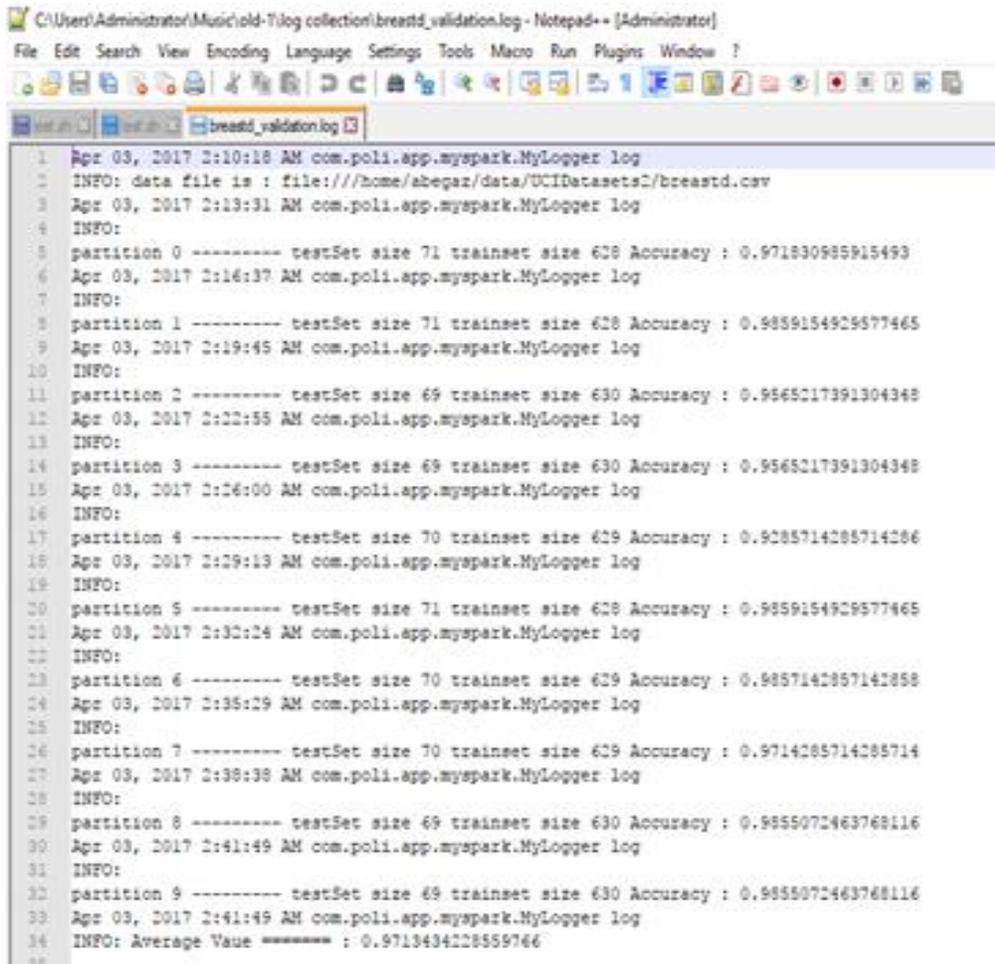
1. $RDD\ trainset \leftarrow null$
2. $List<String> []\ partition \leftarrow input_RDD.collectPartitions(10);$
3. *For* $I \leftarrow 0$ *To* 10
4. *If* I not equal testSet
5. $trainset \leftarrow trainset.union(sc.parallelizes(partition[i]));$
// collects all data except the test partition
6. *End If*
7. *End For*
8. *Foreach* partition P
9. $WAODE_Estimator\ myestimator \leftarrow new\ WAODE_Estimator(sc);$
10. $Myestimator.train();$
11. *For each* $I \leftarrow 1$ *to* N
12. $class \leftarrow myestimator.Predict(Vector(i))$
13. *IF* class equals old class
14. $count \leftarrow count + 1$
15. *End For each*
16. $accuracy \leftarrow count / testset\ size$
17. $sum \leftarrow sum + accuracy$
18. *End for each partition*
19. $average \leftarrow sum / number\ of\ partitions$
20. *Return average*

Logging

It is a utility class made in the application that uses the java logging service? The logging module helps to write the intermediate results and finale outcome of the testing process which is the 10-fold cross validation. Since the processing time for the algorithm takes a longer time. T

The system uses this logger module to save the results on file.

```
java.util.logging.Logger logger = java.util.logging.Logger.getLogger("MyLog");  
logger.info(text);  
MyLogger.log ("Average Value =====: " + result);
```



```
C:\Users\Administrator\Music\old-T\log collection\breastd_validation.log - Notepad++ [Administrator]  
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?  
breastd_validation.log  
1 Apr 03, 2017 2:10:18 AM com.poli.app.myspark.MyLogger log  
2 INFO: data file is : file:///home/abegaz/data/UCIDatasets2/breastd.csv  
3 Apr 03, 2017 2:13:31 AM com.poli.app.myspark.MyLogger log  
4 INFO:  
5 partition 0 ----- testSet size 71 trainset size 628 Accuracy : 0.971830985915493  
6 Apr 03, 2017 2:14:37 AM com.poli.app.myspark.MyLogger log  
7 INFO:  
8 partition 1 ----- testSet size 71 trainset size 628 Accuracy : 0.9859154929577465  
9 Apr 03, 2017 2:19:45 AM com.poli.app.myspark.MyLogger log  
10 INFO:  
11 partition 2 ----- testSet size 69 trainset size 630 Accuracy : 0.9565217391304348  
12 Apr 03, 2017 2:22:55 AM com.poli.app.myspark.MyLogger log  
13 INFO:  
14 partition 3 ----- testSet size 69 trainset size 630 Accuracy : 0.9565217391304348  
15 Apr 03, 2017 2:26:00 AM com.poli.app.myspark.MyLogger log  
16 INFO:  
17 partition 4 ----- testSet size 70 trainset size 629 Accuracy : 0.9285714285714286  
18 Apr 03, 2017 2:29:13 AM com.poli.app.myspark.MyLogger log  
19 INFO:  
20 partition 5 ----- testSet size 71 trainset size 628 Accuracy : 0.9859154929577465  
21 Apr 03, 2017 2:32:24 AM com.poli.app.myspark.MyLogger log  
22 INFO:  
23 partition 6 ----- testSet size 70 trainset size 629 Accuracy : 0.9857142857142858  
24 Apr 03, 2017 2:35:29 AM com.poli.app.myspark.MyLogger log  
25 INFO:  
26 partition 7 ----- testSet size 70 trainset size 629 Accuracy : 0.9714285714285714  
27 Apr 03, 2017 2:38:38 AM com.poli.app.myspark.MyLogger log  
28 INFO:  
29 partition 8 ----- testSet size 69 trainset size 630 Accuracy : 0.9855072463768116  
30 Apr 03, 2017 2:41:49 AM com.poli.app.myspark.MyLogger log  
31 INFO:  
32 partition 9 ----- testSet size 69 trainset size 630 Accuracy : 0.9855072463768116  
33 Apr 03, 2017 2:41:49 AM com.poli.app.myspark.MyLogger log  
34 INFO: Average Vaue ===== : 0.9713434228559746  
35
```

Figure 8 7 Sample Log file showing 10-fold cross validation accuracy result for a dataset

D Experimental Results

i. Average Accuracy and Execution Time of the implemented prediction algorithms (WAOE, AnDE, WAnDE) on selected UCIDatasets.

Dataset	WAOE	ET	WAOE	ET	WAnDe	ET	waNde	ET	aNde		aNde	ET
	M-E		Laplace		M-E		Laplace		M-E		Laplace	
Austral	0.8494818	98s	0.85152431	97s	0.85147591	164s	0.85731445	169s	0.853837787	150s	0.8603421	141s
Breast	0.9683942	87s	0.96685577	86s	0.96839424	135s	0.96536324	139s	0.969886773	123s	0.9668558	118s
Cleved	0.8295854	82s	0.82958539	81s	0.81939491	135s	0.82045552	138s	0.822728243	119s	0.8227282	117s
crex	0.8580923	103s	0.86263112	104s	0.86294112	177s	0.87131172	174s	0.860727756	154s	0.8681243	147s
diabets	0.7804979	80s	0.77770662	80s	0.78110266	122s	0.77836294	126s	0.780416748	110s	0.7793298	107s
glass	0.7793352	66s	0.75377605	69s	0.76090382	105s	0.75423715	106s	0.77678617	92s	0.7601195	92s
heart	0.8287757	82s	0.82877567	81s	0.82813155	126s	0.82813155	130s	0.828131549	114s	0.8281315	115s
hepati	0.8790759	77s	0.85870551	79s	0.87282588	131s	0.87907588	131s	0.879075878	119s	0.8790759	116s
horse	0.8162924	110s	0.81198836	113s	0.81684558	470s	0.83092075	200s	0.821157241	187s	0.8255095	184s
labor	0.9542424	49s	0.95424242	51s	0.97424242	78s	0.97424242	84s	0.974242424	71s	0.9742424	72s
mushroom	1	143s	0.99987179	148s	1	424s	1	457s	1	417s	1	400s
nursary	0.9271733	109s	0.92693599	119s	0.94921758	181s	0.94859827	189s	0.946484002	166s	0.9454229	168s
pima	0.7838758	82s	0.7838758	87s	0.79285323	123s	0.79411906	125s	0.795798705	111s	0.7957987	110s
segmentation	0.9662024	159s	0.95760634	166s	0.96798749	428s	0.95794958	445s	0.968332049	441s	0.958766	433s
sick	0.9757968	133s	0.97509552	136s	0.9746876	453s	0.97403006	466s	0.974691351	449s	0.9736928	445s
sonar	0.8456517	120s	0.83660146	122s	0.85190166	603s	0.84094928	624s	0.845651664	909s	0.8366015	896s
soyabin-s.csv	1	58s	1	59s	1	111s	1	108s	1	103s	1	101s
hypo	0.9787536	180s	0.96631571	179s	0.98066328	869s	0.96787898	727s	0.981947428	652s	0.9757705	659s
tic-tac	0.7269482	97s	0.7266693	98s	0.8859782	410s	0.88574327	145s	0.905983306	135s	0.9071881	131s
voting	0.9512139	99s	0.94410926	103s	0.95127516	156s	0.95035529	158s	0.954737574	147s	0.947482	145s
wine	0.9891813	74s	0.98362573	75s	0.98362573	119s	0.98362573	117s	0.983625731	109s	0.9836257	107s
yeast	0.5973158	96s	0.59475364	99s	0.60529614	142s	0.60006873	141s	0.599729112	128s	0.5957406	123s
zood	0.9777778	66s	0.92209596	67s	0.96868687	102s	0.93118687	102s	0.968686869	94s	0.922096	94s
german	0.7579323	118s	0.75928417	130s	0.75351115	227s	0.75551294	236s	0.749705136	210s	0.7606239	204s
ionod	0.9394478	122s	0.93087637	128s	0.93613898	377s	0.93206274	395s	0.936138977	429s	0.9320627	417s

Table 3 . Measuring the accuracy of WAOE, AnDE, WAnDE estimators by using 10-fold cross validation methodology on selected UCIDatasets

ii. Average Accuracy results of WAODE, AnDE, WAnDE distributed versions with respect to WAODE centralized version (Rapid Miner version).

Dataset	WOADE	WOADE	waNde	waNde	aNde	aNde	WOADE Average accuracy RapidMiner version (%)
	M-Estimator	Laplace	M-Estimator	Laplace	M-Estimator	Laplace	
austra	0.849481805	0.851524314	0.851475912	0.857314453	0.853837787	0.860342085	84.64% +/- 4.21% (mikro: 84.64%)
breast	0.968394235	0.966855774	0.968394235	0.965363236	0.969886773	0.966855774	97.13% +/- 1.82% (mikro: 97.14%)
cleved	0.829585385	0.829585385	0.819394909	0.820455515	0.822728243	0.822728243	82.17% +/- 3.07% (mikro: 82.18%)
Crex	0.858092321	0.862631125	0.862941116	0.871311719	0.860727756	0.868124266	86.67% +/- 2.73% (mikro: 86.67%)
diabets	0.780497855	0.777706617	0.781102663	0.778362937	0.780416748	0.779329791	77.85% +/- 4.23% (mikro: 77.86%)
Glass	0.77933519	0.75377605	0.760903817	0.754237151	0.77678617	0.760119504	76.67% +/- 7.99% (mikro: 76.64%)
Heart	0.828775672	0.828775672	0.828131549	0.828131549	0.828131549	0.828131549	82.59% +/- 5.25% (mikro: 82.59%)
hepati	0.879075878	0.858705507	0.872825878	0.879075878	0.879075878	0.879075878	83.88% +/- 5.17% (mikro: 83.87%)
horse	0.816292393	0.811988364	0.816845576	0.830920751	0.821157241	0.825509496	82.03% +/- 5.36% (mikro: 82.07%)
Labor	0.954242424	0.954242424	0.974242424	0.974242424	0.974242424	0.974242424	95.00% +/- 7.64% (mikro: 94.74%)
mushroom	1	0.999871795	1	1	1	1	100.00% +/- 0.00% (mikro: 100.00%)
nursary	0.92717333	0.926935993	0.949217584	0.948598268	0.946484002	0.945422861	92.91% +/- 0.59% (mikro: 92.91%)
Pima	0.783875804	0.783875804	0.792853234	0.794119057	0.795798705	0.795798705	78.64% +/- 4.89% (mikro: 78.65%)
segmentation	0.96620244	0.957606344	0.967987489	0.957949581	0.968332049	0.958766603	96.93% +/- 1.33% (mikro: 96.93%)
Sick	0.975796845	0.975095516	0.974687603	0.974030062	0.974691351	0.97369276	97.61% +/- 0.72% (mikro: 97.61%)
sonar	0.845651664	0.836601457	0.851901664	0.840949283	0.845651664	0.836601457	86.48% +/- 8.36% (mikro: 86.54%)
soyabin-s.csv	1	1	1	1	1	1	100.00% +/- 0.00% (mikro: 100.00%)
Hypo	0.978753553	0.966315715	0.980663279	0.967878982	0.981947428	0.975770468	98.13% +/- 0.66% (mikro: 98.13%)
tic-tac	0.72694819	0.726669296	0.8859782	0.885743268	0.905983306	0.907188125	71.72% +/- 4.17% (mikro: 71.71%)
voting	0.951213875	0.944109263	0.951275158	0.950355289	0.954737574	0.947482016	94.27% +/- 2.92% (mikro: 94.25%)
Wine	0.989181287	0.983625731	0.983625731	0.983625731	0.983625731	0.983625731	98.89% +/- 2.22% (mikro: 98.88%)
yeast	0.597315818	0.594753641	0.605296141	0.600068725	0.599729112	0.595740647	59.23% +/- 2.94% (mikro: 59.23%)
Zood	0.977777778	0.92209596	0.968686869	0.931186869	0.968686869	0.92209596	97.09% +/- 4.45% (mikro: 97.03%)
german	0.757932333	0.75928417	0.753511147	0.755512936	0.749705136	0.76062388	75.40% +/- 4.32% (mikro: 75.40%)
lonod	0.939447801	0.930876372	0.936138977	0.932062743	0.936138977	0.932062743	93.16% +/- 3.66% (mikro: 93.16%)

Table 4 Average accuracy results of the (WAODE, A2de, WAnDE) versus WAODE centralized version (Rapid Miner) on selected UCIDatasets

iii. Accuracy results of WOADE distributed versions with respect to WOADE centralized version (Rapid Miner version).

Dataset	WOADE M-Estimator Distributed version	ET	WOADE Average accuracy RapidMiner version (%)
austra	0.849482	98s	84.64% +/- 4.21% (mikro: 84.64%)
breast	0.968394	87s	97.13% +/- 1.82% (mikro: 97.14%)
cleved	0.829585	82s	82.17% +/- 3.07% (mikro: 82.18%)
crex	0.858092	103s	86.67% +/- 2.73% (mikro: 86.67%)
diabets	0.780498	80s	77.85% +/- 4.23% (mikro: 77.86%)
glass	0.779335	66s	76.67% +/- 7.99% (mikro: 76.64%)
heart	0.828776	82s	82.59% +/- 5.25% (mikro: 82.59%)
hepati	0.879076	77s	83.88% +/- 5.17% (mikro: 83.87%)
horse	0.816292	110s	82.03% +/- 5.36% (mikro: 82.07%)
labor	0.954242	49s	95.00% +/- 7.64% (mikro: 94.74%)
mushroom	1	143s	100.00% +/- 0.00% (mikro: 100.00%)
nursary	0.927173	109s	92.91% +/- 0.59% (mikro: 92.91%)
pima	0.783876	82s	78.64% +/- 4.89% (mikro: 78.65%)
segmentation	0.966202	159s	96.93% +/- 1.33% (mikro: 96.93%)
sick	0.975797	133s	97.61% +/- 0.72% (mikro: 97.61%)
sonar	0.845652	120s	86.48% +/- 8.36% (mikro: 86.54%)
soyabin-s.csv	1	58s	100.00% +/- 0.00% (mikro: 100.00%)
hypo	0.978754	180s	98.13% +/- 0.66% (mikro: 98.13%)
tic-tac	0.726948	97s	71.72% +/- 4.17% (mikro: 71.71%)
voting	0.951214	99s	94.27% +/- 2.92% (mikro: 94.25%)
wine	0.989181	74s	98.89% +/- 2.22% (mikro: 98.88%)
yeast	0.597316	96s	59.23% +/- 2.94% (mikro: 59.23%)
zood	0.977778	66s	97.09% +/- 4.45% (mikro: 97.03%)
german	0.757932	118s	75.40% +/- 4.32% (mikro: 75.40%)
ionod	0.939448	122s	93.16% +/- 3.66% (mikro: 93.16%)

Table 5 Average accuracy comparison of WOADE Distributed implementation vs WOADE centralized (Rapid Miner version).

iv. Accuracy results of WAnDE distributed version with respect different M-Estimator functions

Dataset	WaNde ME	WAnDE ME2	WAnDE ME3
austra	0.851023545	0.853115649	0.856080855
breast	0.970377565	0.96868265	0.970336741
cleved	0.815285455	0.818142598	0.82785868
crex	0.859936733	0.857188756	0.853806815
diabets	0.787336927	0.789815507	0.771418053
glass	0.769705969	0.774969127	0.789539645
heart	0.830455867	0.830455867	0.831271718
hepati	0.843366007	0.843366007	0.83669934
horse	0.816764192	0.829867551	0.81494931
labor	0.943333333	0.943333333	0.963333333
mushroom	1	1	1
nursary	0.948315484	0.948315484	0.940300611
pima	0.788263122	0.78704361	0.768990403
segmentation	0.966159794	0.966159794	0.967033573
sick	0.974435475	0.975180777	0.974043318
sonar	0.854598314	0.863487203	0.857931647
soyabin-s.csv	1	1	1
hypo	0.980824023	0.988112241	0.980202306
tic-tac	0.891363084	0.898863615	0.836160513
voting	0.956221229	0.958923931	0.958493956
wine	0.991608392	0.991608392	0.996153846
yeast	0.60013184	0.60913184	0.596632452
zood	0.953012266	0.959012266	0.978409091
german	0.758002363	0.755806959	0.759540428
ionod	0.936764671	0.936764671	0.939264671

Table 6 Average accuracy comparison of WAnDE with respect to 3 different M-estimator Implementations.

v. Accuracy results of WAnDE distributed version with respect to Different Laplace Estimator functions.

Dataset	Wande Laplace	Wande Laplace 2	Wande Laplace3
austra	0.864535483	0.863330664	0.861779102
breast	0.967416827	0.967416827	0.969111743
cleved	0.815285455	0.815285455	0.818142598
crex	0.865565459	0.868198862	0.860516551
diabets	0.789968506	0.791131297	0.771418053
glass	0.761332764	0.761332764	0.769705969
heart	0.830455867	0.830455867	0.831271718
hepati	0.843366007	0.843366007	0.843366007
horse	0.832897634	0.835529213	0.833404734
labor	0.963333333	0.963333333	0.963333333
mushroom	1	1	1
nursary	0.947769336	0.947769336	0.94023249
pima	0.788263122	0.78704361	0.770209915
segmentation	0.954515602	0.954970147	0.955384153
sick	0.973018477	0.973354048	0.973018477
sonar	0.856217362	0.856217362	0.856217362
soyabin-s.csv	1	1	1
hypo	0.968769239	0.964653731	0.96973356
tic-tac	0.890352983	0.896830928	0.83963705
voting	0.949883497	0.949883497	0.94919842
wine	0.986345234	0.986345234	0.986345234
yeast	0.595522933	0.595522933	0.598401874
zood	0.940512266	0.946512266	0.929401154
german	0.757707947	0.760900151	0.759623972
ionod	0.922902237	0.920680014	0.922902237

Table 7 Average accuracy comparison of WAnDE with respect to 3 different Laplace Estimator Implementations

vi. Summary of Experimental results

Algorithm	Average accuracy results for WAODE Centralized version (Rapid Miner) vs WAODE Distributed Implementation for 25 UCI Datasets.	
	Average Accuracy	Time(seconds)
WAODE Centralized	0.8755	77.6
WAODE Distributed	0.874906848	69.6

Table 8 Summary of experimental results of WAODE Implementations

According to the Experimental results the distributed implementation of WAODE have equal accuracy with respect to the centralized version (Rapid Miner). The similarity shows that our implementation of the distributed version is correct. In addition, the distributed versions performs greater efficiency with respect to the centralized one.

Algorithm	Average accuracy results for WAODE vs AnDE Distributed Implementation on 25 UCI Datasets.	
	Average Accuracy	Time(seconds)
WAODE ME	0.874906848	69.6
WAODE LAPLACE	0.87287444	72.6
AnDE ME	0.885575541	149.56
AnDE LAPLACE	0.883096376	165.68

Table 9 Summary of Experimental results for AnDE and WAODE

According to the Experimental results A2DE performs better accuracy with respect to WAODE. while taking double the time needed for same dataset. However, Since the accuracy difference is little WAODE can be more usable than AnDE.

Algorithm	Average accuracy results for different WAnDE implementations tested on 25 UCI Datasets.					
	WAnDE ME1	WAnDE ME2	WAnDE ME3	WAnDE LA 1	WAnDE LA 2	WAnDE LA 3
Average Accuracy	0.883491426	0.885893913	0.882738052	0.882637503	0.883202542	0.878894228
Time(second s)	254.72	242.62	258.44	233.28	230.74	245.55

Table 10 Summary of experimental results for WAnDE implementation

Comparison of 3 different WAnDE implementations shows that M-Estimation together with Weight function 2 perform better accuracy than the rest of implementations.

According to the Experimental results, WAnDE implementation performed similar accuracy with AnDE, this shows that adding Weight function on AnDE doesn't increase the accuracy as expected.

However, the experiment shows that WeightFunction2 performed better than the other weight functions, as a result it can be used in the implementation of WAODE and AnDE algorithms to further improve the prediction performance.

6.2. Conclusions and Future work

This paper presents the implementation scheme of Distributed WAODE algorithm based on Spark. It is experimentally proven that implementing WAODE by using spark improves its efficiency. It is also experimentally tested to prove that adding weight on the AODE algorithm can improve the accuracy without increasing computational complexity. Other parallel implementations like A2DE show that increasing the attribute dependency with respect to the class can also improve the predictive accuracy. According to the results from this thesis paper it can be concluded that not only the WOADE algorithm but also other classification algorithms, such as AWAODE-GW, DTWAODE etc. can also be speed up by using the Spark platform.

Bibliography

(n.d.).

1. apache. (n.d.). Retrieved from <http://hadoop.apache.org/>
2. bigdataanalytic. (n.d.). Retrieved from aleprabatech.com/.../big-data-analytic
3. Hadoop. (n.d.). Retrieved from bigdatahadooppro.com/.../hadoop-tutorial-for-beginners
4. Heidelberg, B. (n.d.). *Computing Infrastructure for Big Data Processing* . Retrieved from <http://www.istc-cc.cmu.edu/publications/papers/2013/LingLiu-FCS.pdf>
5. mdpi. (n.d.). Retrieved from www.mdpi.com/.../407/htm
6. mdppi. (n.d.). Retrieved from <http://www.mdpi.com/.../407/htm>
7. prabatech. (n.d.). Retrieved from <http://prabatech.com/big-data-analytic>
8. researchIjcaOnline. (n.d.). Retrieved from research.ijcaonline.org/.../number2/pxc3902356.pdf
9. SasInsights. (n.d.). Retrieved from https://www.sas.com/en_us/insights/analytics/big-data-analytics.html
10. scholar, S. (n.d.). *Semantic scholar pdf*. Retrieved from pdfs.semanticscholar.org/.../98602c43c3fb6aee8b...05ed9ded0bccbc.pdf
11. SemanticScholar. (n.d.). Retrieved from pdfs.semanticscholar.org/.../5cb86900e16ae7af39...a525569e8d4df2.pdf
12. springer, l. (n.d.). *link springer pdf*. Retrieved from link.springer.com/.../10.1007/978-3-540-36668-3_116.pdf
13. technopedia. (n.d.). Retrieved from <https://www.techopedia.com/2/31773/technology-trends/open-source/why-spark-is-the-future-big-data-platform>
14. webopedia. (n.d.). Retrieved from https://www.webopedia.com/.../B/big_data_analytics.html
15. webopedia. (n.d.). Retrieved from https://www.webopedia.com/TERM/B/big_data_analytics.html
16. Wikipedia. (n.d.). Retrieved from en.wikipedia.org/.../wiki/Distributed_computing