Exploring and Evaluating the Scalability and Efficiency of Apache Spark using Educational Datasets

Jian Zhang

A THESIS SUBMITTED TO

THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF ARTS

Graduate Program in

Information System and Technology

York University

Toronto, Ontario

April 2018

© Jian Zhang, 2018

Abstract

Research into the combination of data mining and machine learning technology with web-based education systems (known as education data mining, or EDM) is becoming imperative in order to enhance the quality of education by moving beyond traditional methods. With the worldwide growth of the Information Communication Technology (ICT), data are becoming available at a significantly large volume, with high velocity and extensive variety. In this thesis, four popular data mining methods are applied to Apache Spark, using large volumes of datasets from Online Cognitive Learning Systems to explore the scalability and efficiency of Spark. Various volumes of datasets are tested on Spark MLlib with different running configurations and parameter tunings. The thesis convincingly presents useful strategies for allocating computing resources and tuning to take full advantage of the in-memory system of Apache Spark to conduct the tasks of data mining and machine learning. Moreover, it offers insights that education experts and data scientists can use to manage and improve the quality of education, as well as to analyze and discover hidden knowledge in the era of big data.

Acknowledgements

I would like to express my gratitude to my supervisor Professor Zijiang Yang for the useful comments, remarks and engagement through the learning process of my master thesis and the research. Her guidance helped me in all the time of my pursuing of the master academic learning. I also own a deep sense of gratitude to Professor Marin Litoiu for his keen informative interest and support with my thesis. His course offered me extensive and clear clue on the research of the thesis.

Furthermore, I would like to thank Professor Huaiping Zhu for his support with my thesis. Also, I want to thank Professor Michael Chen for the support with his laboratory of Apache Spark Cluster in York University.

Finally, I would extend my appreciation to my family and my mother for their supports and encouragement during various phases of my study in York University.

Table of Contents

Abstract ii

Acknowledgements iii

Table of Contents iv

List of Tables vi

List of Figures vii

Chapter 1. Introduction 1

1.1 Motivation 1

1.2 Problem Definition 4

1.3 Significance 7

1.4 List of Contributions 8

1.5 Thesis Outline 8

Chapter 2. Literature Review 10

2.1 Cognitive Learning System Analytics with Data Mining 10

2.1.1 Cognitive Tutor System 11

2.1.2 Educational Data Mining and Big Data 11

2. 2 KDD Competition Paper Review 16

Chapter 3. Data Mining Methodologies 20

3.1 Data Mining Process 20

3.2 Data Preparation 23

3.2.1 Data Transformation 23

3.2.2 Data Discretization 24

3.2.3 Feature Selection 25

3.2.4 Imbalanced and Sparse Data 26

3.2.5 Missing Values 27

3.2.6 Normalization and Standardization 28

3.3 Data Mining Algorithms 30

3.3.1 Logistic Regression 30

3.3.2 Decision Tree and Random Forest 31

3.3.3 Support Vector Machine 34

3.3.4 Clustering 35

Chapter 4. Big Data with Distributed Systems 37

4.1 Apache Hadoop 37

4.2 Apache Spark 40

Chapter 5. Methodology 47
5.1 Data Pre-processing 48
5.1.1 Feature Generation 49
5.1.2 Feature Manipulation 50
5.1.3 Standardization 52
5.2 Learning Algorithms 52
5.3 Implementation with Python on Apache Spark 53
5.4 Scalability and Performance Evaluation Design 55
5.4.1 The Infrastructure of York University Spark Cluster 55
5.4.2 The Infrastructure of Google Cloud Dataproc Spark Cluster 57
Chapter 6. Results and Discussion 59
6.1 Datasets Overview 60
6.2 Data Pre-processing 62
6.2.1 Feature Generation 62
6.2.2 Feature Manipulation and Standardization 65
6.3 Learning Algorithms Experiments on Spark Clusters 66
6.3.1 Classification Results with York University Lab Spark Cluster 66
6.3.2 Classification Models for Google Cloud Dataproc Spark Cluster 72
6.3.3 Experiments on Datasets with Varied Volumes 79
6.3.4 Experiment with Re-partition of One million Cases Dataset 86
Chapter 7. Conclusion and Future Works 89
7.1 Conclusion 89
7.2 Future Work 91
Reference 93
Appendix 102
A. Coding Reference 102
1. Logistic Regression on Spark version 2.2.0/2.0.0 103
2. Support Vector Machine on Spark version 2.2.0 104
3. Decision Tree on Spark version 2.2.0 105
4. Random Forest on Spark version 2.2.0 107

List of Tables

Table 1 KDD Competition Datasets 47
Table 2 Results of Logistic Regression experiment I on York U Spark Cluster 67
Table 3 Results of Logistic Regression experiment II on York University Spark Cluster 69
Table 4 Comparison of memory strategies on York U Spark Cluster 71
Table 5 Results of Logistic Regression(10-folded) on Cloud Spark Cluster 73
Table 6 Results of SVM (10-folded) on Cloud Spark Cluster 74
Table 7 Results of Random Forest (non-folded) on Cloud Spark Cluster 75
Table 8 Results of Decision Tree (non-folded) on Cloud Spark Cluster 76
Table 9 Results of SVM (non-folded) on Cloud Spark Cluster 77
Table 10 Results of LR (non-folded) on Cloud Spark Cluster 78
Table 11 Results of LR (non-folded) with 2 Million cases dataset 80
Table 12 Results of SVM (non-folded) with 2 Million cases dataset 81
Table 13 Results of LR (non-folded) with One Million cases dataset 81
Table 14 Results of SVM (non-folded) with One Million cases dataset 82
Table 15 Results of LR (non-folded) with Half Million cases dataset 83
Table 16 Results of SVM (non-folded) with Half Million cases dataset 83
Table 17 Results of LR re-partition with One Million cases dataset 87
Table 18 Results of LR re-partition and the number of nodes with One Million cases dataset ... 88

List of Figures

Figure 1 Data mining working process 21
Figure 2 Apache Hadoop Ecosystem 38
Figure 3 Apache Spark Stack 42
Figure 4 Design of the experiments 48
Figure 5 The York University Spark Cluster Architecture 56
Figure 6 The Google Cloud Spark Cluster Architecture 57
Figure 7 Feature break up 63
Figure 8 Results of Logistic Regression experiment I on York U Spark Cluster 68
Figure 9 Results of Logistic Regression experiment II on York University Spark Cluster 69
Figure 10 Comparison of memory strategies on York U Spark Cluster 71
Figure 11 Results of Logistic Regression(10-folded) on Cloud Spark Cluster 73
Figure 12 Results of SVM (10-folded) on Cloud Spark Cluster 74
Figure 13 Results of Random Forest (non-folded) on Cloud Spark Cluster 75
Figure 14 Results of Decision Tree (non-folded) on Cloud Spark Cluster 76
Figure 15 Results of SVM (non-folded) on Cloud Spark Cluster 77
Figure 16 Results of LR (non-folded) on Cloud Spark Cluster 78
Figure 17 Results of LR (non-folded) with 2 Million cases dataset 80
Figure 18 Results of SVM (non-folded) with 2 Million cases dataset 81
Figure 19 Results of LR (non-folded) with One Million cases dataset 82
Figure 20 Results of SVM (non-folded) with One Million cases dataset 82
Figure 21 Results of LR (non-folded) with Half Million cases dataset 83
Figure 22 Results of SVM (non-folded) with Half Million cases dataset 84
Figure 23 Results of LR re-partition with One Million cases dataset 87
Figure 24 Results of LR re-partition and the number of nodes with One Million cases dataset .. 88

Chapter 1. Introduction

While the meteoric rise of digital technologies, including e-learning, has had a massive impact on education in practically every domain, including learner engagement, teaching management, content generation, and performance evaluation, and more, at the same time, vast volumes of data are being generated by innovated learning facilities that could be meaningful for both academic and scientific analysis. This thesis focuses on applying the data mining and machine learning methods to analyze the data of student's online learning activities with the distributed computing system of Apache Spark. In doing so, it reveals the value of Apache Spark's scalability and the performance in comparison to the big data analytics of online cognitive learning systems.

1.1 Motivation

For decades, experts and educators have been searching for more practical ways to assess student learning outcomes and curricular design without the requirement of human expertise or analysis. Does the knowledge structure meet the requirements of the subject or curriculum with proper difficulties that most of students could accept? Is there any variance in the learning rates between students? Is it possible to design the problems that suit learning outcomes and are based directly on the students' performance data? Currently, K-12 education is focused on assessment using new, high-stakes standards-based tests, as required by the No Child Left Behind Act (NCLB) of the US government. However, this has put incredible pressure on schools to spend significant time and effort to prepare and take these tests, often sacrificing valuable time that would otherwise be spent on deep learning (long-term retention, transfer and building the desire for future learning). The limited classroom time available in school mathematics classes causes a further dilemma in that it compels teachers to choose between time spent assisting students' development and time spent assessing students' abilities [6].

To resolve this dilemma, the U.S. Department of Education built an integrated web-based tutoring system, ASSESSment, that can do assessment and provide assistance at the same time. It offers instruction to students while providing a more detailed evaluation of their abilities to the teacher.

Online tutoring systems for mathematics are based on the cognitive model, which is a set of production rules or skills encoded in intelligent tutors to model how students solve problems. Productions embody the knowledge that students are trying to acquire, and they allow the tutor to estimate each student's learning of each skill as the student works through the exercises [31]. Cognitive tutoring system help students work through complicated problems by breaking them into sub-steps with sub-knowledge components, while simultaneously collecting all the system interaction data of the students' performance, including accuracy, speed and hints times. Used in more than 2,500 schools serving half a million students every year across the U.S., the cognitive tutors generate a vast amount of students' interaction data that can be used as a rich source for making assessment and prediction of both individual students and groups.

Various statistical works and analyses have been developed by experts to help teachers better understand students' performance and progress in order to improve instructions [1]. The important practical fact is that, with designed analyses and improved models, millions of hours of students' time spent learning mathematics could be saved, and desired achievement levels could be increased. From the scientific perspective, discovering methods for accurately predicting students' performance could help to uncover critical underlying factors of curricula and lesson design.

Additionally, the development and application of data mining and machine learning with the use of big data can lessen the amount of expertise required to conduct analyses of cognitive tutoring. The combination of data mining technology with web-based education systems (known as education data mining, or EDM) is becoming an important research area in this regard, as it shows increasingly promising results.

A variety of research has been conducted applying data mining approaches to education; these include predicting the student retention risk and academic performance, curricula design and more [32]. Yadav, Bharadwaj and Pal [33] used a machine learning algorithm to predict the dropping out risk for the first-year students in higher education, while Márquez-Vera, Cano, Romero and Ventura [22] proposed a genetic programming algorithm and different data mining approaches to predict the student failure in high school. Moreover, Kabakchieva [23] presented a method of classification for predicting students' performance in college based on personal and pre-university characteristics.

Recently, cloud-based technologies like Apache Hadoop and Spark have enhanced our ability to conduct such analyses by offering significantly increased computing speed, flexibility and scalability. IT behemoths like Facebook, eBay, IBM and LinkedIn have used the MapReduce framework of Hadoop with the distributed file system HDFS for large volume data storing, managing and processing for many years. Apache Spark, the second generation of Hadoop, is becoming the de facto standard for big data analytics with some of its key features including in memory computing, fault tolerance and data structure abstraction.

1.2 Problem Definition

This thesis is based on the KDD Cup 2010 Educational Data Mining Challenge. KDD Cup is the annual data mining and knowledge discovery competition organized by the Association for Computing Machinery's (ACM) Special Interest Group on Knowledge Discovery and Data Mining (KDD). ACM is the leading professional organization of data miners. This year's competition provided student's interaction log datasets from two Online Intelligent Tutoring Systems. The task was to use the development datasets or training sets to build learning models and accurately predict the students' performance on the test datasets.

Datasets for data mining from educational online learning systems normally have some common characteristics, that make both the data analytics and prediction works difficult to undertake with traditional, computer-based data mining and machine learning workflows. These include the following:

- Variant structured data
- Large amount of categorical features

- Super sparse value matrix
- Imbalanced output
- Streaming data

The volume of data from web-based learning systems has been one of the main concerns for EDM, which needs various computing algorithms and high-level iterations for regression and classification analytics. For example, of the datasets in this thesis, the smallest one is from the Carnegie Learning Algebra system deployed in 2005-2006, which covers only 515 students with 813,661 cases. The data have variant data types, including string, integer, float, time sequences and categorical values, while some of the variables are extremely imbalanced and sparse. Some features, such as *'KC',* which represents the knowledge components of each question, make for an extreme sparse vector matrix with an imbalance in the number of knowledge components in most cases. The feature with the highest number of categorical values is *StepName,* which has more than 180,000 different nominal values, making it difficult to perform the classification algorithms on a normal computing platform. With the tremendous growth of data sources, innovative technologies, including data distribution and parallelization computing, are becoming more appealing to conduct this kind of task of data mining and machine learning with massive data.

Teams from all over the world joined in the competition. Two results attributes reflected the prediction accuracy from the applicants; these were *Cup Score* and *Leaderboard Score*. The *Cup Score* was the evaluation method of using the majority of the prediction files to reach an RMSE score, while the *Leaderboard Score* used the small portion of the prediction files to score the entry.

The teams on the *Cup Score* board took the processes of feature generation, feature selection, latent factor identification, regularization, loss function and ensemble together with a variety of classification and regression methods including Decision Tree, Linear classifier, Non-linear kernel method, Random Forest, Neural Network, Nearest Neighbor, Neighborhood/correlation based collaborative filtering and more. Some teams used very simple classification algorithms and only a portion of the dataset but finally got acceptable prediction results on the board. This phenomenon brings the probability of the information contained in the datasets coming with limitation, so the performance on the accuracy of the prediction could be improved with limited extent even sophisticated data mining processes involved. Nevertheless, almost all of the teams were concerned with the size of the datasets and the running time required for model building and validation. The Zach A. Pardos team, which reached fourth position on the All Teams board, indicated that the running time for the KC model they built with the hardware that included a 30 node rocks cluster with 4 CPUs per node and a 6 node rocks cluster with 8 CPUs per node was two days. This time requirement is neither acceptable nor practical in an industry that is growing at such a fast pace. Therefore, we must find methods to improve the performance of the prediction process. One method involves digging deeply into the processes of data mining algorithms in order to increase prediction accuracy results. This thesis, however, focuses on evaluating the running time and scalability of the popular distributed computing engine Apache Spark working for this large scale big data analytics with computing resources utilization and tuning.

1.3 Significance

Previous research has revealed the potential scalability of the techniques of data mining and machine learning techniques on distributed computing systems when conducting the big data analysis [38][40][54][71][73][75]. The new platform, Apache Spark, based on the most famous distributed computing system, Apache Hadoop, can reach the speeds up to 100x faster than Hadoop MapReduce on most of the computing tasks [72]. Thus, it has emerged as the next generation of big data processing engines and is one of the most scalable and efficient platforms for big data analytics. Spark's machine learning library (MLlib) makes practical machine learning scalable and easy by providing a few of the most commonly used data mining classification and regression algorithms as well as variety of utilities and tools, including feature processing, pipeline tuning and more. In this thesis, four classification methods including Logistic Regression, Linear SVM, Decision Tree and Random Forest in MLlib are coded in Python to predict student's performance based on the massive data from the online cognitive learning systems. The purpose of this thesis is to reveal the scalability of the Apache Spark on big data of education data mining. The experiments were performed on a local Hadoop YARN based Spark cluster and a Google Cloud Hadoop YARN managed Spark cluster. The results of the thesis provide consolidated supports for the scalability and efficiency of Spark in terms of resource allocation and management as well as the run-time tuning for the EDM purposes. While such work continues to require extensive exploration in the future, the findings of this thesis can help education experts to discover important information regarding students' learning and predict future academic performance based on the big data resolution of the students' historical records.

1.4 List of Contributions

This thesis provides some notable contributions to the employment of Apache Spark's distributed computing platform to conduct data mining on big data produced by educational online learning systems.

- Feature analysis and processing: This thesis presents a full path of feature analysis and manipulation during the process of data mining with the datasets from the Online Learning Systems.
- Experiments on the scalability of Spark: A few experiments are made with the Cloud and local YARN based Apache Spark clusters to reveal the scalability and efficiency of the platform of Spark on the tasks of big data analytics with education purpose.
- Resource allocation, utilization and tuning analysis of Spark: A thorough discussion and analysis of the results of Spark resource allocation and utilization as well as run-time parameter tuning for optimization.

1.5 Thesis Outline

The remainder of the thesis is organized as follows:

Chapter 2 is the literature review that introduces the cognitive tutor systems and discusses data mining techniques as well as data mining for educational purposes. In Chapter 3, the most commonly used data mining and data processing methodologies and approaches are introduced, and the process of the data mining presented here is revealed. Chapter 4 introduces the Distributed Computing System of Apache Hadoop and Spark. Chapter 5 provides the methodologies designed for this thesis. In Chapter 6 the results and discussions are presented based on the outputs of the experiments, while Chapter 7 concludes the thesis and summarizes suggestions for future work.

Chapter 2. Literature Review

Emerging data mining and big data technologies have made it possible to explore the potential understanding of the students and the quality of teaching materials using the increasingly large-scale data being produced by the domain of educational technology. The data can come from online interactive learning systems or schools' administrative records and may contain meaningful information for the educators and experts to improve the teaching quality and gain more insight into the design of the learning environments and educational resources.

2.1 Cognitive Learning System Analytics with Data Mining

Cognitive learning refers to how a person processes and reasons information. Cognitive learning systems are intelligent tutoring systems that employ cognitive learning by using a set of production rules or skills to model how students learn and solve problems. With the development and incorporation of innovative technologies, including data mining, machine learning, artificial intelligence and more, online cognitive tutoring systems have the potential to significantly improve education by accelerating learning speed, saving learning time, providing insights that can improve education theories and learning outcomes and more.

2.1.1 Cognitive Tutor System

A cognitive tutor is a type of theory based intelligent tutor. Intelligent tutors draw on artificial intelligence technology to provide interactive instructions that adapt to individual students' needs and, most typically, supports student practice in learning complex problem solving and reasoning skills. The theory of cognitive psychology of problem solving and knowledge components structure for learning experience makes the cognitive tutor system an effective method to evaluate and make prediction for the performance of students [3]. Koedinger and Aleven [3] provided a few examples of experiments within cognitive tutors that explored trade-offs between giving and withholding instructional assistance, which provided support for cognitive tutors to balance the giving and withholding of information and for individual interactive elements.

Cen, Koedinger and Junker [4] proposed a semi-automated method to improve a cognitive model called Learning Factors Analysis (LFA), which was combined with a statistical model, human expertise and combinatorial search, to measure the difficulty and learning rates of knowledge components and to predict student performance. With the statistical method, a multiple regression model was developed to quantify the skills.

2.1.2 Educational Data Mining and Big Data

Data mining is defined as the process of discovering patterns in data. The process must be automatic or semiautomatic. The patterns discovered must be meaningful in that they lead to some advantage, usually an economic one. The data is invariably present in substantial quantities

[2]. Data mining involves various modern technologies including machine learning, database technology and statistics. The purpose of data mining is to discover and reveal the hidden patterns that may have significant importance to industry, business and science as well as normal human life. Enormous efforts have been made in the domain of data mining from the perspectives of learning algorithms, dataset manipulation, feature selection and system design and implementation.

Data mining technology has been applied in the education domain for a long time, and numerous studies involving a variety of learning algorithms have been conducted with the goals of improving learning quality, predicting retention possibilities, discovering learning curves and more. Romero and Ventura [26] surveyed the most relevant studies carried out in the field of education data mining. They introduced EDM and described the diverse groups of users, types of educational environments and the data users provided. They also listed the most typical/common tasks in the educational environment that had been resolved through data mining techniques and discussed some of the most promising future lines of research in EDM.

Ben Daniel's [19] research identified contemporary challenges that institutions of higher education worldwide are facing and explored the potential of big data to address these challenges. Slater, Joksimović, Kovanovic, Baker and Gasevic [29] highlighted some of the most widely used, most accessible, and most powerful tools available for the researchers interested in conducting education data mining/ learning analytics research.

Abuteir and El-Halees [21] conducted a case study about how educational data mining could be used to improve graduate students' performance and to overcome the problem of graduate students' low grades. They conducted four data mining tasks, including association, classification, clustering and outlier detection to present the extracted knowledge and describe its importance in the educational domain.

Feng, Heffernan and Koedinger [6] built a "lean" Rasch model (1-PL IRT model) and used Linear Regression to predict student proficiency and MCAS test scores based on each student's performance history in the ASSISTment Tutor System. They also ran a stepwise regression analysis, called the assistance model, based on the interactions between the student and the system. With these models, new features were generated and fitted into a new stepwise regression model called the mixed model. They concluded that the mixed model made significantly good predictions for the student's performance in the system.

Yu, DiGangi, Jannasch-Pennell and Kaprolet [7] brought in a new perspective by exploring student retention possibilities with three data mining technologies: classification trees, multivariate adaptive regression splines (MARS), and neural networks. They discovered some useful insights into various factors that impact student retention, including transferred hours, residency, and ethnicity.

Ramaswami and Bhaskara [8] built a tree-based CHAID prediction model to predict students' performance based on the dataset from a detailed questionnaire using an experimental methodology. The model indicated that features such as medium of instruction, marks obtained in secondary education, location of school, living area and type of secondary education were the strongest indicators of the students' performance in higher secondary education.

Hung, Hsu and Rice [10] investigated an innovative approach to program evaluation through analyses of student learning logs, demographic data, and end-of-course evaluation surveys in an online K–12 supplemental program. Clustering analysis was applied to reveal the students' characteristics, while decision tree was used to predict student performance and satisfaction level about the course. Their study demonstrated the benefits of incorporating data mining into the program evaluations of K-12 online education.

Vandamme, Meskens and Superby [11] conducted a study in which they classified university students into three groups based on their chances of success in academic learning. They created a questionnaire to collect a large amount of information from students and distributed this questionnaire to first-year students in three French-speaking universities in Belgium. After the feature selection they chose the most significantly correlated variables. Decision trees, neural networks and a linear discriminant analysis were applied to make the prediction.

In this new era, enormous increases in data generation and storage have shifted attention in the domain of data science toward the techniques of big data. The development of distributed and cloud computing and storage systems greatly extends the scalability, as well as the performance of traditional process of data mining and machine learning techniques. Kumar and Rath [71] proposed methods of MapReduce based tests for feature selection along with the MapReduce based proximal support vector machine (mrPSVM) classifier to classify the microarray datasets on the distributed framework of Hadoop cluster with four slave (data) nodes and a conventional system. The performance of the classifier for various datasets was evaluated by varying the number of features. Their experiment presented the significance of distributed computing for better storage and faster processing of datasets, as well as system scalability.

As a part of the Apache Hadoop Ecosystem, Apache Spark was introduced to solve the drawbacks of Hadoop by adding much faster in-memory computing speed, especially for large scale data processing. Maillo, Ramírez, Triguero, and Herrera [72] presented an alternative distributed kNN model for big data classification using Spark, which was denoted as kNN-IS. They reduced the complexity of kNN to *m* tasks without requiring any preprocessing in advance and relied on Spark to reuse the previously split training set with different chunks of the test set. All the operations were performed within the RDD objects provided by Spark. Compared with the traditional way of dealing with large scale data using the algorithm of kNN, the use of Apache Spark has provided them with a simple, transparent and efficient environment to parallelize the kNN algorithm as an iterative MapReduce process.

Scalability has always been a prolific field of study in the areas of data mining and machine learning [73]. Arias, Gamez, and Puerta [73] experimented with the adaptability of the family of Bayesian Network Classifiers (BNCs) to the MapReduce and Apache Spark frameworks. Their proposal focused on the learning stage of such models, for which they introduced a general framework that characterized the full family of BNC classifiers under the MapReduce paradigm. To evaluate their proposal, they conducted a series of experiments over a broad range of both synthetic and real problems on a competitive cluster of computers. Their approach was based on the general framework of learning the probabilistic models from large scale datasets and high dimensional feature space.

Mavridis, and Karatza [38] investigated log file analysis with the cloud computational frameworks Apache Hadoop and Apache Spark to study and compare the performance of the two frameworks in terms of scalability and resource utilization. They used an IaaS (Infrastructure as a Service) to create a private cloud infrastructure and then developed and ran realistic log analysis applications with real log files. They compared the two frameworks, evaluating the performance of execution time, scalability, resource utilization, cost and power consumption. Finally, they reached the conclusion that the rising Spark outmatched Hadoop in almost all cases with faster speed, higher mean utilization for resources and more flexible implementation, while both frameworks offered significant scalability.

2. 2 KDD Competition Paper Review

This thesis is based on the KDD 2010 Cup competition on education data mining. The goal of the competition was to predict the student's first attempt to solve a question with the given datasets as model training and verifying, while the test datasets were supplied to evaluate the correctness of the prediction models.

Yu, et al. [46] were organized as six student sub-teams and each student sub-team extracted different features from the data sets according to their analysis and interpretation of the data. Each team chose different classifiers for learning based on the internal set. The feature engineering approaches can be categorized into two types: sparse feature sets generated by binarization and discretization techniques, and condensed feature sets using simple statistics on the data. Finally, ensemble methods were applied to the testing results from sub-teams. They adopted the LIBLIENAR as an easy-to-use tool to deal with large scale classification problems. The tool supports L2-regularized logistic regression(LR), L2-loss and L1-loss linear support vector machines (SVMs). It inherits many features of the popular SVM library LIBSVM, while it offers better performance and efficiency than LIBSVM on large-scale classification [60]. They used Random Forest, AdaBoost and Logistic Regression as the classifiers, and some linear ensemble approaches including simple averaging, linear SVM and linear regression were applied for the ensemble of results from each student group.

T¨oscher and Jahrer [45] used an ensemble of collaborative filtering technology with neural network blending to fit the competition. They got the idea from the recommender systems with the same characteristics as missing values, big matrix and sparse data. They chose the collaborative filtering method with the algorithms *K* Nearest Neighbor (KNN), Singular Value Decomposition (SVD), Factor Model 1 (FM1), Factor Model 2 (FM2), Factor Model 3 (FM3), Group Factor Model (GFM) and Restricted Boltzmann Machines (RBM). For calculating similarities between users, they used the Pearson correlation between students calculated on the subset of commonly answered steps [45]. Every model was trained and evaluated using 8-fold cross validation. Finally, the neural network with two hidden layers was adopted as the blender to combine the results of all the algorithms.

Pardos and Heffernan got the second place student prize in the competition with their method of combining Bayesian Hidden Markov Models (HMMs) and bagged decision trees. The model learns individualized student specific parameters (learn rate, guess and slip) and then uses these parameters to train skill-specific models. The resulting model, which considers the composition of user and skill parameters outperforms models that only consider parameters of the skill [62]. During the data pre-processing stage, they paid a lot of attention to clean the *Step duration* variable, which reflected the time spent on the *step*. Bayesian Networks were used to model students' knowledge over time. Based on the parameters of the HMM for that skill and the student's past responses, a probability of knowledge was inferred [62]. The new feature sets generated by HMM were brought into Random Forest for training and making predictions

Shen et al. team [48] came in third in the student competition with the framework of a vague prediction procedure. First, they built a scoring machine from the training set. The scoring machine was designed to return a score vector when given a test record (that is, a test set step record), and the scores in the returned vector were intended to reflect relevant information from the *Correct First Attempt*, including student performance, step difficulty and more. Second, they used the scoring machine to compute a score vector for every test record. Third, they made predictions from the score vectors.

Tabandeh and Sami [49] used a relatively simply framework to come in fourth in the student competition. During the feature selection stage, they removed all the features that did not appear on the test data and made a conversion algorithm that converted highly categorical features to numerical ones based on their "percentages of positive class instances". To reduce the running time and avoid the limitation of the hardware requirement on big volume datasets, they sampled the data by deleting one-third and one-seventh of all data. Then, C4.5 and Linear Regression algorithms were applied to predict the student's performance.

Chapter 3. Data Mining Methodologies

Data mining is the computing process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems[38]. The overall goal of data mining is to discover information from the existing data and interpret it into an understandable format with reasonable computing costs. The era of big data is coming, which brings increasing sources of data that can be used to generate meaningful knowledge from hidden patterns, while the fast-developing progresses of both hardware and software computer technologies combined with various data mining algorithms make it possible to discover and fully use of that hidden information in the massive data. Various data mining and statistical algorithms have been designed and deployed to build analytic and predictive models across a variety of industries and domains. Some most popular ones, including Logistic Regression, k-NN, Decision Tree, Support Vector Machine, Random Forest and more, are widely applied and have made signification contributions to the development of data science.

3.1 Data Mining Process

Business-driven needs push the fast development of processes in data mining technology, including business understanding, data source retrieval, data manipulation, feature selection, model building and output prediction. A simple chart in Figure 1 shows the working process of data mining:

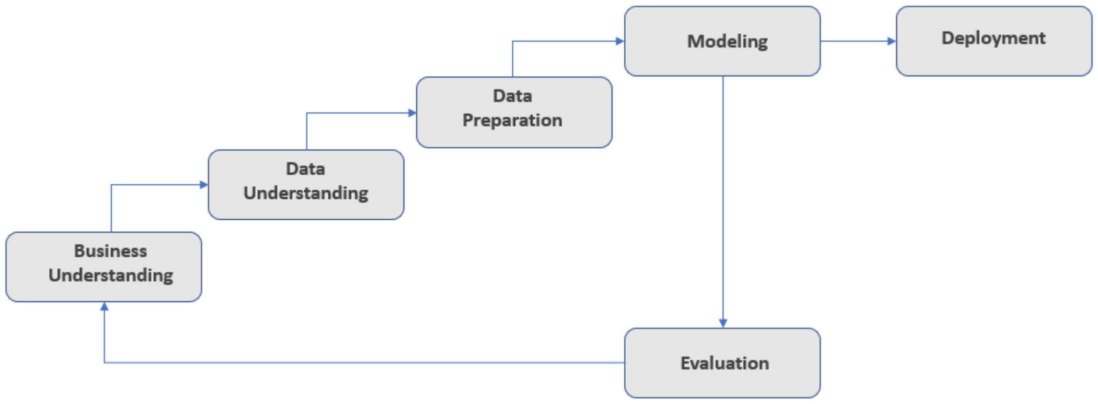


Figure 1 Data mining working process

The process of data preparation or data pre-processing can be divided into several sub-processes, including data cleaning, data integration, feature selection and data transformation. Among them, feature selection has always been of greatest interest with the fast growth of real-world data sources at both the dimensional aspect and mass quantity side.

Modeling is the process through which various software and modeling techniques are selected and applied with tuned parameters to find optimal solutions. With different forms of data and final requirements, different techniques and algorithms should be used to build models. The most well-known data mining models are normally classified as supervised models and un-supervised models:

- Regression is the most straight forward and well known predictive model. Linear Regression and Logistic Regression are the most popular ones used in a variety of domains.
- Association Rule Discovery is a rule-based machine learning method for discovering relationships between variables in large databases [50].

- Classification is used to classify big data according to the format and characteristics of the data. Many classification methods and algorithms are used based on the application selected and the problem that needs to be solved. Some of the most popular supervised classification methods, including Decision Tree (DT) and Support Vector Machine (SVM) are used widely on a variety of data mining applications and platforms.
- Clustering is the un-supervised method of grouping of sets of data based on their characteristics and similarities. Within a clustering model, clustering algorithms can be categorized into different models. Among them, the most well-known model is *k*-means clustering.

Model evaluation is an important part of the data mining process, because it is through evaluation that the best model and parameters for the future stages are found. There are two most commonly used methods of model evaluation in data science are Hold-Out and Cross-Validation. The former method randomly divides the dataset into three subsets called training set, validation set and test set. The latter method is known as *k*-fold cross-validation, which divides the dataset into *k* equal size subsets and leaves out *k-1* of the subset for training with the test set.

After the model is validated and evaluated, it will be produced and implemented with other processes, such as reporting systems or predicting applications. New data will be applied to the final production, and the results will be used for the business decisions or other activities.

3.2 Data Preparation

Data preparation is one of the most important parts of the data mining workflow. For one or more of the following reasons, raw data is not normally ready for immediate analysis by data-mining computing algorithms: data is not clean or missing values, data format needs to meet the input requirements of analysis algorithms, data has noise, input/output values are imbalanced and more. In most cases, the data mining and machine learning results can be improved markedly by suitable manipulations of the data before analysis. In this section, some of the most commonly used data preparation methods for data mining are illustrated.

3.2.1 Data Transformation

During data transformation, data are transformed or integrated into appropriate forms for data mining using different methods and strategies as well as software or computing tools. The general process for data transformation can be broken down as follows:

- Data discovery: The first step of the process, when data are profiled with designed structures.
- Data mapping: The process of defining how individual fields are manipulated to the final output
- Code generation: The process of generating or producing executable code that transforms data into the desired format or output following the designed data mapping rules.
- Code executing: The process of executing the generated code on the data to get the desired output.

- Data review: The process of ensuring the output data meets the requirements.

In the process of data mining, data transformation methods and strategies vary depending on business scenarios and the data characteristics that are being analyzed.

3.2.2 Data Discretization

Data discretization is an essential step if the input dataset variables involve numerical or continuous format and the chosen learning schemes or algorithms can only deal with categorical attributes. In the real world, many data mining or machine learning tasks come with the continuous attributes. When working with categorical attributes, some induction algorithms significantly increase the computing speed compared with continuous ones. In practice, discretization can be viewed as a data reduction method, since it maps data from a huge spectrum of numerical values to a greatly reduced subset of discrete values [55]. Continuous variable discretization has received significant attention in the data mining and machine learning community; for example, the Decision Tree C4.5 algorithm will discretize numerical variables during the learning process. There are a variety of categories of discretization methods based on their characteristics; these include static-dynamic, univariate-multivariate, supervised-unsupervised, splitting-merging-hybrid, global-local, direct-incremental, and evaluation measure. Analogous to supervised and unsupervised learning methods, the supervised-unsupervised data discretization methods are the most widely adopted in the data mining process. Supervised methods, such as error-based, entropy-based or statistics-based, consider the class values, while the unsupervised methods, such as equal-width and equal-frequency based, only focus on the attributes being discretized. Even in a classification process that involves more supervised discretization methods, there are still many cases in which unsupervised methods offer equal or better performance. It is of vital importance to select the proper discretization methods based on the chosen datasets and the learning algorithms.

3.2.3 Feature Selection

The feature selection process in data mining or machine learning is the process of selecting subsets of relevant variables or features for the model building. Feature selection is one of the most important processes in the data mining and machine learning workflow. With proper methods, it could reduce the number of variables for building the model, remove irrelevant and redundant data, clean the noise data, etc. In most cases, feature selection can significantly speed up the computing time for data mining algorithms, improve the accuracy or performance of the model and result in comprehensibility.

Normally there are three categories of feature selection methods: *Filter-based, Wrapper-based* and *Embedded-based* ones. The *Filter-based* methods do feature selection and evaluation before the data mining algorithms are applied and they are independent of the algorithms used in the data mining. In most cases, the feature relevance scores are calculated, and those features with low relevance scores and low correlation results are removed. Afterward, the features left will build the subset of features for the final data mining algorithms. The advantage of *Filter-based* feature selection is simple and fast, while the disadvantage side is that it ignores the interaction with the data mining algorithms, which may lead to the worse performance of the data mining output.

The *Wrapper-based* feature selection approaches embed the data mining algorithms hypothesis with the feature subset search. This method generates all the possible feature subsets and evaluates them by putting them into the data mining algorithms and gets the classification results. Then finally a tailored subset of the data mining algorithms is obtained considering the evaluation of the results. The advantage of the *Wrapper-based* approach is the interaction between selected feature subset and the classification model, which may lead to better performance, while the drawback is the problem of overfitting and the high computational cost.
The *Embedded-based* method is the way embedding the search for feature subset into the classifier construction. The advantage of *Embedded-based* is that it builds the interaction with the classification models, while at the same time needs less computational cost than *Wrapper-based* approach.

3.2.4 Imbalanced and Sparse Data

In many supervised learning applications, there is a significant difference between the prior probabilities of different classes, i.e., between the probabilities with which an example belongs to the different classes of the classification problem. This situation is known as the class imbalance problem, and it is common in many real problems from telecommunications, web, finance-world, ecology, biology, medicine not only, and which can be considered one of the top problems in data mining today [57]. The problem with imbalanced data is that many standard classification learning algorithms tend to generate biased results toward the majority class while leading to higher misclassification for the minority class. For imbalanced datasets several categories were brought out as follows:

- Data sampling: It is the method of sampling the imbalanced data into balanced class distribution dataset as the input of the learning algorithms.
- Algorithmic modification: This procedure is oriented towards the adaptation of base learning methods to be more attuned to class imbalance issues.
- Cost-sensitive learning: This type of solution incorporates approaches at the data level, at the algorithmic level, or at both levels combined, considering higher costs for the misclassification of examples of the positive class with respect to the negative class, and therefore, trying to minimize higher cost errors [57].

In data science, sparse data are also named as sparse matrix or sparse array, meaning in the matrix, most of the elements are zero, while on the other hand, if most of the elements are nonzero, then the matrix can be considered as dense data. In many data mining and machine learning cases especially those with high dimensional variables, the sparse datasets are often applied with specialized algorithms and data structures, which will take advantage of the sparse matrix structure.

3.2.5 Missing Values

Missing values are frequently indicated by out-of-range entries: perhaps a negative number (e.g., –1) in a numerical field that is normally only positive, or a 0 in a numerical field that can never normally be 0. For nominal attributes, missing values can be indicated by blanks or dashes [2]. There are a variety of reasons that cause missing values such as the fault of measure facilities, no response or information provided or the change of data collecting methods and more.

There are some approaches that could be applied to the data of missing values:

- Discard instances: It is the simple solution to discard the instances with missing values. This is the method appropriate for those cases that the missing values are completely at random.
- To get the missing values: It is the method to obtain the missing values with extra costs such as from the third party.
- Imputation: It is the methods to make estimation or prediction of the missing values following their distribution or through designed models. Some well-known treatment named Multiple Imputation has emerged to deal with the missing value problem with the method of generating multiple simulated versions of data sets where each is analyzed, and the results are combined to generate inference [58].
- Reduced-feature Models: It is an alternative approach to Imputation, which incorporates only attributes that are known for the test instance [58].

3.2.6 Normalization and Standardization

Both of normalization and standardization are referred as scaling the attribute values to fit in a specific range. In most cases, they are important parts during the pre-processing stage of data mining. Some measurements like Euclidean distance are sensitive to the variable difference in the magnitudes of scales, so the methods of normalization and standardization are used to scale down the magnitudes while keeping the equal weighting information of the variables. The most commonly used normalization techniques are Min-Max, Z-Score, and Decimal Scaling normalization. The Min-Max method of normalization performs a linear alteration on the original data which could be described as [74]:

$$v_{new} = \frac{v - Min_A}{Max_A - Min_A}(New\_Max_A - New\_Min_A) + New\_Min_A \quad (1)$$

The value v of an attribute A with range $[Min_A, Max_A]$ is mapped to a new range $[New\_Min_A, New\_Max_A]$. The Z-Score or Standardization is the method to map data based on the mean and standard deviation. The formula is [74]:

$$v_{new} = \frac{v - \bar{v}}{\sigma} \quad (2)$$

Here $\bar{v}$ is the mean of the feature, and σ is the standard deviation of the feature.

The Decimal Scaling method just simply scales the data by moving the decimal point of the value of the features needed to be normalized, which could be described as [74]:

$$v_{new} = \frac{v}{10^j} \quad (3)$$

where j is the smallest integer so that $Max(|v_{new}|) < 1$.

In the data mining process, which method of normalization and standardization to choose depends on the characteristics of the datasets and the learning algorithms which will be applied. Typically, normalization will reach the output range of values between 0 to 1, while standardization is to measure how much the values deviate to the *means* with the assumption of data having a Gaussian distribution.

3.3 Data Mining Algorithms

3.3.1 Logistic Regression

Logistic regression is a mathematical modeling approach that can be used to measure the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function [30]. Logistic Regression could be classified with the outcome as binominal(binary), ordinal and multi-nominal logistic regression, and it is used in various domains including machine learning, medical science, social science and more.

The logistic regression is based on the logistic function. In [30], the logistic model is detailed described as:

$$z = \alpha + \beta_1 X_1 + \beta_2 X_2 + \cdots \beta_k X_k \tag{1}$$

$$f(z) = \frac{1}{1 + e^{-(\alpha + \sum \beta_i X_i)}} \tag{2}$$

The linear regression model z is the dependent variable, while the $X_1 \ldots X_k$ are independent variables. The $f(z)$ will get the probability of the output of 0~1, which could be denoted by the conditional probability statement:

$$P(D = 1 | X_1, X_2, \cdots X_k) \tag{3}$$

$$P(X) < \mu \rightarrow' 0' \tag{4}$$

$$P(X) > \mu \rightarrow '1' \quad (5)$$

In this thesis, the prediction output is a binary variable and the binominal logistic regression model is built as the main algorithm to evaluate computing resources cost.

3.3.2 Decision Tree and Random Forest

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences in business or computer programming. The branches of the tree represent choices with associated risks, costs, results, possibilities, event outcomes, resource and more. Based on the approach of top-down strategy and classification rules, the family of the decision tree is constructed with the root of the tree and proceeding to the leaves through the values of a set of attributes or properties. The decision tree is a very effective method of supervised data learning method.

The algorithm ID3 (Iterative Dichotomiser 3) is one of a series of programs developed from Concept Learning System CLS framework (Hunt, Marin and Stone, 1966) in response to a challenging induction task posed by Donald Michie, viz. to decide from pattern-based features alone whether a particular chess position in the King-Rook vs King-Knight endgame is lost for the Knight's side in a fixed number of ply [34]. ID3 was designed where there are many attributes and the training set contains many objects, but a reasonably good decision tree is required without much computation. The basic structure of ID3 is iterative. A subset of the training set called the window is chosen at random and a decision tree is formed from it. This tree correctly classifies all objects in the window. All other objects in the training set are then classified using the tree. If the tree gives the correct answer for all these objects, then it is correct for the entire training set and the process terminates. If not, a selection of the incorrectly classified objects is added to the window and the process continues [34]. The ID3 algorithm begins with the original set *S* as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set *S* and calculates the *entropy H(S)* or *information gain IG(S)* of that attribute. It then selects the attribute which has the smallest *entropy* (or largest *information gain*) value. The set *S* is then split by the selected attribute to produce subsets of the data. The algorithm continues to recur on each subset and only selects those attributes never selected before. The *entropy H(S)* is measured with the method as [76]:

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x) \tag{6}$$

S– the dataset to be calculated

X – set of classes in S

p(x) – the portion of element class *x* in the dataset of *S*

The *information gain IG(S)* is measured with the method as [76]:

$$IG(A, S) = H(S) - \sum_{t \in T} p(t)\, H(t) \tag{7}$$

H(S)– the *entropy* of dataset *S*

T– the subsets created from splitting dataset *S* by attribute A

p(t) – the portion of number of element *t* in the dataset of *S*

H(t)– the *entropy* of dataset *t*

The algorithm of C4.5 is an extension of the earlier ID3 algorithm [2]. It uses Information gain as the splitting criterion, while it has some improvements over the classical algorithm ID3 including numerical values, pruning procedure and handling missing values and more. The splitting criterion is based on the *information value* or *entropy*:

$$entropy(p_1, p_2, \cdots p_n) = -p_1 log p_1 - p_2 log p_2 \cdots - p_n log p_n \qquad (8)$$

The multistage decision information value can be described as:

$$entropy(p, q, r) = entropy(p, q + r) + (q + r) \times entropy(\frac{q}{q + r}, \frac{r}{q + r}) \qquad (9)$$

Random forest is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and the same distribution for all trees in the forest. A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathbf{x}, \Theta_k), k = 1, \ldots\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input **x** [35]. Random Forest algorithm was developed as the extension of Bagging of Classification Tree and has been demonstrated to have excellent performance compared to other machine learning algorithms. Bagging predictor is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class. The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets [52].

3.3.3 Support Vector Machine

The original support vector machine SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963 [78]. Then in 1992, a way to create non-linear classifier by applying the kernel trick o maximum margin hyperplane was proposed by Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik [77]. The SVM is a kernel-based computing algorithm that assigns labels to cases based on the margin maximization hyperplane principle by learning the characteristics of the examples. The instances that have the least distances to the maximum margin hyperplane are called support vectors, while the set of support vectors could uniquely define the maximum margin hyperplane for the learning problem [2]. With the example of two attributes case, a hyperplane separating the two classes could be described as [2]:

$$x = w_0 + w_1 a_1 + w_2 a_2 \tag{8}$$

The two attributes are a_1 and a_2, while the three weights w_0 , w_1 and w_2 need to be learned. In terms of support vectors, a y with the class values of either 1 or -1 is designed to the maximum margin hyperplane concept as [2]:

$$x = b + \sum_{i\ is\ suport\ vector} a_i y_i\, \alpha(i) \cdot \alpha \tag{9}$$

In equation (9) y_i is the class value of the training instance $\alpha(i)$, while b and a_i are numerical parameters that need to be generated by the learning algorithm. The term of $\alpha(i) \cdot \alpha$ represents the dot product of the test instance with one of the support vectors: $(i) \cdot \alpha = \sum_j \alpha(i)_j \alpha_j$. Finally, b and a_i are parameters that determine the hyperplane [2]. Then it leads to the problem of finding the support vectors with the training sets and determine the parameter of b and a_i, which is a typical optimization problem.

3.3.4 Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than those in other groups (clusters). It can be stated as follows: given a representation of *n* objects, find *k* groups based on a measure of similarity such that the similarities between objects in the same group are high while the similarities between objects in separate groups are low [43]. Clustering analysis has the main task of exploratory data mining, and it is one of the common techniques for statistical data analysis. As an unsupervised data mining method, clustering is used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

A variety of clustering models were employed by researchers with the concept of “a group of data objects”, which led to different clustering algorithms based on these models. Some popular ones are Connectivity-based clustering (*hierarchical clustering*), Centroid-based clustering (*k-means clustering*), Distribution-based clustering (*EM clustering*), Density-based clustering(*DBSCAN*), etc. Among all the algorithms, the most popular and simplest one is *k-means Clustering* due to its ease of implementation, simplicity, efficiency, and empirical success in a variety of domains for more than 50 years.

The algorithm *k-means Clustering* could be described as follows. Let $X = \{x_i\}, i = 1, \dots n$ be the set of *n* d-dimensional points to be clustered into a set of *K* clusters, $C = \{c_k , k = 1, \dots K\}$. The *k-means* algorithm finds a partition such that the squared error between the empirical mean of a cluster and the points in the cluster is minimized. Let μ_k be the mean of cluster c_k. The squared error between μ_k and the points in cluster c_k is defined as [43]:

$$J(c_k) = \sum_{x_i \in c_k} \|x_i - \mu_k\| \tag{10}$$

The goal of *k-means* is to minimize the sum of the squared error over all *K* clusters,

$$J(C) = \sum_{k=1}^{K} \sum_{x_i \in c_k} \|x_i - \mu_k\|^2 \tag{11}$$

Anil [43] described the main steps of *k-means* algorithm as follows:

1. Select an initial partition with *K* clusters; repeat steps 2 and 3 until cluster membership stabilizes.

2. Generate a new partition by assigning each pattern to its closest cluster center.

3. Compute new cluster centers.

Chapter 4. Big Data with Distributed Systems

The continuous increase of the volume and feature dimensions captured by organizations, such as the rise of social media, Internet of Things (IoT), and multimedia, has produced an overwhelming flow of data in either structured or unstructured format [53]. The rise of big data brought tremendous opportunities and conveniences, on the other hand, some challenges came with it including the scalability, data storage, analytics speed based on the computer architecture and more. The traditional centralized data mining technologies in some cases are not appropriate for the increasing new challenges. Then the distributed data mining environments and cloud computing architectures offer better solutions for the need of faster and safer data mining techniques. The distributed systems have the distributed computational units connected and organized by network, which meets the requirement of large-scale and high-performance computing. There are many kinds of distributed systems like Grids, Cloud Computing Systems, P2P systems, and etc. The distributed systems made a significant improvement to data mining on both scalability and performance. Apache Hadoop and Apache Spark are becoming the most popular ones used across various domains.

4.1 Apache Hadoop

Apache Hadoop was first developed in 2003 based on the Apache Nutch project [79]. The genesis of Hadoop originated from the Google File System paper published in October 2003.

Hadoop is the open source software framework used for distributed storage and processing of dataset of big data using the MapReduce computational paradigm. It is one of the most well-established software platforms that support data-intensive distributed applications [54]. There are four main parts of the framework: Hadoop Common, Hadoop Distributed File System(HDFS), Hadoop YARN and Hadoop MapReduce. Based on this basic ecosystem, some other software package or models were developed and installed such as Apache Pig, Apache Hive, Apache HBase, Apache Spark, and etc. which built the whole Apache Hadoop Ecosystem shown in Figure 2.

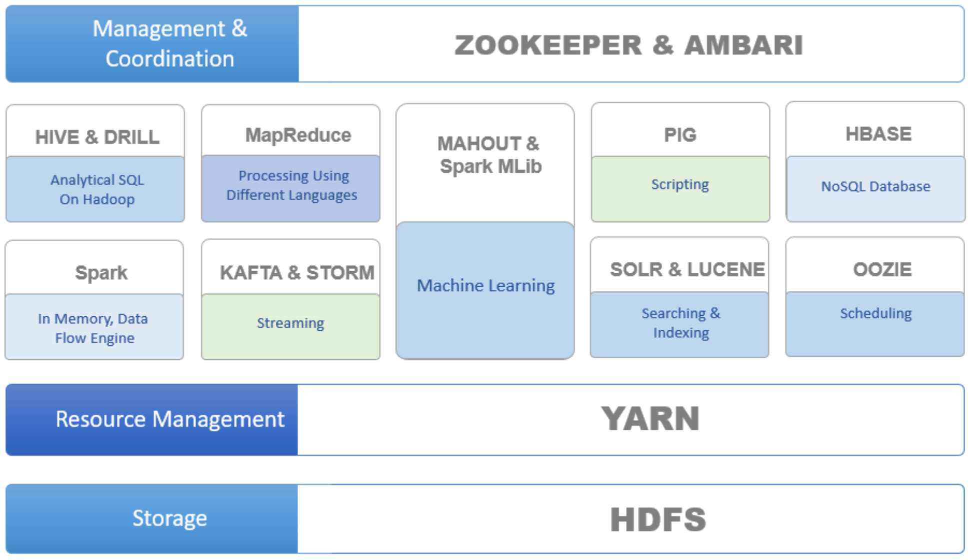


Figure 2 Apache Hadoop Ecosystem

The Hadoop Distributed File System(HDFS) was developed as a distributed and scalable file system that stores large files across multiple machines. It links together the whole files on local nodes building one single file system. HDFS has the features of scalable, fault-tolerant, flexibility and reliability, which makes HDFS a fault-tolerant and self-healing distributed file system to a massively scalable pool of storage. HDFS divides files into smaller blocks and stores them in different nodes of Hadoop clusters, which makes it possible to store much bigger size files than the disk capacity of each node.

The MapReduce Framework was developed by Google with the highly distributed large clusters of computers to automatically process large-scale datasets. MapReduce is based on the *divide and conquer* method and works by recursively breaking down a complex problem into many sub-problems until these sub-problems is scalable for solving directly. After that, these sub-problems are assigned to a cluster of working notes and solved in separate and parallel ways. Finally, the solutions to the sub-problems are then combined to give a solution to the original problem [54]. The framework is designed on the base of two phases: Mapping and Reducing. As a revolutionary programming model, MapReduce was designed as the Map function processing the Key/Value pair to generate the set of intermediate pairs, while the Reduce function merging these intermediate pairs with the matching keys and distributing them across nodes to independent processing. With the MapReduce model, Hadoop becomes a powerful framework for easily writing applications which process vast quantities of data in parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner [54]. The MapReduce Framework has features of accessibility, flexibility, reliability and scalable, which makes it a popular batch-processing tool for big data.

The Apache Mahout aims to provide scalable and commercial machine learning techniques for large-scale and intelligent data analysis applications [54]. Apache Mahout has some core machine learning algorithms for clustering, classification, batch based collaborative filtering which is implemented on the base of Apache Hadoop using MapReduce method.

Apache YARN is the foundation of the 2nd generation of Hadoop that enables users on various locations to explore the modern data architecture with multiple data processing engines. Thanks to the decoupling of resource management and programming framework, YARN provides 1) greater scalability, 2) higher efficiency, and 3) enables a large number of different frameworks to efficiently share a cluster. These claims are substantiated both experimentally (via benchmarks), and by presenting a massive-scale production experience of Yahoo!—which is now 100% running on YARN [55].

4.2 Apache Spark

Apache Spark is an open source in-memory computing, cluster-based data processing API framework. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. Spark runs on the Apache Hadoop YARN and provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance. Spark brought out the new data structure system named Resilient Distributed Dataset (RDD) to respond to limitations of MapReduce cluster computing paradigm. An RDD is a read-only, partitioned collection of records. RDDs provide fault-tolerant, parallel data structures that let users store data explicitly on disk or in memory, control its partitioning and manipulate it using a rich set of operators [41]. Instead of with disk to perform Input & Output functional manipulations, Spark offers users a fast and versatile processing structure and storage model based on the distributed shared memory, which boasted 100 times faster than Hadoop.

There are three main deployment architectures of Spark which are [80]:

Standalone Cluster: It is the simplest deployment of Spark Cluster with the static nodes of master and workers running with MapReduce in parallel. The master node manages and drives all the worker nodes and resources, while the worker nodes take on the task of computing and data processing. The Standalone Cluster deployment mode can also run alongside with the existing Apache Hadoop deployment and access to the Hadoop Distributed File System (HDFS). The Standalone Cluster mode gracefully supports the failure of the worker nodes. With the help of Apache ZooKeeper, it also has the ability to use multiple standby master nodes as a backup, which gives high availability to the production jobs.

Spark on Hadoop YARN: The main benefit of this Spark deployment mode is the dynamically sharing the cluster resources between different frameworks that running on YARN. Spark can also get support from YARN schedulers and the cluster security management. Running Spark on Hadoop YARN can make it directly and quickly to access the data stored in HDFS on the same nodes.

Spark on Apache Mesos: As a general-purpose cluster manager that can run both analytical jobs and long-term services on the cluster, the Mesos master will replace the Spark master as the cluster manager in this mode, which will bring the advantages of dynamic partitioning between Spark and other frameworks and scalability between multiple Spark instances.

Based on the structure of RDD, Spark brings out the strategy of series of parallel operations called: Transformations and Actions. Transformations are deterministic but lazy, operations which define a new RDD without immediately computing it [40]. The processes of actions will launch the computation with the RDD finally and return the results to shared memory or disks.

Transformations are only really executed when an action is called. With this strategy, Spark could break the computation into tasks to run paralleled on separated machines. With the function of the pipeline, these tasks are organized into multiple stages and separated by distributed shuffle operations for redistributing data [40].

Spark extends the popular MapReduce model and supports the combination of a wider range of data processing techniques, such as SQL-type queries and data flow processing. For ease of use, Spark has Python, Java, Scala and SQL APIs [38]. Since its release, Apache Spark has seen rapid adoption by enterprises across a wide range of domains. Some big brand online platforms such as Netflix, Yahoo, and eBay have deployed Spark at massive scale, processing multiple petabytes of data on clusters of over 8,000 nodes. Spark grows quickly that became the largest open source community in big data domains. Spark was designed with features for data mining and knowledge learning. Spark MLlib provides a library with a growing set of machine learning algorithms and utilities for data science techniques including classification, regression, clustering, collaborative filtering and feature selection as well as model evaluation utilities and tools.

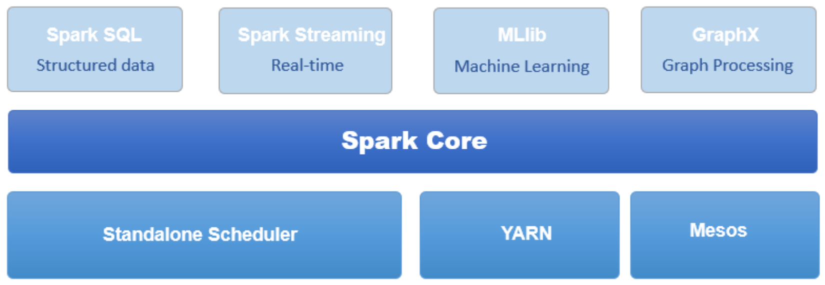


Figure 3 Apache Spark Stack

The ecosystem of Spark empowers it a powerful open source processing engine with fast speed, ease of use, and data science analytics. The components build the Spark ecosystem are illustrated as Apache Spark Stack showed in Figure 3:

- Apache Spark Core API: It is the foundation of parallel and distributed processing component of the huge datasets. All the functional manipulations by Spark are built on the top of Spark Core. It has the capabilities of fast speed in-memory computing and generalized execution models which support a wide variety of application like Scala, Java, Python, and R for the convenient of development.
- Apache Spark SQL + DataFrames: It is designed for structured data processing with the abstract concept of DataFrames based on the RDD. It acts as the distributed SQL query engine with extraordinary faster running speed than former unmodified Hadoop Hive. It also provides the integration with other components of the Spark ecosystem.
- GraphX: It is a graph computation API built on top of Spark that enables users to interactively build, transform and analyze graph-structured data with remarkable speed while retaining Spark's flexibility, fault tolerance and ease of use.
- Streaming Analytics: It enables the real-time processing the interactive and analytical applications with both streaming and historical data. It also perfectly integrates with most popular data source like HDFS, Twitter, Flume, and Kafka.
- MLlib: It is a scalable machine learning library that is implicitly suitable for iterative processes with both high-performance algorithms and computing speed. The library was developed with Java, Scala and Python, and could be easily included into the application workflow. MLlib contains a large set of popular learning algorithms with the new versions mainly based on DataFrames including Classification, Regression, Clustering and Feature selections and more.

For machine learning, Apache Spark supplies the MLlib scalable machine learning library, which provides machine learning algorithms and tools list below:

- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and loading algorithms, models, and Pipelines
- Utilities: linear algebra, statistics, data handling, etc.

Spark's MLlib has two main packages: *spark.mllib* and *spark.ml*. The former one was designed on the top of RDD based API and is only in maintenance mode, while the latter one is based on DataFrames with the new Pipelines API and some other new features and will be the primary API. The DataFrames is conceptually equivalent to a table in Relational Database, but it can be constructed from a wide array of sources including structured data files, tables in Hive, external databases, or existing RDDs.

Spark provides utilities for data mining and knowledge learning including: Summary Statistics (*mean, variance, count, max*, etc.), Correlation (*Spearman* and *Pearson* correlation), Stratified Sampling (*sampleByKey* and *sampleByKeyExact*), Hypothesis Testing (Pearson's chi-squared test) and Random Data Generation (RandomRDDs, Normal, Poisson), Kernel Density Estimation (*kernelDensity*).

Spark MLlib supports various methods for classification and regression with RDD-based API. For binary classification and multiclass classification there are *linear SVMs*, *logistic regression*, *decision trees, random forests, gradient-boosted trees, naive Bayes* methods, and for *Regression linear least squares, Lasso, ridge regression, decision trees, random forests, gradient-boosted trees*, *isotonic regression* are also available for data mining usage.

Spark MLlib supports *ALSModel* as the model-based collaborative filtering method for the popular recommender system, in which users and products are described as small sets of latent factors and these factors can be used to predict missing entries for various usages.

For unsupervised clustering both the DataFrames based API and RDD based API have some most popular methods like *K-means, Latent Dirichlet allocation(LDA), Bisecting k-means, Gaussian Mixture Model(GMM)* algorithms implemented in Scala, Java, and Python.

Spark MLlib also supplies a set of methods and algorithms for extracting, transforming and selecting features in the process of data mining. For feature extraction, methods of *TF-IDF, Word2Vec, CountVectorizer* are some popular ones in text-mining domains. Various methods for feature transformation are supported such as *Tokenizer*, *StringIndexer, VectorIndexer*, and etc. Three feature selection methods are included in the MLlib which are *VectorSlicer, RFormula* and *ChiSqSelector*.

The concept of ML Pipelines provides a uniform set of high-level APIs built on top of DataFrames that help users create and tune practical machine learning pipelines (spark.apache.org). The API enables the combination of multiple algorithms and data mining processes into one pipeline workflow. The Pipelines in Spark bring the concepts of Transformer and Estimator. The transformer is an algorithm or data mining step which can transform one DataFrame into another DataFrame, while an Estimator fits the DataFrame to the learning algorithm model with parameters. Then the pipeline chains multiple Transformers and Estimators together to form a workflow with a common API for specifying parameters.

Spark MLlib provides tools for tuning ML algorithms and Pipelines. Users can tune the entire pipeline at once instead of tuning each component in the pipeline separately. Two methods *CrossValidator* and *TrainValidationSplit* work as the tuning tools with some other methods including *Estimator*, *ParamMaps*, and *Evaluator*.

Chapter 5. Methodology

The datasets in this thesis are from Intelligent Tutoring Systems (ITS) for thousands of students in the US with the spanning time of 2005-2009 school years. The competition was to predict the students' future performance of solving algebra problems with their historical data on the ITS. The variable representing the ability of the student for the specific problem is *Correct_First_Attemp*, which has binary classes with '0' for correct and '1' for incorrect. All the participants of the competition need to build the prediction model with three training datasets and two development datasets which are for the further prediction. The participants' finished models will be tested with a test data and the evaluation of their prediction accuracy will be compared with the real output with the RMSE method.

The datasets for this competition have a large volume. Some attributes have very high categorical distributions. The description of the datasets is listed as:

Datasets	Students	Steps	Attributes
Development Data Sets			
Algebra I 2005-2006	575	813,661	22
Algebra I 2006-2007	1840	2,289,726	22
Bridge to Algebra 2006-2007	1146	3,656,871	20
Challenge Data Sets			
Algebra I 2008-2009	3310	9,426,966	22
Bridge to Algebra 2008-2009	6043	20,768,884	22

Table 1 KDD Competition Datasets

To evaluate and reveal the scalability of the architecture of Apache spark on predicting the performance of students with the online cognitive learning systems, the methodology designed for this thesis is to apply different classification and regression algorithms with designed computing resources allocated including the number of executor nodes, nodes memory, executor cores and dataset partition with the Apache Spark cluster. By analyzing and comparing the experiment results, the advantages, as well as the bottleneck of distributed system Apache Spark on education data mining, can be revealed. The design of the methodology for this experiment could be described in Figure 4:

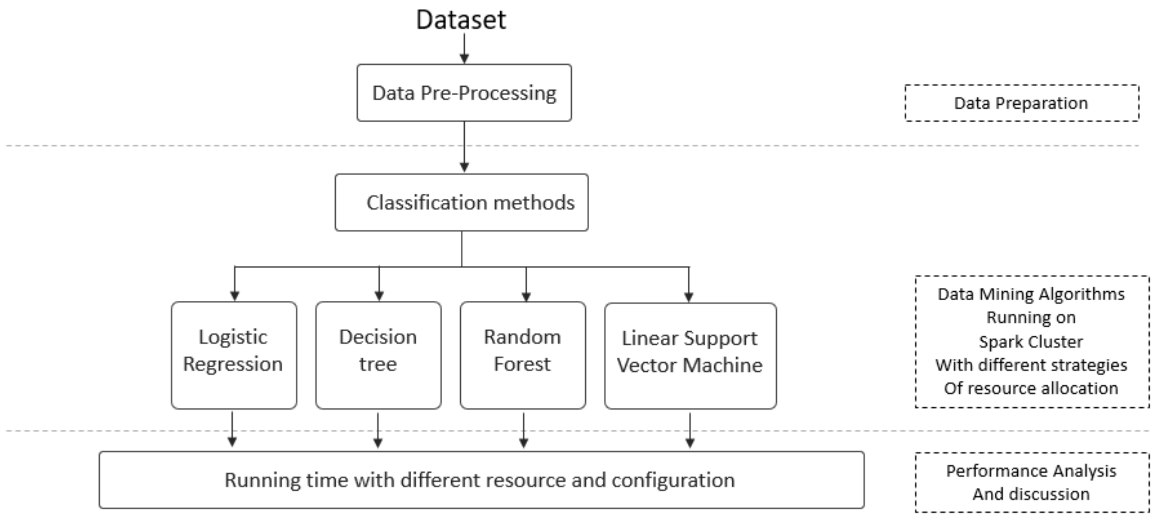

Dataset
Data Pre-Processing
Data Preparation
Classification methods
Logistic Regression
Decision tree
Random Forest
Linear Support Vector Machine
Data Mining Algorithms Running on Spark Cluster With different strategies Of resource allocation
Running time with different resource and configuration
Performance Analysis And discussion

Figure 4 Design of the experiments

5.1 Data Pre-processing

To guarantee the accuracy of the predicting algorithms implemented on the stage of training and predicting, also to meet the requirements of different data mining algorithms in Spark, the raw datasets need to be pre-processed before the process of analyzing with Spark.

5.1.1 Feature Generation

The raw datasets have the feature named *KC* which means the knowledge components separated with '~~' for the problem. In according with *KC* there is the feature of *Opportunity* with same structure indicating the count of this student encounter of the *KCs*. Both variables need to be broken up into separate variables of each knowledge component and related times of encounter of the knowledge component. A new variable needs to be generated which has the number of knowledge components in the problem. It is possible that the number of knowledge components may lead to higher level of difficulty of the problem and less chance to answer it correctly.

New groups of features reflecting student's ability on mathematics and the difficulty level of the problem will be generated at this stage, which is listed below:

- The Student group: It reflects the ability of the student on math based in his/her history data. New generated features come from the statistical count of the student's history counts of problems solved, Hints times, Incorrect answers and Correct answers based on the students ID.
- Problem Name group: It reflects the historical records of the problem from the variable of *Problem Name*. New generated features come from the statistical count of the *Problem Name* total number, Hints times, Incorrect answers and Correct answers.
- Problem Unit group: It reflects the historical records of the problem from the variable of *Problem Unit*. New generated features come from the statistical count of the *Problem Unit* total number, Hints times, Incorrect answers, and Correct answers.

- Problem Section group: It reflects the historical records of the problem from the variable of *Problem Section*. New generated features come from the statistical count of the *Problem Section* total number, Hints times, Incorrect answers and Correct answers.
- Knowledge Components group: It reflects the historical records of the separated knowledge components from the new generated *KC*s. New generated features come from the statistical count of the *KC*s total number, Hints times, Incorrect answers, and Correct answers.

The feature generation processes will be finished with the tools of Microsoft SQL Server 12.0 and IBM SPSS Statistics v22.0. These tools have high performance on data manipulation and more suitable for the other steps of data pre-processing like feature correlation analysis and selection.

5.1.2 Feature Manipulation

This process is done using the tool of IBM SPSS Statistics v22.0. At the beginning of this stage, all the variables connected with the label of '*Time*' will be removed based on the rules of the competition and other applicants' data analytics works [45] [46] [48] [49] [61].

For the categorical variables with a high number of classes including *ProblemName* (819 classes), Automatic Recode will be applied with SPSS to change the categorical variables values from string to numerical type (values between 1 and *n* classes). The reason for this step is to reduce the size of the dataset finally into the learning algorithms and to fit the type requirements of the input variable in some algorithms.

The newly generated variables of KCs are very sparse with *null* value because most of the problems only have 1-2 knowledge components. The highest number of components is 7 which only has 2 cases and the problems with 6 KCs have 48 cases. Thus, these two generated features and the related Opportunity features could be ignored when building the final feature subset. For the cases with features of KCs related Opportunities having the *null* value, a new designed value could be assigned to them with a number significantly more than the highest value in the column. This approach is based on the meaning of the variable Opportunity which represents the number of the encounter of the knowledge component before for this student. The higher the number means more chances the student has worked on the same knowledge component, and more chance he/she can make the correct choice on this problem. The *null* value of Opportunity means there is no knowledge component of this column in this problem. Then the opportunity to make the correct attempt will be even higher. The new value to replace the *null* is designed as:

$$v_{null} = v_{max} + \bar{v} \tag{12}$$

The variable of v_{max} is the maximum of the value in the column, and $\bar{v}$ represents the *mean* of the values of the column.

With the completion of the new features, basic correlation analysis is applied for both the categorical and numerical features toward the dependent variable which indicates the correctness of the problem. The method of correlation test takes the simple Filter way because we evaluate a variety of learning algorithms in the following process and there is no specific one for the data mining process.

5.1.3 Standardization

All the numerical attributes including the newly generated ones are standardized before final output. This work is completed in IBM SPSS Statistics v22.0. The process of standardization for these numerical features is to map each value based on the mean and standard deviation of the features. It generates the 'Z-scores' for all the numerical variables and these newly generated standardized variables can be used in the future data mining algorithms.

5.2 Learning Algorithms

Since the purpose of this thesis is to explore and evaluate the scalability of the distributed system Apache Spark working in the field of education data mining (EDM) with large-scale datasets of cognitive learning systems, only a few most commonly used learning algorithms included in the Spark *MLlib* library are applied to the processed data for training and making prediction. This thesis only involves classification-based algorithms, and the experiments focus on the evaluation of the performance of vectorized categorical features with different mining algorithms. Before loading the dataset into the learning algorithms, all the re-coded categorical variables are converted to sparse vector *LabeledPoints* to fit in these *DataFrame* based algorithms.

- **Logistic Regression**: There are two LR models *in spark.ml* for both binomial logistic regression and multinomial logistic regression. In this thesis, the prediction output is binary class. Therefore, the binomial model is applied. The implemented L-BFGS algorithm is chosen for training and making the prediction. The tuning parameters selected are *regParam* and *maxIter* with different assigned numbers.

- **Decision Trees**: The *spark.ml* implementation supports decision trees for binary and multiclass classification and for regression, using both continuous and categorical features. The new implementation also working with *DataFrame* and *ML Pipelines* which offer more functionality than the original method. In this thesis, the *DecisionTreeClassifier* function is chosen as the classifier for the given dataset.
- **Support Vector Machine:** *LinearSVC* in *spark.ml* supports binary classification with linear SVM. It is also trained with both L2 and L1 regularization. The tuning parameters with SVM classification are *regParam* and *maxIter*.
- **Random Forest:** The *spark.ml* implementation supports random forest for binary and multiclass classification as well as for regression, and the features can be both continuous and categorical features. The implementation uses the existing decision tree implementation with the function name *RandomForestClassifier*.
- **Evaluation and Validation:** 10-folded cross-validation is applied for some of the learning algorithms in the tuning part of the programs, and different algorithms are involved different parameters tuning that fit for the specific algorithms.

5.3 Implementation with Python on Apache Spark

Apache Spark supports various API like **Java, Python, Scala** and **R**. Before Spark 2.0, the main programming interface of Spark was the Resilient Distributed Dataset (RDD). After Spark 2.0, RDDs are replaced by Dataset, which is strongly-typed like an RDD, but with richer optimizations under the hood. The Dataset API is one of the ways to interact with the module of Spark SQL which provides Spark with more information about the structure of both the data and the computation performed. *DataFrame* is a Dataset organized into name columns. It is conceptually like the 'table' in a relational database. *DataFrames* can be constructed from a wide array of sources including structured data files, tables in Hive, external databases, or the former version RDDs. In this thesis, only the learning methods in *spark.ml* are applied and all of them prefer *DataFrames* as the interaction API.

Python-based API named Pyspark are used. Two versions of Spark including Spark 2.0.0 and Spark 2.2.0 are the platforms to run the Python programs. The Spark 2.0.0 is based on the York University Hadoop +Spark cluster with the resource manager as YARN. The Spark 2.2.0 is the Google Cloud Platform of Cloud Dataproc, which has the built-in Hadoop 2.7 and Spark 2.2.0 for big data analytical tasks. Both versions of Spark work with Python 2.6+ or Python 3.4+. It can use the standard CPython interpreter. Thus, C libraries like NumPy can be used. PySpark shell is responsible for linking the Python API to the Spark core, and it provides the users with a variety of great tools for machine learning. The reasons for implementation with Python instead of Scala are that Python is a highly productive language and the learning curve is much shorter with its full package of libraries and tools for data science.

For the reason of different Spark versions in different environments, both RDD based and *DataFrame* based PySpark libraries are involved in programing the data mining algorithms in Python.

York University Hadoop + Spark Cluster: Spark 2.0.0

Logistic Regression	Tuning with 10-folded Cross-validation

Google Cloud Dataproc Hadoop +Spark Cluster: Spark 2.2.0

Logistic Regression	Tuning with 10-folded Cross-validation
Decision Tree Classification	No tuning
Random Forest Classification	No tuning
Linear Support Vector Machine	Tuning with 10-folded Cross-validation

5.4 Scalability and Performance Evaluation Design

In this thesis, two infrastructures of Apache Hadoop +Spark cluster platforms are used to evaluate the scalabilities of data mining algorithms on distributed computing systems. One cluster is from York University with 19 nodes for research purpose. The other cluster is on Google Cloud Dataproc with the dynamic choice for the node number with Google core infrastructure for data analytics and machine learning.

5.4.1 The Infrastructure of York University Spark Cluster

The architecture of this Spark system is a Hadoop YARN managed Cluster with one master node, one secondary name node and 17 worker nodes, which is shown in Figure 5:

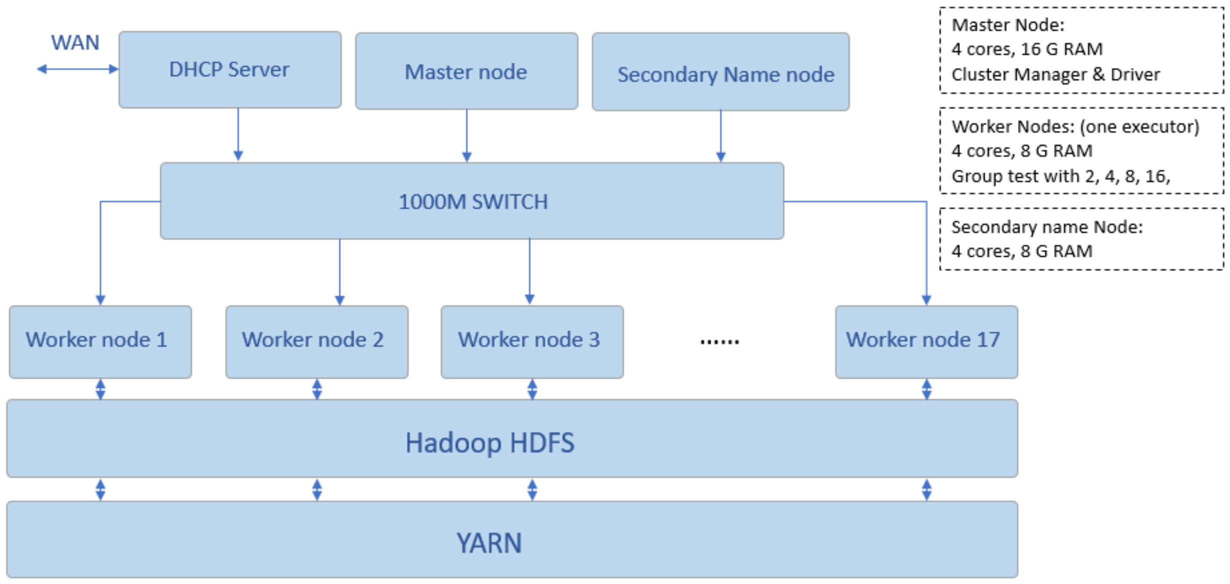


Figure 5 The York University Spark Cluster Architecture

The setting of the nodes in the cluster are:

Master node:	i7-3770 CPU @ 3.40GHz	16G RAM
Secondary name node:	i7-3770 CPU @ 3.40GHz	8G RAM
Worker nodes:	i7-3770 CPU @ 3.40GHz	8G RAM
DHCP server:	i7-3770 CPU @ 3.40GHz	32G RAM

The DHCP server is working as the function of network router with a designed firewall for remote cluster accessing and monitor. The Master node and Secondary name node will not take the task of computing.

Considering the overhead memory for each instance and the executor when Spark node makes computing, 1G memory with 2 VCores for each container is set for the scalability evaluation of this distributed computing system with the limitation of 7G RAM on each worker node. The spark running instance properties configuration will be set as:

spark.executor.memory	1G

spark.network.timeout	800000ms
spark.executor.num	4, 8, 16, 32, 64

Spark resource manager will dynamically assign a designed number of partitions to different nodes.

Dataset is stored in the Hadoop Distributed File System (HDFS).

The accessing and manipulations of this cluster are on remote terminal with SSH through Port 22.

The monitor of the cluster and data nodes are remotely on terminal web browser trough Port 8088 and 50070.

5.4.2 The Infrastructure of Google Cloud Dataproc Spark Cluster

Google Cloud Dataproc is a fast, easy-to-use and fully managed cloud service for data analytical works with Apache Hadoop and Spark. For this experiment a Hadoop 2.7 + Spark 2.2.0 YARN-based cluster is built, which is shown in Figure 6:

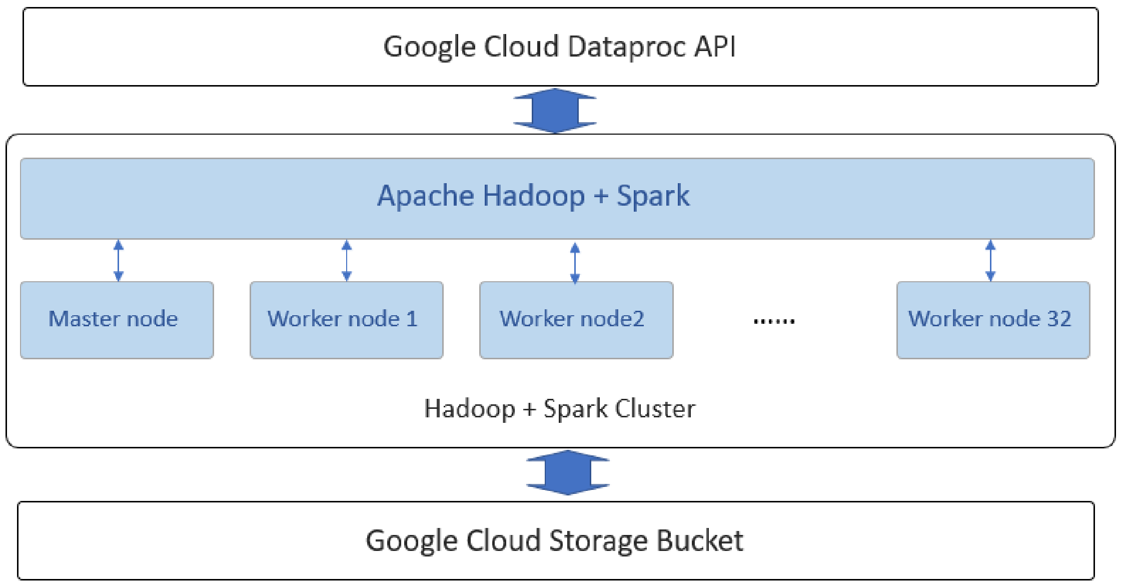


Figure 6 The Google Cloud Spark Cluster Architecture

The setting of the nodes in the cluster are:

Master node:	16G RAM
Worker nodes:	16G RAM

The dataset is stored in the Google Cloud Storage Bucket, which has several advantages over HDFS like direct-data-access, HDFS compatibility, interoperability between Hadoop and Spark, better accessibility, higher availability, no storage management overhead and easy set-up.

The Google Cloud Dataproc API takes the mission of monitoring the jobs, resizing the cluster and managing the cluster. The Pyspark computing jobs are submitted by the tools of Google Cloud SDK.

Considering the Overhead memory consumption for each Spark instance, the spark running instance properties configuration is set as:

spark.executor.memory	7G / 10G (re-partition experiments)
spark.network.timeout	800000ms
spark.executor.num	4, 8, 16, 32, 64 (2,4,8,16,32 for re-partition experiments)

Chapter 6. Results and Discussion

In this part, we use the Python API for Spark to evaluate the scalability of Spark with different data learning methods on the generated datasets from the Online Cognitive Learning Systems. The learning algorithms applied in this thesis are based on the package of *spark.ml* and *spark.mlib* on both versions of Apache Spark 2.2.0 and Spark 2.0.0, which support the new interfaces of Dataset as well as the former Resilient Distributed Dataset (RDD). The learning algorithms used in this thesis are Logistic Regression, Random Forests, Decision Tree and Support Vector Machine with the selected functions in *spark.ml* and *spark.mlib*.

In this thesis, the categorical features most related to the problem are chosen to evaluate the scalability of the distributed system of Spark on different algorithms. At the same time, all the numerical features are still pre-processed and grouped for the future work. All the selected features are transferred, vectorized and assembled to sparse vector labeled point format to fit the data mining algorithms.

This thesis is focused on the scalability and running time performance of Apache Spark. Hence, all the learning Pyspark programs are run on a different number of worker nodes with designed resource allocation to evaluate the relationship between computing resources and running time performance for different algorithms applied on Spark.

6.1 Datasets Overview

The dataset in this thesis comes from the KDD Cup 2010 Educational data mining Challenge. All the datasets are students' logs of two Mathematics Online Tutoring Systems called the *Carnegie Learning Algebra* system deployed as *Algebra I 2005-2006 and 2006-2007*, and the *Bridge to Algebra system* deployed *2006-2007*.

Development Datasets

• Algebra I 2005-2006	575 students	813,661 cases
• Algebra I 2006-2007	1,840 students	2,289,726 cases
• Bridge to Algebra 2006-2007	1,146 students	3,656,871 cases

Challenge Datasets

• Algebra I 2008-2009	3,310 students	9,426,966 cases
• Bridge to Algebra 2008-2009	6,043 students	20,768,884 cases

All the datasets have the similar feature structure, with 20 features in *Algebra I* and 19 features in *Bridge to Algebra*. Considering the massive size and similar feature structure of the datasets, only the *Algebra I 2005-2006* and *Algebra I 2006-2007* are selected as the working data sets. Challenges of the dataset are provided as below:

- Vast number of instances with high feature dimensions. (813,661 cases, 20 features)
- Sparse data.
- Highly categorical variables.

The original *Algebra I 2005-2006* and *Algebra I 2006-2007* data sets come with 20 features:

- *Row* - The row number of the case
- *Anon Student Id* – Unique student ID
- *Problem Name* - Identifier of the problem that indicates a task for student to perform and normally contains multiple steps.
- *Problem View* - Total number of time the student encountered this problem
- *Step Name* - Unique identifier for the step which is an observable part of the solution to a problem.
- *Step Start Time* - Start time of the step
- *First Transaction Time* - First transaction time of the step
- *Correct Transaction Time* - The time of correct attempt of the step
- *Step End Time* - The time of last transaction of the step
- *Step Duration (sec)* - The time to finish this step
- *Correct Step Duration (sec)* - The step duration if the first transaction is correct for this step
- *Error Step Duration (sec)* - The step duration if the first try of this step is not correct
- *Correct First Attempt* - The evaluation of first try of the step, 1-correct, 0-error
- *Incorrects* - Total number of incorrect attempts for this step
- *Hints* - Total number of Hints the student used for this step
- *Corrects* - Total correct attempts this student done for this step
- *KC(Default)* - The skills involved in this problem. Each step can have more than one *KC*s separated with '~~'
- *Opportunity(Default)* - The number of times the student encountered the *KC*s before. Multiple Opportunities are separated by '~~'

- *Proble_Unit* - The classification of the curriculum
- *Proble_Section* - The portion of the Unit

The performance is evaluated by the predicting accuracy of the binary class *Correct First Attempt*, which represents whether the student fully masters the knowledge components of the problem and makes the right choice at first try.

6.2 Data Pre-processing

Before the evaluation of the scalability of Spark on Education data mining, the original dataset is pre-processed to fit in different algorithms of data mining Pyspark programs. This process is mainly based on former KDD challenge winners' papers [45][46][47][48], which have made feature correlation evaluations from different methods and algorithms.

6.2.1 Feature Generation

In this stage, all the manipulation and programming are completed with Microsoft SQL Server 2012 and IBM SPSS Statistics 22.

The variables of *KC* and *Opportunity* need to be broken into different components they embedded. The *KC* indicates the skills that are used in the problem and a problem step may contain multiple *KC*s. These *KC*s are separated by '~~ '(two tildes). The maximum number of *KCs* in one problem step is 7 and all the *KC*s will be separated into 7 different tiers as new generated features of *KC1-KC7*. The variable of *Opportunity* is a chance for a student to demonstrate whether he or she has learned a given knowledge component. A student's *opportunity* value for a given knowledge component increases by 1 each time the student encounters a step that requires this knowledge component. According to the knowledge components (represented as *KC*s) in one step of problem, the variable of *Opportunity* contains the times the student encountered these knowledge components before, which are separated by '~~ '(two tildes) too. The process of generating these new features is shown in Figure 7:

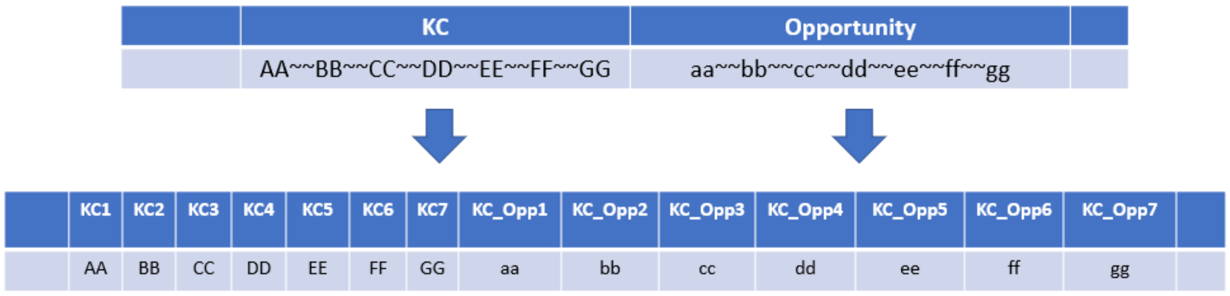


Figure 7 Feature break up

The features of the original dataset can be classified into 4 groups:

- Useless for data mining process: *Row*, *Anon Student Id*
- Time-related: *Step Start Time, First Transaction Time, Correct Transaction Time, Step End Time, Step Duration (sec), Correct Step Duration (sec), Error Step Duration (sec)*
- Problem & Step related: *Problem Name, Problem View, Step Name, Incorrects, Hints, Corrects, Correct First Attempt, Proble_Unit, Proble_Section*
- Knowledge Component related: *KC(Default), Opportunity(Default)*

The new generated features will come from the Problem & Step related group and Knowledge Component related group of features. The tools used for this step are MS SQL Server and IBM SPSS Statistics 22.0 with the new group of generated features list below:

- *KCnumber*: to sum up the total KCs (knowledge components) in each question.
- *ProblemNameNominal, StepNameNominal, Proble_UnitNominal, Proble_SectionNominal*: These four categorical features is nominalized into integers with SPSS. The new generated features are applied with the One-Hot process in Pyspark.
- *Row_T, Hint_T, Incorrects_T, Corrects_T*: the numerical variables calculated to represent for each student's historical records of total questions answered, number of hints applied, total times of incorrect answers and total correct choices.
- *Row_P, Hit_P, Incorrects_P, Corrects_P:* the numerical variables generated to describe for this question's Problem attribute history records of total questions answered, number of hints applied, total times of incorrect answers and total correct choices.
- *Row_U, Hit_U, Incorrects_U, Corrects_U :* the numerical variables generated to describe for this question's Unit attribute history records of total questions answered, number of hints applied, total times of incorrect answers and total correct choices.
- *Row_S, Hit_S, Incorrects_S, Corrects_S:* the numerical variables generated to describe for this question's Section attribute history records of total questions answered, number of hints applied, total times of incorrect answers and total correct choices.
- *KC#_row, KC#_hint, KC#_incorrects, KC#_corrects:* (# is from 1~6) the numerical variables generated to describe each KC history records of total questions answered, number of hints applied, total times of incorrect answers and total correct choices.

6.2.2 Feature Manipulation and Standardization

All the time related features such as *Step Start Time, First Transaction Time, Correct Transaction Time, Step End Time, Step Duration (sec), Correct Step Duration (sec), Error Step Duration (sec)* are dropped due to the missing values in test datasets [45] [46] [47] [48]. The new generated features are made correlation test with the dependent variable in SPSS. Since the main purpose of this thesis is to evaluate the scalability of distributed computing system Apache Spark on the educational datasets, all the new generated features are standardized with SPSS and go into the data mining process in Spark.

To lower the volume of the datasets, the categorical features are applied with Automatic Recode with the tool of SPSS to change the *String* values to nominal numbers. This procedure assigns each unique category with a number code and saves the converted values into a new variable. All the new generated categorical features are called recoded categorical features.
All the numerical features including the newly generated ones are standardized. This work is completed in IBM SPSS Statistics. The process generates the 'Z-scores' for all the numerical variables and these new generated standardized variables start with character 'Z' plus the old feature names.

The selected features will be classified into two groups:

- Recoded categorical features, mainly used to compare the scalability of Spark with different resources allocated.
- Standardized numerical features: for future work.

6.3 Learning Algorithms Experiments on Spark Clusters

Two Apache Spark platforms are used to execute the data mining programs. For different platform infrastructures and software version, different Python data mining programs are implemented as well as the job submission strategies and settings.

6.3.1 Classification Results with York University Lab Spark Cluster

Resource planning (executors, memory, cores etc.) plays a vital role when running Spark application on Hadoop YARN. The *memory Overhead* is one of the parameters that need to consider when designing the experiment strategy and tuning the configuration of Spark. Due to the limitation of the 7G RAM for each worker node, the fixed number 17 of worker nodes and the version of Spark 2.0.0, the classification algorithm of **Logistic Regression** with 10-folded cross-validation model is programmed with different resource allocated strategies. One parameter of *maxIte*r is tuned with choice of [1,2,4] in the tuning process. The dataset is stored in the Hadoop HDFS distributed file system. Only the nominalized categorical features are transferred to vectors with a One-Hot process and fitted into the model. The dataset is split into 70% training data and 30% test data. Two experiments with different strategies are executed on this cluster.

Experiment I: The Spark mining resource allocation strategy is designed as 1G RAM for each instance with dynamic executor selection by Spark and the number of instance selection is set as

4, 8, 16, 20, 32 and 64. The resource allocation of some main parameters of Spark case is shown below:

Spark.driver.memory	1G
Spark.executor.memory	1G
Spark.yarn.executor.memoryOverhead	384MB

The running results for different numbers of cluster usage and the speedup are shown in Table 2 and Figure 8. The result shows when the number of executor increases from 4 to 8, the running speed gets significant 35.71% speedup ratio. From 8 to 16 instances the running speed increases by 16.39%, while more nodes added after this the running speed does not get significant improvement.

Logistic Regression	Number of Nodes	Running Time	Speedup Ratio
1G / instance	4	5774	
	8	3712	35.71%
	16	2766	52.10%
	20	2821	51.14%
	32	2743	52.49%
	64	2797	51.56%

Table 2 Results of Logistic Regression experiment I on York U Spark Cluster

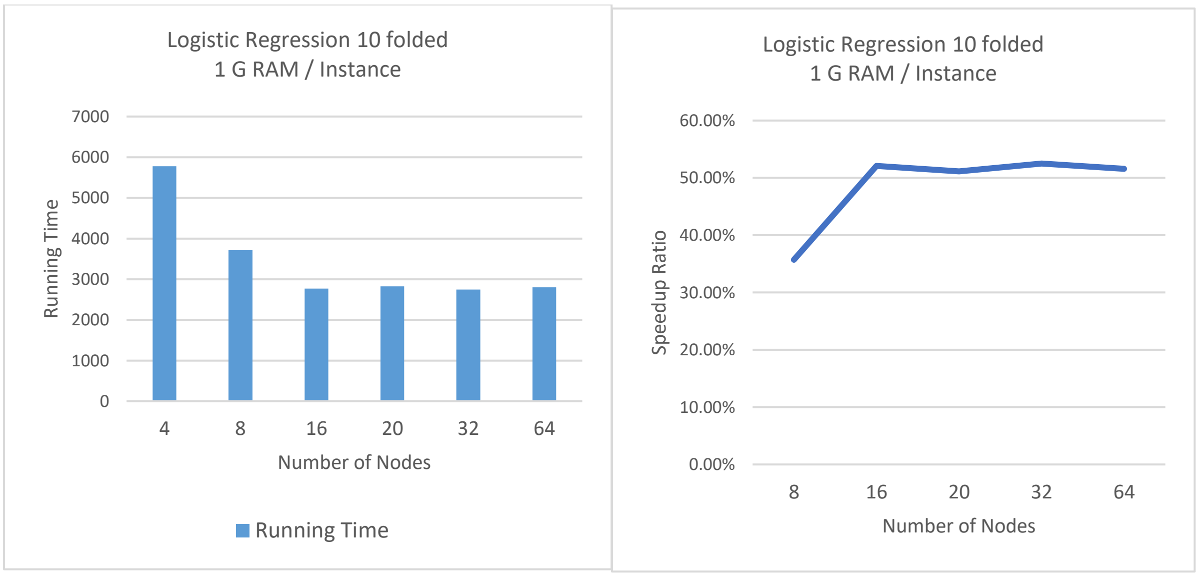


Figure 8 Results of Logistic Regression experiment I on York U Spark Cluster

Experiment II: The Spark mining resource allocation strategy is designed as 4G RAM for each instance with dynamic executor selection by Spark and the number of instance selection is set as 1, 2, 4, 8, and 16. The resource allocation of some main parameters of Spark case is shown below:

Spark.driver.memory	1G
Spark.executor.memory	4G
Spark.yarn.executor.memoryOverhead	spark.executor.memory * 0.1 =400MB

The running results for a different number of cluster usage and the speedup for this experiment are shown in Table 3 and Figure 9. Significant speedup ratio could be observed from the results when more Spark worker nodes involved from one node to two (36.41%), two nodes to four (62.26%) and four nodes to eight (76.51%), while just the same as Experiment I results showed, when the number of worker nodes increase from 8 to 16, the running speed only increases to

82.55% and from the curve we can see the increasing ratio of the speedup becomes steady. Because of the limit of total 119 G RAM (17 x 7 G) of this Hadoop cluster, for 4 G RAM / instance experiment cannot extend the worker number to 32.

LR	Number of Nodes	Running Time	Speedup Ratio
4G/instance	1	4751	
	2	3021	36.41%
	4	1793	62.26%
	8	1116	76.51%
	16	829	82.55%

Table 3 Results of Logistic Regression experiment II on York University Spark Cluster

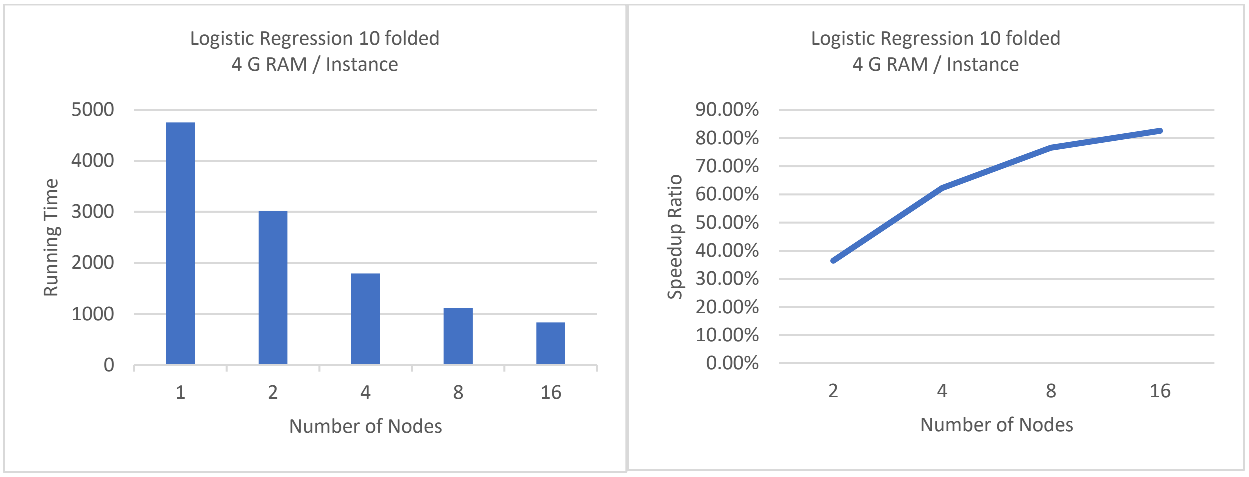


Figure 9 Results of Logistic Regression experiment II on York University Spark Cluster

Based on **Experiment I** and **Experiment II**, we can also notice that with Spark distributed system resource allocation strategy, the running speed for Logistic Regression algorithm with this dataset can be increased significantly with more memory for each executor instance other than more nodes added with the dataset at this size. Hadoop works with the principle of data locality and pushing computation to data. Hadoop stores the data in HDFS. The data are split into blocks and distributed on different DataNodes. Hadoop MapReduce schedules the computation tasks to executors, which are normally on various DataNodes. The computation tasks will be executed on the nodes where the data blocks stored or the nodes on the same rack. This minimizes the data transfer latency and significantly improve the performance. When Spark loads the data, by default the partition size for the dataset is set to 128M. The volume of this dataset is 840M, which is divided into seven partitions by default. Each executor on the DataNode can only work on one partition at one time. More executors than seven will not significantly improve the parallelism computing time on each executor when there is no *repartition()* progress to partition data into more blocks and distribute them on more nodes. While Spark works with the unique strategy of in-memory computing, which relies on the capacity of executor's memory for iterative jobs. More memory on each executor will significantly improve the computing speed, which takes the dominant part of running time in this case. Table 4 and Figure 10 show the comparison of the memory of 1G /instance with 4G/instance strategies with the same amount of total memory working on the program. Compared to 1G RAM per executor when running the LR prediction, the 4G RAM strategy shows significant advantage when total memory is over 16 G. For 1G / instance the number of worker nodes reaches 16, while for 4G/instance case the number of worker nodes is only 4, which still has the potential to raise the performance with more nodes involved. We can reach the point that the scalability of distributed system Spark has some strategies with the resources allocation design on YARN-based system. For this dataset, when limited executors are involved, the time of computing dominates the running time, while the costs of time for the processes of '*map-shuffle-reduce*' does not take significant parts. When executors are more than 8, compared to the computing time, the '*map-shuffle-reduce*' processes will have obvious negative effects to the whole running time performance. On the other hand, the increasing of the memory for each executor could significantly reduce the computing time. We can conclude that the most optimized solution for the classification of this dataset on Spark can be increasing the memory on each executor and setting 'number of executors' to 8-16.

LR/10 folded	Memory	1G / Instance Running Time	4G / Instance Running Time
Total Memory	4	5774	4751
	8	3712	3021
	16	2766	1793
	32	2743	1116
	64	2797	829

Table 4 Comparison of memory strategies on York U Spark Cluster

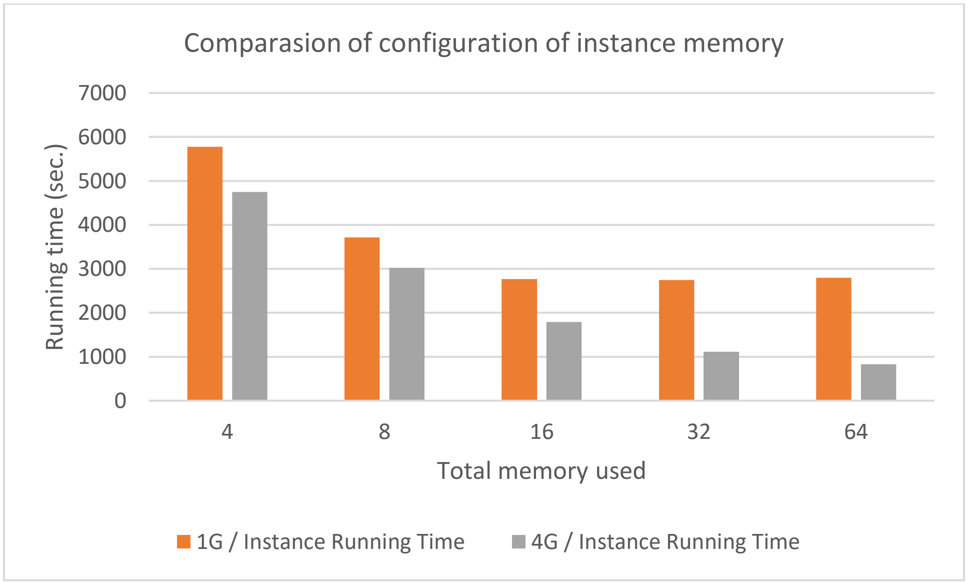


Figure 10 Comparison of memory strategies on York U Spark Cluster

6.3.2 Classification Models for Google Cloud Dataproc Spark Cluster

The technology of Cloud computing service makes it possible to build the Hadoop distributed system with infinite resource and minimal management effort. This also gives us the best way to explore the scalability of data mining methods on educational dataset especially for the online real-time analytic with data mining methods. This part of the thesis is based on Google Cloud Dataproc, which is an easy-to-use and fully managed cloud service for building and running Hadoop and Spark cluster. To avoid the network latency between nodes, all the worker nodes in the cluster are selected in the location zone of 'usa-east-1b'. The master node of the Spark cluster is 4 Cores Intel CPU, 15G memory and 20G hard drive. All the worker nodes in the Spark cluster are set as 4 Cores Intel CPU, 15 G memory and 10G hard drive. Four data mining algorithms are programmed and applied to the dataset with different strategies and resources to test the scalability of Spark on Google Cloud. To avoid the possible abnormal noise with the results, each submission of the Pyspark task is conducted three times. The average of the three results is used as the final output.

Logistic Regression with 10-folded Cross-validation: The experiment is designed with 7G RAM/instance and the worker number is set to 1, 2,4, 8, 16 and 32. The tuning parameter in the Spark Pipeline falls on the *maxIter* with choice of [1,2,4]. The results of running speed with different worker nodes are shown in Table 5 and Figure 11. From the results, we can see the performance improves significantly from 1 worker node to 2 nodes with speedup ratio to 39.45%, and with the increasing of the node number to 4 the speedup ratio still gets a high increase with 60.96%. After reaching to the number of 8 worker nodes the performance increases to 66.80%

while the curve becomes smooth, and with more nodes added there is no significant improvement to the running time.

LR (10 folded)	Num of Nodes	Running Time(Sec.)	Speedup Ratio
7G / instance	1	4063	
	2	2460	39.45%
	4	1586	60.96%
	8	1349	66.80%
	16	1371	66.26%
	32	1380	66.03%

Table 5 Results of Logistic Regression(10-folded) on Cloud Spark Cluster

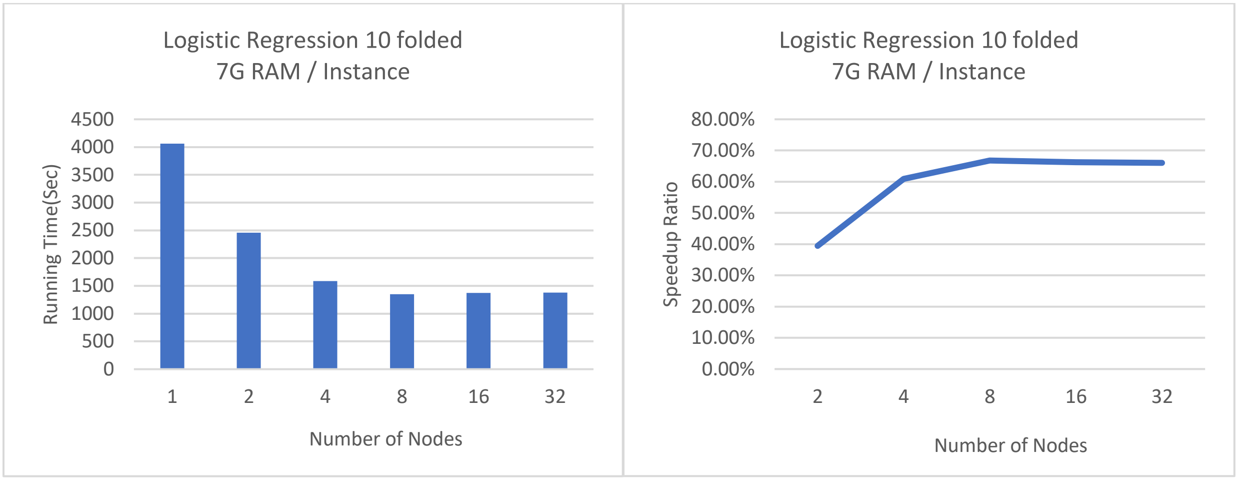


Figure 11 Results of Logistic Regression(10-folded) on Cloud Spark Cluster

Support Vector Machine (SVM) with 10-folded Cross-validation: The experiment is designed with 7G RAM/ instance and the worker number is set to 1, 2,4, 8, 16 and 32. The results of running speed with different worker nodes are shown in Table 6 and Figure 12. We can see the performance of SVM on Spark improves significantly when worker nodes added from 1 to 2 (20.54%) as well as from 2 to 4 (34.72%), but after that when more nodes are involved in the distributed computing process there is an only minor improvement on the performance of running speed.

SVM(10 folded)	Num of Nodes	Running Time(Sec.)	Speedup Ratio
7G / instance	1	2497	
	2	1984	20.54%
	4	1630	34.72%
	8	1624	34.96%
	16	1593	36.20%
	32	1653	33.80%

Table 6 Results of SVM (10-folded) on Cloud Spark Cluster

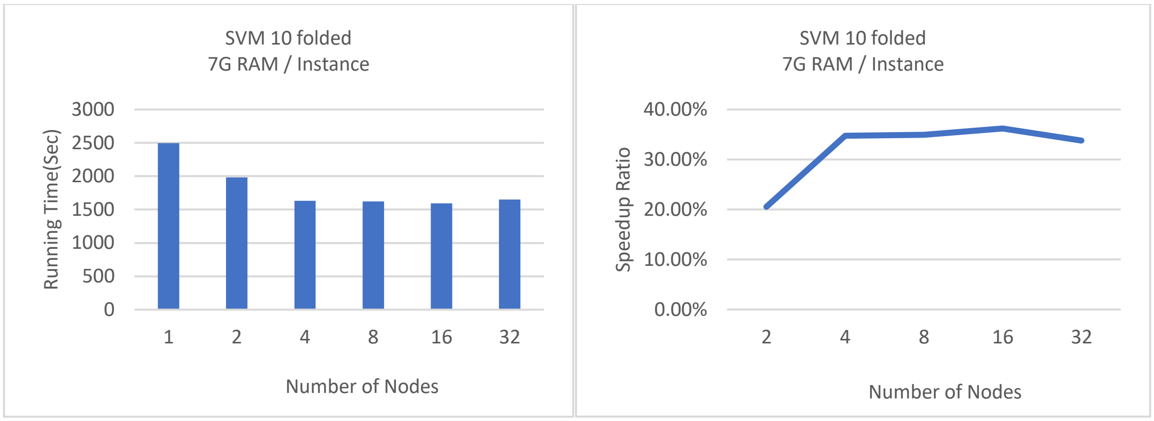


Figure 12 Results of SVM (10-folded) on Cloud Spark Cluster

Random Forest Classification (non-folded): The experiment is designed with 7G RAM/instance and the worker number is set to 1, 2,4, 8, 16 and 32. When working with 10-folded cross-validation, the running time for Random Forest is extremely long. Due to the purpose of the thesis is to explore the scalability of Distribute System of Spark, for Random Forest and Decision Tree classification there is no cross-validation during the mining process. The results of running time for Random Forest classification are shown in Table 7 and Figure 13.

We can see when the number of worker nodes increased from 2 to 4, the running speed performance improves significantly to 66.61%. When the number of nodes grows to 8, the improvement is 72.02% and the curve for speedup ratio becomes smooth. When the number of worker nodes is more than 8, the performance has no notable change.

RF	Num of Nodes	Running Time(Sec.)	Speedup Ratio
7G / instance	1		
	2	4911	
	4	1640	66.61%
	8	1393	71.64%
	16	1374	72.02%
	32	1405	71.39%

Table 7 Results of Random Forest (non-folded) on Cloud Spark Cluster

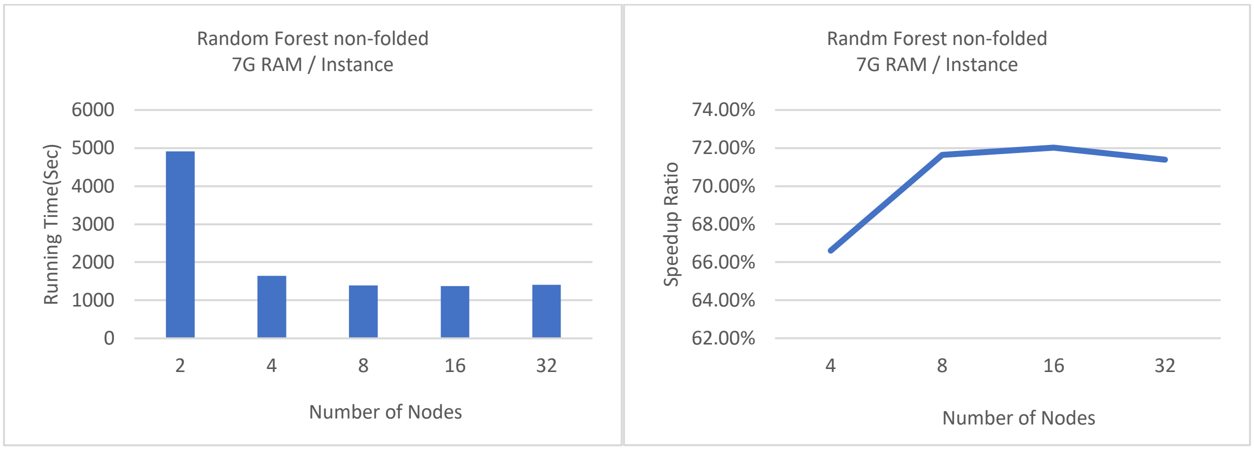


Figure 13 Results of Random Forest (non-folded) on Cloud Spark Cluster

Decision Tree Classification DT (non-folded): The experiment is designed with 7G RAM/instance and the worker number is set to 1, 2,4, 8, 16 and 32. The results of running time with a different number of worker nodes are shown in Table 8 and Figure 14. Just like Random

Forest, the Decision Tree Classification has significant speedup rate of performance when the number of worker nodes increases from 2 to 4 (44.43%), and minor improvement from 4 to 8 (49.60%). It also has the bottleneck when the number of worker nodes is more than 8 for the improvement of running time.

DT	Num of Nodes	Running Time(Sec.)	Speedup Ratio
7G / instance	1		
	2	2748	
	4	1527	44.43%
	8	1385	49.60%
	16	1394	49.27%
	32	1397	49.16%

Table 8 Results of Decision Tree (non-folded) on Cloud Spark Cluster

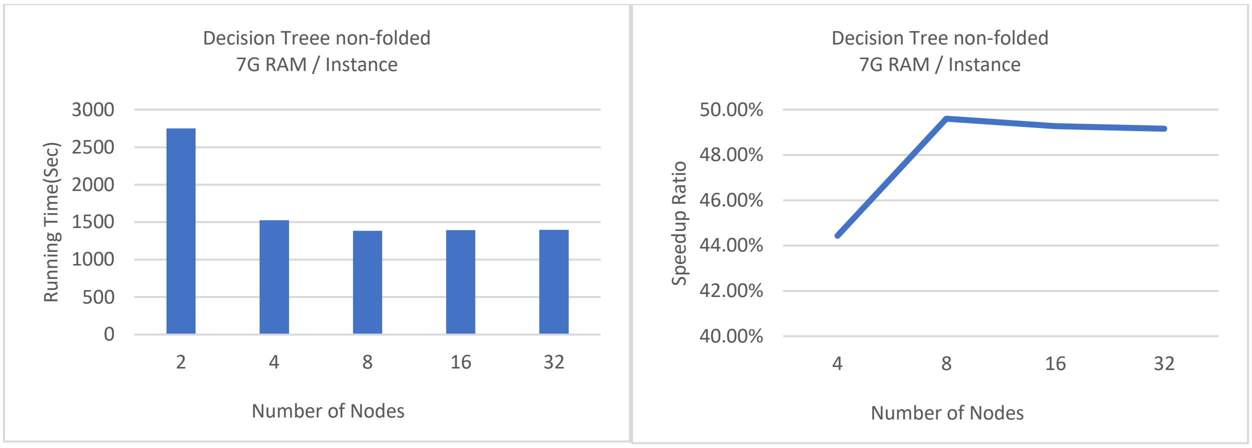


Figure 14 Results of Decision Tree (non-folded) on Cloud Spark Cluster

Support Vector Machine SVM (non-folded): The experiment is designed with 7G RAM/instance and the worker number is set to 1, 2,4, 8, 16 and 32. For the prediction of student's performance with this educational dataset, SVM and Logistic Regression have much higher running speed and accuracy than Decision Tree and Random Forest with the vectored categorical feature group. The results of running speed of different numbers of worker node are shown in Table 9 and Figure 15. As most of the experiments we made in this thesis, it has the same curve of performance increased significantly from 2 nodes to 4 nodes and from 4 nodes to 8, but beyond 8 nodes the performance will drop or become smooth because of the mapping process to distribute the RDD to all executors.

SVM	Num of Nodes	Running Time(Sec.)	Speedup Ratio
7G / instance	1		
	2	152	
	4	115	24.34%
	8	100	34.21%
	16	109	28.29%
	32	116	23.68%

Table 9 Results of SVM (non-folded) on Cloud Spark Cluster

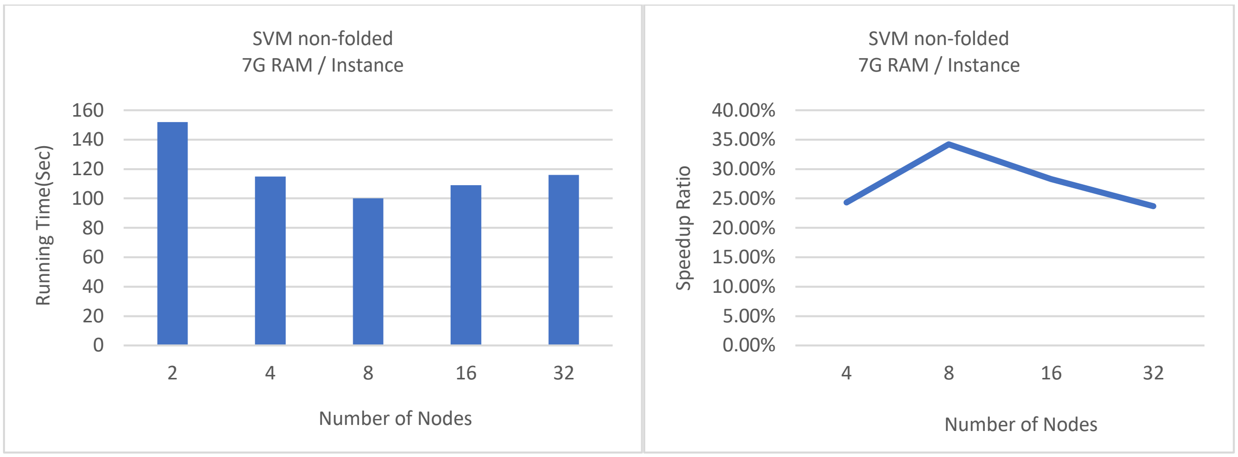


Figure 15 Results of SVM (non-folded) on Cloud Spark Cluster

Logistic Regression LR (non-folded): The experiment is designed with 7G RAM/instance and the worker number is set to 1, 2,4, 8, 16 and 32. The results of running speed with a different number of worker nodes are shown in Table 10 and Figure 16.

LR	Num of Nodes	Running Time(Sec.)	Speedup Ratio
7G / instance	1		
	2	114	
	4	106	7.02%
	8	89	21.93%
	16	83	27.19%
	32	90	21.05%

Table 10 Results of LR (non-folded) on Cloud Spark Cluster

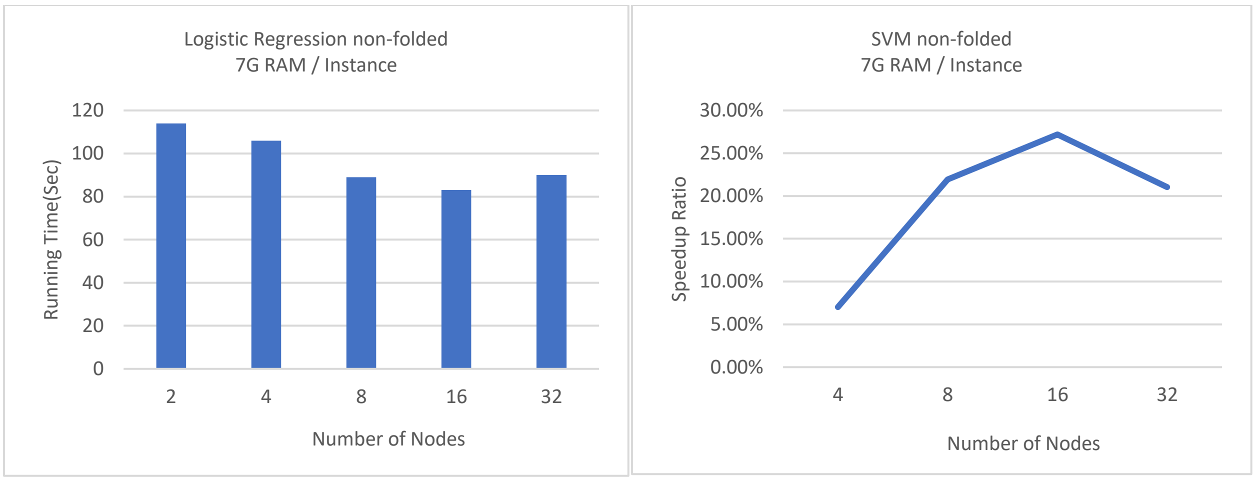


Figure 16 Results of LR (non-folded) on Cloud Spark Cluster

From the experiments on Google Cloud Spark cluster with this dataset, we can reach the point that the optimized resource allocation strategy could be around 8 to 16 executor nodes for Logistic Regression, Decision Tree, Random Forest and SVM. There is no process of 're-partition' and the block division for the dataset. This process is automatically done by Google

File System (GFS). The dataset volume size is around 830M, and by default Google File System chunk size is 64M. Based on the storage of dataset on GFS, Spark will automatically set the partition number to 13 for this dataset. Map tasks in MapReduce normally operate one block at a time. The mappers will map the jobs to these nodes and the computing time will be shortened when more executors involved. On the other hand, when the number of chunks is less than the number of nodes, the performance has no improvement when more nodes are added. We can conclude that the scalability of Spark for big data can be designed with the consideration of resource allocation based on the size of dataset besides the learning algorithms.

6.3.3 Experiments on Datasets with Varied Volumes

To explore the scalability and the bottleneck of Spark with classification algorithms on varied volumes of dataset, a new dataset of *Algebra 2006-2007* from the KDD competition with the size of 2 million cases are pre-processed and split into a half million cases dataset and a one million cases dataset. The details of the new datasets are listed as:

Whole2006_2007Trimed.csv	2,270,384 cases	36 features
OneMill2006_2007.csv	1,135,020 cases	23 features
HalfMill2006_2007.csv	567,619 cases	23 features

In this experiment, SVM with 10-folded cross-validation and Logistic Regression with 10-folded cross-validation algorithms are applied to Google Cloud Dataproc Spark cluster. Only the vectored categorical variables of *ProblemName, StepName, ProblemUnit, and ProblemSection* are selected to make the prediction. The number of Spark cluster is designed as 1, 2, 4, 8, 16 and

32. The master node of the Spark cluster is 4 Cores Intel CPU, 15G memory and 20G hard drive. All the worker nodes in the Spark cluster are set as 4 Cores Intel CPU, 15 G memory and 10G hard drive. To reduce the network latency between each node, the cluster is set in the same Google Cloud Zone of USA-east-1b. The configuration *spark.dynamicAllocation.enabled* of Spark running environment is set to '*false*' in order to get full use of all the worker nodes allocated. The results of the experiment are list below:

Num of Nodes	Running Time(Sec.)	Speedup Ratio
1	4039	0.00%
2	3407	15.65%
4	3310	18.05%
8	3238	19.83%
16	3355	16.93%
32	3236	19.88%

Table 11 Results of LR (10-folded) with 2 Million cases dataset

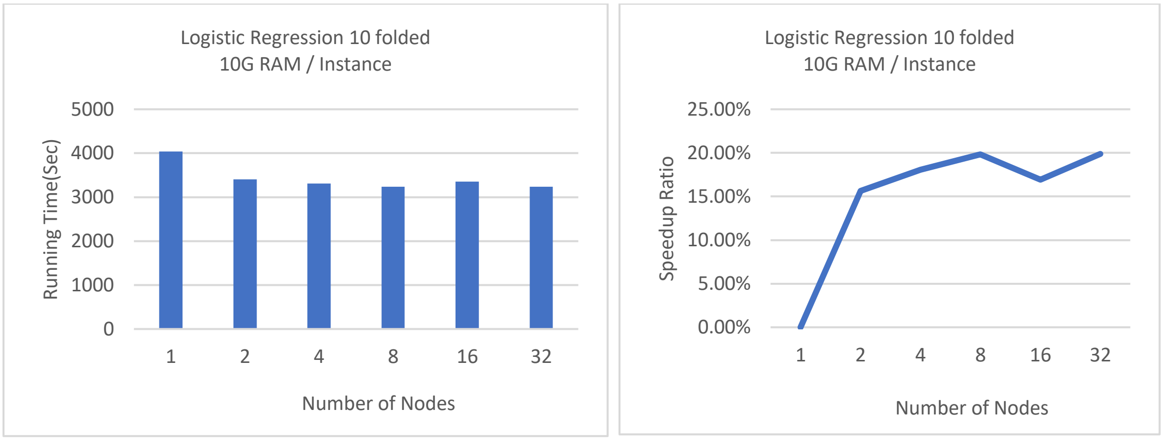


Figure 17 Results of LR (10-folded) with 2 Million cases dataset

Num of Nodes	Running Time(Sec.)	Speedup Ratio
1	3887	0.00%
2	3433	11.68%
4	3425	11.89%
8	3427	11.83%
16	3499	9.98%
32	3543	8.85%

Table 12 Results of SVM (10-folded) with 2 Million cases dataset

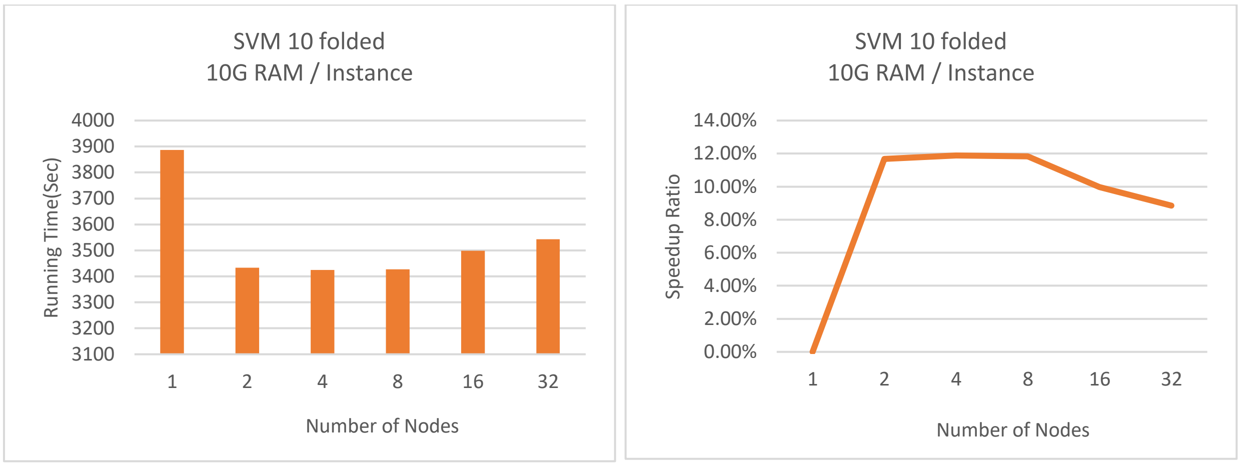


Figure 18 Results of SVM (10-folded) with 2 Million cases dataset

Num of Nodes	Running Time(Sec.)	Speedup Ratio
1	1745	0.00%
2	1569	10.09%
4	1572	9.91%
8	1578	9.57%
16	1575	9.74%
32	1578	9.57%

Table 13 Results of LR (10-folded) with One Million cases dataset

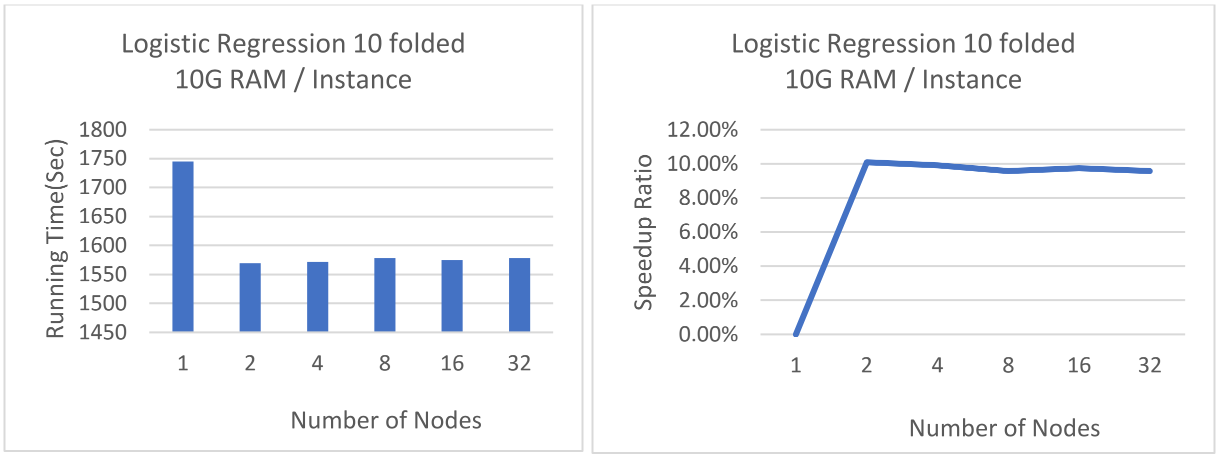


Figure 19 Results of LR (10-folded) with One Million cases dataset

Num of Nodes	Running Time(Sec.)	Speedup Ratio
1	1709	0.00%
2	1528	10.59%
4	1581	7.49%
8	1566	8.37%
16	1544	9.65%
32	1555	9.01%

Table 14 Results of SVM (10-folded) with One Million cases dataset

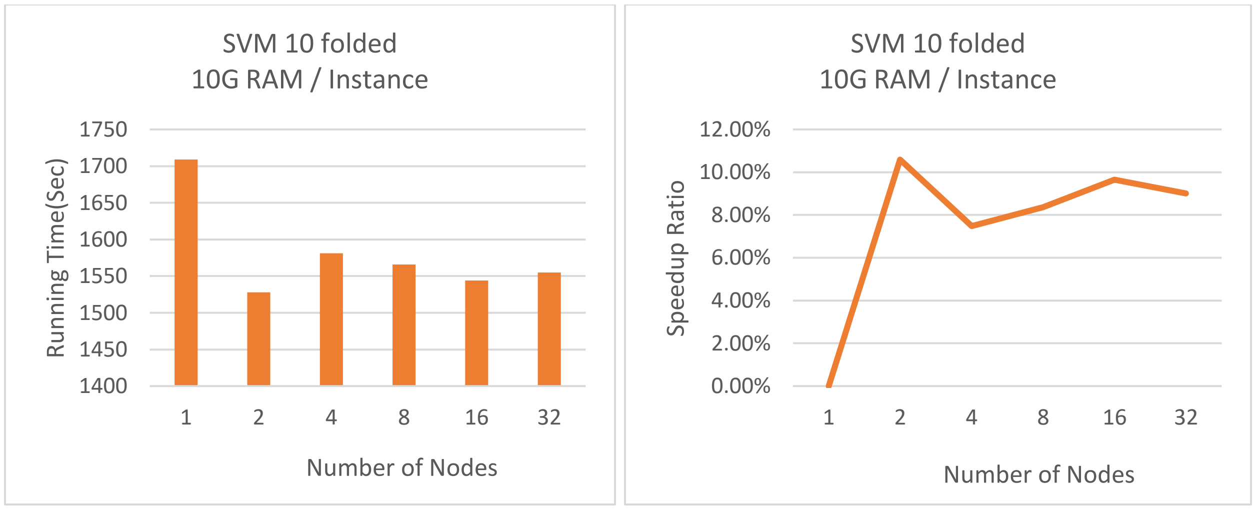


Figure 20 Results of SVM (10-folded) with One Million cases dataset

Num of Nodes	Running Time(Sec.)	Speedup Ratio
1	917	0.00%
2	795	13.30%
4	819	10.69%
8	844	7.96%
16	844	7.96%
32	833	9.16%

Table 15 Results of LR (10-folded) with Half Million cases dataset

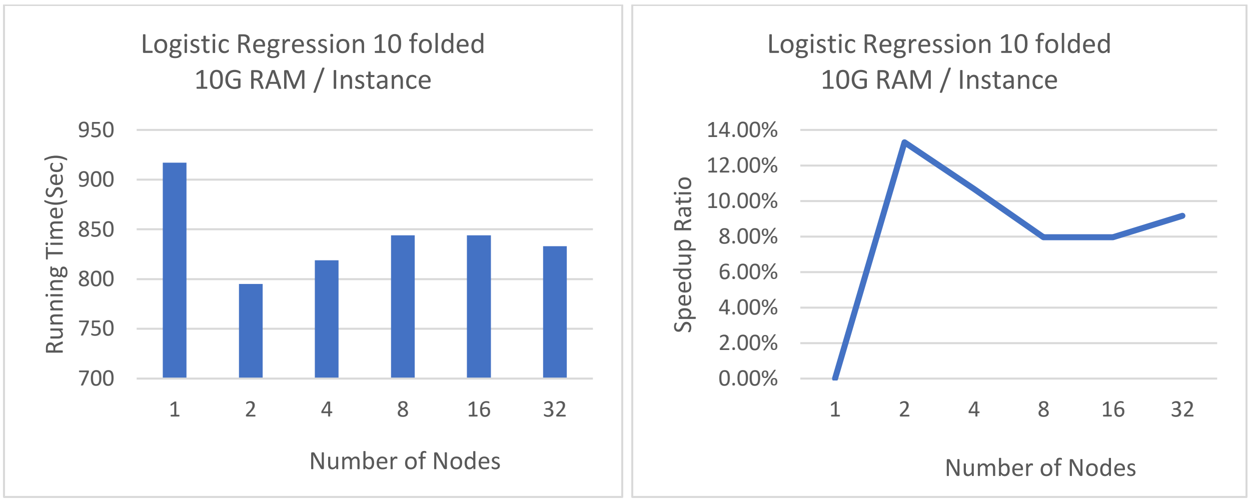


Figure 21 Results of LR (10-folded) with Half Million cases dataset

Num of Nodes	Running Time(Sec.)	Speedup Ratio
1	789	0.00%
2	774	1.90%
4	804	-1.90%
8	819	-3.80%
16	886	-12.29%
32	891	-12.93%

Table 16 Results of SVM (10-folded) with Half Million cases dataset

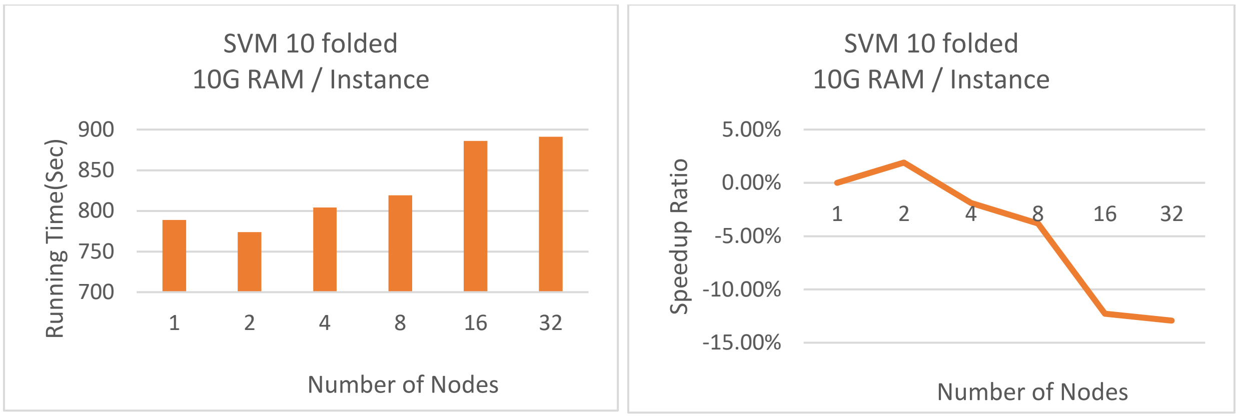


Figure 22 Results of SVM (10-folded) with Half Million cases dataset

From the experiment we can see, for the 2 million cases dataset, Logistic Regression algorithm gets the best performance at the point around 8 worker nodes (speedup ratio 19.83%) and beyond this point, the curve goes smoothly with more nodes added into the cluster. The algorithm of SVM reaches the highest performance around the point of 4 worker nodes (speedup ratio 11.89%) and with the increase of the number of nodes, the speedup ratio drops. The Spark running environment setting of *'spark.default.parallelism'* and *'spark.sql.shuffle.partitions'* are also manipulated during the experiment following the direction of YARN Cluster-Mode.

To avoid the possible abnormal variance with the results on a small volume of datasets, each submission of the Pyspark task is conducted three times. The average running time is calculated based on the three results and will be used as the final output. If there is any WARN notice showed on the Cloud monitor SDK during the computing process, the result will be discarded. The WARN notice normally shows with 'limited resources' when 1-2 nodes allocated and bring 10-15 seconds variance for the result. A new submission will be re-conducted in this case. For the one million cases dataset, there is no setting of parameter *'spark.sql.shuffle.partitions'*, which configures the number of partitions when shuffling data for aggregation. The partition number relies on the default partition strategy of GFS, which will partition the data into the blocks at the size of 64M. The performances for both Logistic Regression and SVM get the peaks around 2 worker nodes. Both the algorithms get the performance steady after the peak point. When more worker nodes added to the cluster, there is no significant improvement for the running time speedup. The reason is that the volume of the one million cases dataset is only 116M. By default, the partition number will be set to 2 with GFS. The Logistic Regression and SVM algorithms take high-speed computation on each node when a small volume of the partitioned block of data involved. Each executor works on one block of data every time. When more executors allocated in the computation task, it will lead to the situation of resource underused. For the concurrent jobs, Spark also offers the dynamic resource allocation as the solution for resources underused or misused. It helps to avoid the situation when the cluster composition doesn't fit the distributed computing workload. In this case, when more than two executors involve in the distributed computing, at the beginning stage all the executors allocated will be called to involve into the task. While based on the Spark solution of dynamic resource allocation, each time the real number of nodes working on the tasks is two. Other executors will be set as idle and no resource will be used. This can be observed and tracked from the Hadoop YARN monitor UI interface. The same issue happens on the experiment on half-million cases datasets. For the half million cases dataset, both the Logistic Regression and SVM get the best performance at the number of 2 worker nodes, while the speedup curves start to drop down when more worker nodes are added into the cluster.

6.3.4 Experiment with Re-partition of One million Cases Dataset

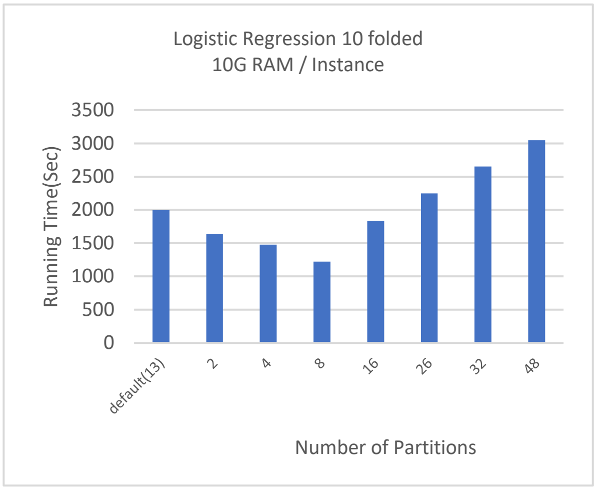
Tree Aggregate at LogisticRegression and *map at BinaryClassificationEvaluator* are two most time-consuming stages in the Spark classification process. Both stages involve heavy workloads of *'Input'* and *'Shuffle Write'* that connect with the subset of the partitions. In this experiment, different numbers of partitions will be set to test the scalability of Spark with Logistic Regression classification on the one million cases dataset. The '*default'* partition value is the setting by Spark system during the computing process. Spark automatically sets the number of partitions of the input file according to its size. In most of the computing cases, the strategy that the number of partitions equals to the number of executors is recommended. While in data mining process, this needs a run-time tuning test based on the dataset and the algorithm involved in the computing.

The first test is designed with 8 worker nodes using the same hardware as before on Google Cloud Dataproc. The dataset is loaded and re-partitioned with the function of *'repartition()'*. Even though this procedure takes some computing time, the whole classification performance gets significant improvement compared to the *'default'* partition at some re-partition numbers. The results are shown in Table 17 and Figure 23. From the test, we can see when partition number is set to 8 it gets the best performance with 38.65% speedup ratio compared to the default number. With 8 partitions the stages of *'treeAggregate at LogisticRegression'* and *'map at BinaryClassificationEvaluator'* take around 7 to 8 seconds, but when the number of nodes reaches 32, both stages take around 17 to 19 seconds. From this test, we can get the conclusion that for the one million cases dataset, when making classification on Apache Spark with Logistic

Regression on the categorical features, one of the best resource tunings for performance can be around 8 worker nodes and 8 partitions.

Number of Nodes	Number of Partitions	Running Time(Sec.)	Speedup Ratio
8 worker nodes	default(13)	1995	
	2	1637	17.94%
	4	1479	25.86%
	8	1224	38.65%
	16	1835	8.02%
	26	2247	-12.63%
	32	2651	-32.88%
	48	3046	-52.68%

Table 17 Results of LR re-partition with One Million cases dataset



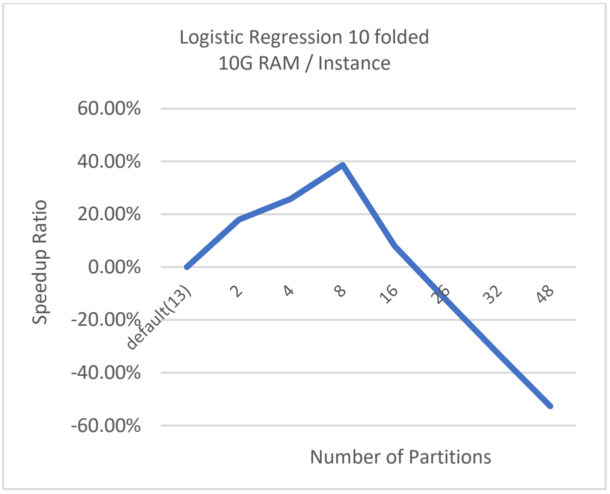
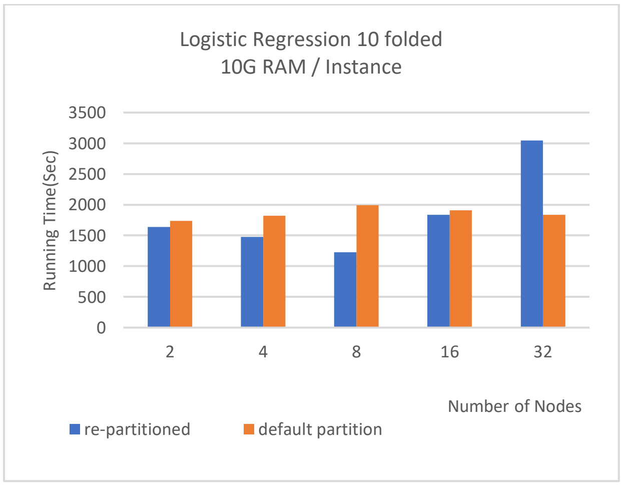


Figure 23 Results of LR re-partition with One Million cases dataset

Another experiment on re-partition is designed to compare the effect of a re-partition dataset on the different number of nodes. This time for each comparison the number of re-partitions is set to the same number of the worker nodes. The node numbers are chosen as 2, 4, 8, 16 and 32. The experiment is also on Google Cloud Dataproc Spark cluster with the same configuration as the former experiments. The results are listed in Table 18 and Figure 24. From the test, we can observe that the process of re-partition on Spark can significantly increase the computing speed of the Logistic Regression classification on the one million cases dataset with 8 worker nodes in the cluster (38.65%). It also brings some improvement with the 4 nodes and 2 nodes cluster, while with the number of the nodes increases, the performance drops with the re-partition of data as the node number.

Number of Nodes	Number of Partitions	Running Time(Sec.)	Un-partition Running time(Sec.)	Speedup Ratio
2	2	1637	1739	5.87%
4	4	1479	1819	18.69%
8	8	1224	1995	38.65%
16	16	1835	1910	3.93%
32	32	3046	1835	-65.99%

Table 18 Results of LR re-partition and the number of nodes with One Million cases dataset



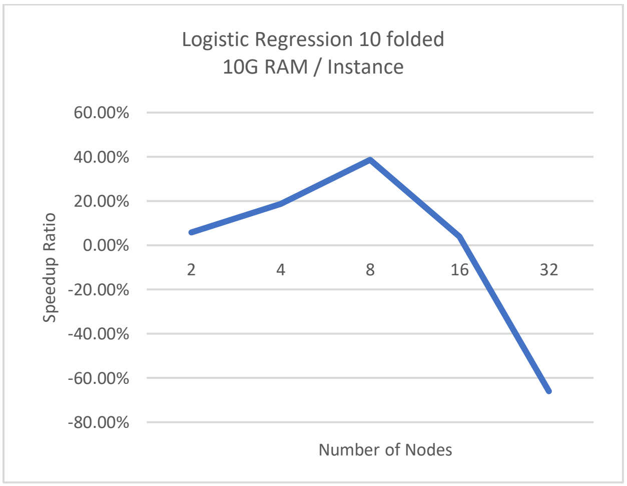


Figure 24 Results of LR re-partition and the number of nodes with One Million cases dataset

Chapter 7. Conclusion and Future Works

7.1 Conclusion

In this thesis, experiments and analysis with the scalability of Apache Spark on big data are explored based on the datasets of Cognitive Online Learning Systems. A literature review of the concept of Cognitive Learning, the process and method introduction of data mining as well as the techniques of big data with the distributed computing system of Hadoop and Spark are presented. Varied sizes and volumes of the datasets are pre-processed and loaded into classification algorithms including Logistic Regression, Decision Tree, Random Forest and Support Vector Machine with parameter tuning on a local YARN-based Spark Cluster and a Cloud YARN-based Spark Cluster. With the experiments and the results discussions, we can conclude that the distributed system of Spark Cluster could significantly improve the data mining computing speed with properly designed resource allocation strategies that make full use of the scalability of the system. For data mining with algorithms in Spark MLlib, the balance between the structure of datasets and strategies of resource allocation also needs to be considered and planned. The fundamental rule of optimization is to fully utilize the resources of the cluster, including maximum partition block size, re-partition of the datasets based on the number of executors and Vcores, increase of memory on each executor and avoid of network latency. For concurrent tasks on a cluster, the dynamic resource allocation of Spark can also be used as the solution for resources underused or misused. With the datasets and the algorithms for this thesis, the suggested resource utilization strategies to conduct the computing tasks are:

- Increase the memory for each worker node with maximum to reduce the computing time on each executor
- Re-partition to the maximum block size with the datasets based on the file systems that the datasets stored on (HDFS or GFS) to reduce the times of *map-shuffle-reduce* processes
- Submit the computing tasks to Spark with the same number of executors as the re-partitioned number to avoid resources underused

With the conclusions of the thesis, we can list some key factors for optimization of resource allocation when exploring data mining and big data analysis on the system of Apache Spark:

- The infrastructure of the computing Spark system: master and worker node hardware configurations, network, I/O, cluster type and more.
- Dataset characteristics: dataset volume, number of instances, storage strategies and more.
- Resource allocation: number of worker nodes, executor cores, executor memory and more.
- Runtime environment tuning: dataset partition number, data block size and more.

For data science, the newest version of Spark 2.2.0 can support the most commonly used classification and regression algorithms including Logistic Regression, Linear SVM, Decision Tree, Random Forest, Naïve Bayes, and etc. The kernels of these algorithms fit the distributed computing system and can significantly improve the computing speed with excellent scalability. With the limitation of this thesis, some data mining methods like clustering, Naïve Bayes and collaborative filtering on Spark are not tested in the experiments. At the same time, the pre-processed numerical attributes of the datasets are not involved in the algorithms in this thesis and will be explored in the future work.

7.2 Future Work

The datasets involved in this thesis are relatively not big enough compared to some real industry production cases, and the scalability of Spark toward those vast volumes of big data has not been fully tested especially the relationship among the size of the data blocks, the partition number and the executor number when specific machine learning algorithm applied. The KDD competition also supplies 9.4 million cases and 20 million cases datasets to download for scientific research. We will use these datasets to make more extensive research with Spark MLlib machine learning library to reveal the scalability of this distributed computing system.

Moreover, in this thesis, most of the experiments are done on the Google Cloud Dataproc platform with the Google File System (GFS) as the data file storage system. There are some limitations with the cloud computing like the network latency, the system infrastructure of different vendors, the skyrocketing cost, etc. In the future work, we will focus more on the local York University Spark Cluster with Hadoop HDFS to explore the scalability of Spark with different data mining algorithms and datasets. This could avoid the latency of network communication and data transfer between the nodes to the most extensive. It also gives us more space to explore some hidden features of Hadoop and Spark from the lower tier of the system.

Other than the system infrastructure, we will continue to explore the scalability of the distributed system of Spark on big data with more classification and regression methods as well as the unsupervised clustering algorithm. Spark Streaming, Spark SQL, GraphX and Spark Structured Streaming make the full ecosystem of the Spark and could be applied in a variety of scenarios both in scientific and production domains. More researches can be taken to combine Spark with some popular data mining and machine learning tools including numpy, pandas, scikit-learn, tensorflow, CUDA, etc. to evaluate the performance and scalability of the distributed system with different real-world datasets.

Reference

[1]. R. E. Clark, D. Feldon, J. van Merriënboer, K. Yates and S. Early. Cognitive task analysis. In J. M. Spector, M. D. Merrill, J. J. G. van Merriënboer, & M. P. Driscoll (Eds.), *Handbook of research on educational communications and technology* (3rd ed., pp. 577-593), 2007.

[2]. I. H. Witten, E. Frank, M. A. Hall and C. J. Pal. data mining: Practical machine learning Tools and Techniques (4th Edition). Morgan Kaufmann, 2016

[3]. K. R. Koedinger and V. Aleven. Exploring the assistance dilemma in experiments with cognitive tutors [J]. *Educational Psychology Review,* 2007, 19(3): 239-264.

[4]. H. Cen, K. Koedinger and B. Junker. Learning factors analysis–a general method for cognitive model evaluation and improvement[C]//*International Conference on Intelligent Tutoring Systems. Springer Berlin Heidelberg,* 2006: 164-175.

[5]. H. Cen, K. Koedinger and B. Junker. Is Over Practice Necessary?-Improving Learning Efficiency with the Cognitive Tutor through Educational data mining[J]. *Frontiers in Artificial Intelligence and Applications*, 2007, 158: 511.

[6]. M. Feng, N. Heffernan and K. Koedinger. Addressing the assessment challenge with an online system that tutors as it assesses [J]. *User Modeling and User-Adapted Interaction: The Journal of Personalization Research (UMUAI journal)*, 2009, 19(3): 243-266.

[7]. C. H. Yu, S. DiGangi, A. Jannasch-Pennell and C. Kaprolet. A data mining approach for identifying predictors of student retention from sophomore to junior year [J]. *Journal of Data Science*, 2010, 8(2): 307-325.

[8]. M. Ramaswami and R. Bhaskara. A CHAID Based Performance Prediction Model in Educational data mining, *IJCSI International Journal of Computer Science Issues*, Vol. 7, Issue 1, No. 1, January 2010

[9]. U. K. Pandey and S. Pal. data mining: A prediction of performer or underperformer using classification [J]. *arXiv preprint arXiv:*1104.4163, 2011.

[10]. J. L. Hung, Y. C. Hsu and K. Rice. Integrating data mining in program evaluation of K-12 online education [J]. *Educational Technology & Society*, 2012, 15(3): 27-41.

[11]. J. P. Vandamme, N. Meskens and J. F. Superby. Predicting academic performance by data mining methods [J]. *Education Economics*, 2007, 15(4): 405-419.

[12]. L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution[C]//*ICML*. 2003, 3: 856-863.

[13]. L. Yu and H. Liu. Efficient feature selection via analysis of relevance and redundancy [J]. *Journal of machine learning research*, 2004, 5(Oct): 1205-1224.

[14]. K.G. Srinivasa, K. R. Venugopal and L. M. Patnaik. Feature extraction using fuzzy c-means clustering for data mining systems [J]. *IJCSNS*, 2006, 6(3A): 230.

[15]. C. Romero and S. Ventura. data mining in education. *WIREs data mining Knowl Discov* 2013, 3: 12–27 doi: 10.1002/widm.1075

[16]. J. R. Anderson, A. T. Corbett, K. R. Koedinger and R. Pelletier. Cognitive tutors: Lessons learned [J]. *The journal of the learning sciences*, 1995, 4(2): 167-207.

[17]. A. Wolff, Z. Zdrahal, A. Nikolov and M. Pantucek . Improving retention: predicting at-risk students by analyzing clicking behavior in a virtual learning environment[C]//*Proceedings of the third international conference on learning analytics and knowledge*. ACM, 2013: 145-149.

[18]. W. Xing, R. Guo, E. Petakovic and S. P. Goggins. Participation-based student final performance prediction model through interpretable Genetic Programming: Integrating learning

analytics, educational data mining and theory [J]. *Computers in Human Behavior*, 2015, 47: 168-181.

[19]. B. Daniel. Big data and analytics in higher education: Opportunities and challenges [J]. *British journal of educational technology*, 2015, 46(5): 904-920.

[20]. W. He. Examining students' online interaction in a live video streaming environment using data mining and text mining [J]. *Computers in Human Behavior*, 2013, 29(1): 90-102.

[21]. M. M. Abuteir and A. M. El-Halees. Mining educational data to improve students' performance: a case study [J]. *International Journal of Information*, 2012, 2(2).

[22]. C. Márquez-Vera, A. Cano, C. Romero and S. Ventura. Predicting student failure at school using genetic programming and different data mining approaches with high dimensional and imbalanced data [J]. *Applied intelligence*, 2013, 38(3): 315-330.

[23]. D. Kabakchieva. Predicting student performance by using data mining methods for classification [J]. *Cybernetics and Information Technologies*, 2013, 13(1): 61-72.

[24]. J. L. Hung and K. Zhang. Revealing online learning behaviors and activity patterns and making predictions with data mining techniques in online teaching [J]. *MERLOT Journal of Online Learning and Teaching*, 2008.

[25]. A. Dutt, M. A. Ismail and T. Herawan. A Systematic Review on Educational data mining [J]. *IEEE Access*, 2017.

[26]. C. Romero, S. Ventura. Educational data mining: a review of the state of the art[J]. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2010, 40(6): 601-618.

[27]. A. Cano, A. Zafra and S. Ventura. An EP algorithm for learning highly interpretable classifiers [C]//*Intelligent Systems Design and Applications (ISDA)*, 2011 11th International Conference on. IEEE, 2011: 325-330.

[28]. A. Satyanarayana and M. Nuckowski. data mining using Ensemble Classifiers for Improved Prediction of Student Academic Performance [J]. *ASEE Mid-Atlantic Section Spring 2016 Conference, George Washington University, Washington D.C*, April 8-9, 2016.

[29]. S. Slater, S. Joksimović, V. Kovanovic, R. S. Baker and D. Gasevic. Tools for Educational data mining A Review [J]. *Journal of Educational and Behavioral Statistics*, 2016: 1076998616666808.

[30]. D.G. Kleinbaum and M. Klein. Survival analysis: a self-learning text [M]. *Springer Science & Business Media*, 2006.

[31]. A. T. Corbett, J. R. Anderson, and A. T. O'Brien. Student modeling in the ACT programming tutor. *Cognitively diagnostic assessment* (1995): 19-41.

[32]. R. S. J. D. Baker and K. Yacef. The state of educational data mining in 2009: A review and future visions. *JEDM-Journal of Educational data mining* 1.1 (2009): 3-17.

[33]. S. K. Yadav, B. Bharadwaj and S. Pal. Mining Education data to predict student's retention: a comparative study. *arXiv preprint arXiv:1203.2987* (2012).

[34]. J. R. Quinlan, "Induction of Decision Trees," *machine learning*, pp. 81-106, (1986).

[35]. L. Breiman Random forests. *machine learning* 45, no. 1 (2001): 5-32.

[36]. B. E. Bernhard, I. M. Guyon, and V. N. Vapnik. "A training algorithm for optimal margin classifiers." In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144-152. ACM, 1992.

[37]. "data mining Curriculum". *ACM SIGKDD. 2006-04-30. Retrieved 2014-01-27.*

[38]. I. Mavridis, and H. Karatza. "Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark." *Journal of Systems and Software* 125 (2017): 133-151.

[39]. Kurgan, Lukasz A., and Petr Musilek. "A survey of Knowledge Discovery and data mining process models." *The Knowledge Engineering Review* 21, no. 1 (2006): 1-24.

[40] S. Salman, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang. "Big data analytics on Apache Spark." *International Journal of Data Science and Analytics*(2016): 1-20.

[41]. Z. Matei. "An architecture for fast and general data processing on large clusters". *Morgan & Claypool,* 2016.

[42]. J. R. Quinlan. "Simplifying decision trees." *International journal of man-machine studies* 27, no. 3 (1987): 221-234.

[43]. K. J. Anil, "Data clustering: 50 years beyond K-means." *Pattern recognition letters* 31, no. 8 (2010): 651-666.

[44]. T. Kanungo, M. M. David, S. N. Nathan, D. P. Christine, S. Ruth, and A. Y. Wu. "An efficient k-means clustering algorithm: Analysis and implementation." *IEEE transactions on pattern analysis and machine intelligence 24,* no. 7 (2002): 881-892.

[45]. A. Toscher, and J. Michael. "Collaborative filtering applied to educational data mining." In *KDD cup* (2010).

[46]. H. F. Yu et al. "Feature engineering and classifier ensemble for KDD cup 2010." In *KDD Cup. 2010.*

[47]. J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier. "Cognitive tutors: Lessons learned." *The journal of the learning sciences 4, no. 2 (1995): 167-207.).*

[48]. Y. Shen, Q. Chen, M. Fang, Q. Yang, T. Wu, L. Zheng, and Z. Cai. "Predicting student performance: A solution for the KDD cup 2010 challenge." *In Proceedings of the KDD Cup*

2010 Workshop held as part of the 16th ACM SIGKDD Conference on Knowledge Discovery and data mining, vol. 129. 2010.

[49]. Y. Tabandeh, and A. Sami. "Classification of tutor system logs with high categorical features." *In Proceedings of the KDD 2010 cup 2010 workshop: Knowledge discovery in educational data, pp.* 54-61. 2010.

[50]. Piatetsky-Shapiro, Gregory. "Discovery, analysis and presentation of strong rules." *Knowledge discovery in databases* (1991): 229-248.

[51]. R. Agrawal, T. Imieliński, and A. Swami. "Mining association rules between sets of items in large databases." *In Acm sigmod record, vol. 22,* no. 2, pp. 207-216. ACM, 1993.

[52]. Breiman Leo. "Bagging predictors." *machine learning 24,* no. 2 (1996): 123-140.

[53]. I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan. "The rise of "big data" on cloud computing: Review and open research issues." *Information Systems 47* (2015): 98-115.

[54]. CL P. Chen, and C. Zhang. "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data." *Information Sciences 275* (2014): 314-347.

[55]. V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves et al. "Apache hadoop yarn: Yet another resource negotiator.*" In Proceedings of the 4th annual Symposium on Cloud Computing, p. 5. ACM*, 2013.

[56]. G. Salvador, J. Luengo, J. A. Sáez, V. Lopez, and F. Herrera. "A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning." *IEEE Transactions on Knowledge and Data Engineering 25,* no. 4 (2013): 734-750.

[57]. V. López, A. Fernández, S. García, V. Palade, and F. Herrera. "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics." *Information Sciences 250* (2013): 113-141.

[58]. M. Saar-Tsechansky and F. Provost. "Handling missing values when applying classification models." *Journal of machine learning research 8,* no. Jul (2007): 1623-1657.

[59]. M. Munk, M. Drlík, L. Benko, and J. Reichel. "Quantitative and Qualitative Evaluation of Sequence Patterns Found by Application of Different Educational Data Preprocessing Techniques." *IEEE Access 5 (2017): 8989-9004.*

[60]. R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. "LIBLINEAR: A library for large linear classification." *Journal of machine learning research 9,* no. Aug (2008): 1871-1874.

[61]. J. Huang, Y. F. Li, and M. Xie. "An empirical analysis of data preprocessing for machine learning-based software cost estimation." *Information and software Technology 67 (2015): 108-127.*

[62]. Z. A. Pardos, and N. T. Heffernan. "Using HMMs and bagged decision trees to leverage rich features of user and skill from an intelligent tutoring system dataset." *Journal of machine learning Research W & CP (2010).*

[63]. Stamper, J., Niculescu-Mizil, A., Ritter, S., Gordon, G.J., & Koedinger, K.R. (2010). [Data set name]. *[Challenge/Development] data set from KDD Cup 2010 Educational data mining Challenge*. Find it at http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp

[64]. M. A. Hall, "Correlation-based feature selection of discrete and numeric class machine learning." (2000).

[65]. M. Dash, K. Choi, P. Scheuermann, and H. Liu. "Feature selection for clustering-a filter solution." *In data mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference* on, pp. 115-122. IEEE, 2002.

[66]. Z. Pan, Y. Wang and W. Ku, "A new general nearest neighbor classification based on the mutual neighborhood information." *Knowledge-Based Systems 121* (2017): 142-152.

[67] N. E. I. Karabadji, Hassina Seridi, F. Bousetouane, W. Dhifli, and S. Aridhi. "An evolutionary scheme for decision tree construction." *Knowledge-Based Systems 119* (2017): 166-177.

[68] B. Liu, Y. Xiao, and L. Cao. "SVM-based multi-state-mapping approach for multi-class classification*." Knowledge-Based Systems 129* (2017): 79-96.

[69] M. Pavlekovic, M. Bensic, and M. Zekic-Susac. "Modeling children's mathematical gift by neural networks and logistic regression." *Expert systems with applications 37*, no. 10 (2010): 7167-7173.

[70] H. Lorentz, O. Hilmola, J. Malmsten, and J. S. Srai. "Cluster analysis application for understanding SME manufacturing strategies." *Expert Systems with Applications 66* (2016): 176-188.

[71] M. Kumar, and S. K. Rath. "Classification of microarray using MapReduce based proximal support vector machine classifier." *Knowledge-Based Systems 89* (2015): 584-602.

[72] J. Maillo, S. Ramírez, I. Triguero, and F. Herrera. "kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data." *Knowledge-Based Systems 117* (2017): 3-15.

[73] J. Arias, J. A. Gamez, and J. M. Puerta. "Learning distributed discrete Bayesian network classifiers under MapReduce with Apache spark." *Knowledge-Based Systems 117* (2017): 16-26.

[74] A. Shalabi, Luai, Z. Shaaban, and B. Kasasbeh. "Data mining: A preprocessing engine." *Journal of Computer Science* 2, no. 9 (2006): 735-739.

[75] B. Yan , Z. Yang, Y. Ren, X. Tan, and E. Liu. "Microblog Sentiment Classification Using Parallel SVM in Apache Spark." *In Big Data (BigData Congress), 2017 IEEE International Congress on, pp. 282-288. IEEE*, 2017.

[76] A. B. E. D. Ahmed and I. S. Elaraby. "Data Mining: A prediction for Student's Performance Using Classification Method." *World Journal of Computer Application and Technology 2, no. 2 (2014): 43-47*.

[77] B. E. Boser, I. M. Guyon, and V. N. Vapnik. "A training algorithm for optimal margin classifiers." *In Proceedings of the fifth annual workshop on Computational learning theory, pp. 144-152. ACM, 1992*.

[78] Vapnik, V. N., and A. Ya Chervonenkis. "A class of algorithms for pattern recognition learning." *Avtomat. i Telemekh 25, no. 6 (1964): 937-945*.

[79] White, Tom. "Hadoop: The definitive guide". *O'Reilly Media, Inc., 2012.*

[80] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia. "Learning spark: lightning-fast big data analysis. " *O'Reilly Media, Inc., 2015.*

Appendix

A. Coding Reference

In this thesis all the programs are coded and debugged with the environment of standalone Spark cluster of Pyspark + Jupyter Notebook on an Ubuntu 16.04 LTS. The experiments are finished on two YARN based Spark with different version of Spark. The Logistic Regression program works for both versions of Spark.

1. Logistic Regression on Spark version 2.2.0/2.0.0

```python
import csv
import pyspark
from pyspark import SparkContext, SparkConf
from pyspark.sql.session import SparkSession
import datetime
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.sql.types import IntegerType
from pyspark.sql.types import DoubleType
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

starttime = datetime.datetime.now()

conf=pyspark.SparkConf().setAppName("LR_test_10folded")
   \.set("spark.executor.memory", "10g")
   \.set("spark.executor.instances","1")
   \.set("spark.yarn.executor.memoryOverhead", "1000M")
   \.set("spark.executor.cores", "2")

sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)

#Read dataset file from the Google Cloud File System

rdd = sc.textFile("gs://xxxxx-xxxx.csv")
rdd = rdd.mapPartitions(lambda x: csv.reader(x))

#Create the Dataframe of dataset

header = rdd.first()
rdd = rdd.filter(lambda x: x!= header)
df = spark.createDataFrame(rdd,header)
df.cache()

#One-Hot categorical variables

categoricalColumns = ['ProbleNameNominal','StepNameNominal'
   \,'Proble_UnitNominal','Porble_SectionNominal']
for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol+"In
dexed")
    model_s = stringIndexer.fit(df)
    df = model_s.transform(df)
oneHotEncoder = OneHotEncoder(inputCol=categoricalCol+"Indexed",
  \outputCol=categoricalCol+"classVec")
    df = oneHotEncoder.transform(df)

#Asseble all categorical variables in to one vector feature called "categAssembVec"

vecAssembler = VectorAssembler(inputCols=["ProbleNameNominalclassVec"
   \, "StepNameNominalclassVec"
   \,"Proble_UnitNominalclassVec"
   \,"Porble_SectionNominalclassVec"]
   \, outputCol="categAssembVec")
df_New=vecAssembler.transform(df)
```

```python
df_New=df_New.withColumn("label",df_New["CorrectFirstAttempt"].cast(IntegerType()))

# dataset split to 70% - 30%

(trainingData, testData) = df_New.randomSplit([0.7, 0.3], seed = 100)

# Create initial LogisticRegression model

lr = LogisticRegression(labelCol="label", featuresCol="categAssembVec", maxIter=1)
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")

# Param tuning and CrossValidation

paramGrid = ParamGridBuilder().addGrid(lr.maxIter, [1, 2, 4]).build()
cv = CrossValidator(estimator=lr, estimatorParamMaps=paramGrid
    \,evaluator=evaluator, numFolds=10)
cvModel = cv.fit(trainingData)
predictions = cvModel.transform(testData)
Eval=evaluator.evaluate(predictions)
endtime = datetime.datetime.now()

print("Prediction Precision with 10-folded crosValidation:", Eval)
print("Running time",(endtime - starttime).seconds,"seconds")
```

2. Support Vector Machine on Spark version 2.2.0

```python
import csv
import pyspark
from pyspark import SparkContext, SparkConf
from pyspark.sql.session import SparkSession
import datetime
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.sql.types import IntegerType
from pyspark.sql.types import DoubleType
from pyspark.ml.classification import LinearSVC
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

starttime = datetime.datetime.now()

conf=pyspark.SparkConf().setAppName("LR_test_10folded")
\.set("spark.executor.memory", "10g")
\.set("spark.executor.instances","1")
\.set("spark.yarn.executor.memoryOverhead", "1000M")
\.set("spark.executor.cores", "2")

sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)

#Read dataset file from the Google Cloud File System

rdd = sc.textFile("gs://xxxxx-xxxx.csv")
rdd = rdd.mapPartitions(lambda x: csv.reader(x))

#Create the Dataframe of dataset

header = rdd.first()
rdd = rdd.filter(lambda x: x!= header)
```

```
df = spark.createDataFrame(rdd,header)
df.cache()

#One-Hot categorical variables

categoricalColumns = ['ProbleNameNominal','StepNameNominal'
\,'Proble_UnitNominal','Porble_SectionNominal']
for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol+"Indexed")
    model_s = stringIndexer.fit(df)
    df = model_s.transform(df)
oneHotEncoder = OneHotEncoder(inputCol=categoricalCol+"Indexed",
\outputCol=categoricalCol+"classVec")
    df = oneHotEncoder.transform(df)

#Asseble all categorical variables in to one vector feature called "categAssembVec"

vecAssembler = VectorAssembler(inputCols=["ProbleNameNominalclassVec"
\, "StepNameNominalclassVec"
\,"Proble_UnitNominalclassVec"
\,"Porble_SectionNominalclassVec"]
\, outputCol="categAssembVec")
df_New=vecAssembler.transform(df)
df_New=df_New.withColumn("label",df_New["CorrectFirstAttempt"].cast(IntegerType()))

# dataset split to 70% - 30%

(trainingData, testData) = df_New.randomSplit([0.7, 0.3], seed = 100)

# Create initial SVM Model
lsvc = LinearSVC(labelCol="label", featuresCol="categAssembVec", maxIter=3, regParam=0.1)
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")

# Create ParamGrid for Cross Validation

paramGrid = ParamGridBuilder().addGrid(lsvc.maxIter, [1, 2, 4]).build()

# Create 10-fold CrossValidator
clsvc = CrossValidator(estimator=lsvc, estimatorParamMaps=paramGrid
\,evaluator=evaluator, numFolds=10)

# Run cross validations
clsvcModel = clsvc.fit(trainingData)
predictions = clsvcModel.transform(testData)
Eval=evaluator.evaluate(predictions)

endtime = datetime.datetime.now()
print("Prediction Precision with 10-folded crosValidation:", Eval)
print("Running time",(endtime - starttime).seconds,"seconds")
```

3. Decision Tree on Spark version 2.2.0

```
import csv
import pyspark
from pyspark import SparkContext, SparkConf
from pyspark.sql.session import SparkSession
import datetime
```

```python
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.sql.types import IntegerType
from pyspark.sql.types import DoubleType
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

starttime = datetime.datetime.now()

conf=pyspark.SparkConf().setAppName("LR_test_10folded")
\.set("spark.executor.memory", "10g")
\.set("spark.executor.instances","1")
\.set("spark.yarn.executor.memoryOverhead", "1000M")
\.set("spark.executor.cores", "2")

sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)

#Read dataset file from the Google Cloud File System

rdd = sc.textFile("gs://xxxxx-xxxx.csv")
rdd = rdd.mapPartitions(lambda x: csv.reader(x))

#Create the Dataframe of dataset

header = rdd.first()
rdd = rdd.filter(lambda x: x!= header)
df = spark.createDataFrame(rdd,header)
df.cache()

#One-Hot categorical variables

categoricalColumns = ['ProbleNameNominal','StepNameNominal'
\,'Proble_UnitNominal','Porble_SectionNominal']
for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol+"In
dexed")
    model_s = stringIndexer.fit(df)
    df = model_s.transform(df)
oneHotEncoder = OneHotEncoder(inputCol=categoricalCol+"Indexed",
\outputCol=categoricalCol+"classVec")
    df = oneHotEncoder.transform(df)

#Asseble all categorical variables in to one vector feature called "categAssembVec"

vecAssembler = VectorAssembler(inputCols=["ProbleNameNominalclassVec"
\, "StepNameNominalclassVec"
\,"Proble_UnitNominalclassVec"
\,"Porble_SectionNominalclassVec"]
\, outputCol="categAssembVec")
df_New=vecAssembler.transform(df)
df_New=df_New.withColumn("label",df_New["CorrectFirstAttempt"].cast(IntegerType()))

# dataset split to 70% - 30%

(trainingData, testData) = df_New.randomSplit([0.7, 0.3], seed = 100)

# Create initial Decision Tree Model
dt = DecisionTreeClassifier(labelCol="label", featuresCol="categAssembVec", maxDepth=3)
```

```

# Train model with Training Data
dtModel = dt.fit(trainingData)
predictions = dtModel.transform(testData)
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")

# Evaluate best model

Eval=evaluator.evaluate(predictions)
endtime = datetime.datetime.now()
print("Prediction Precision:",Eval)
print("Running time",(endtime - starttime).seconds,"seconds")
```

4. Random Forest on Spark version 2.2.0

```
import csv
import pyspark
from pyspark import SparkContext, SparkConf
from pyspark.sql.session import SparkSession
import datetime
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.sql.types import IntegerType
from pyspark.sql.types import DoubleType
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

starttime = datetime.datetime.now()

conf=pyspark.SparkConf().setAppName("LR_test_10folded")
\.set("spark.executor.memory", "10g")
\.set("spark.executor.instances","1")
\.set("spark.yarn.executor.memoryOverhead", "1000M")
\.set("spark.executor.cores", "2")

sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)

#Read dataset file from the Google Cloud File System

rdd = sc.textFile("gs://xxxxx-xxxx.csv")
rdd = rdd.mapPartitions(lambda x: csv.reader(x))

#Create the Dataframe of dataset

header = rdd.first()
rdd = rdd.filter(lambda x: x!= header)
df = spark.createDataFrame(rdd,header)
df.cache()

#One-Hot categorical variables

categoricalColumns = ['ProbleNameNominal','StepNameNominal'
\,'Proble_UnitNominal','Porble_SectionNominal']
for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol+"In
dexed")
    model_s = stringIndexer.fit(df)
    df = model_s.transform(df)
oneHotEncoder = OneHotEncoder(inputCol=categoricalCol+"Indexed",
\outputCol=categoricalCol+"classVec")
    df = oneHotEncoder.transform(df)

#Asseble all categorical variables in to one vector feature called "categAssembVec"

vecAssembler = VectorAssembler(inputCols=["ProbleNameNominalclassVec"
\, "StepNameNominalclassVec"
\,"Proble_UnitNominalclassVec"
\,"Porble_SectionNominalclassVec"]
\, outputCol="categAssembVec")
df_New=vecAssembler.transform(df)
df_New=df_New.withColumn("label",df_New["CorrectFirstAttempt"].cast(IntegerType()))

# dataset split to 70% - 30%

(trainingData, testData) = df_New.randomSplit([0.7, 0.3], seed = 100)
# Create an initial RandomForest model.
rf = RandomForestClassifier(labelCol="label", featuresCol="categAssembVec")

# Train model with Training Data
rfModel = rf.fit(trainingData)
predictions = rfModel.transform(testData)

# Evaluate model
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")

Eval=evaluator.evaluate(predictions)

endtime = datetime.datetime.now()
print("Prediction Precision:", Eval)
print("Running time",(endtime - starttime).seconds,"seconds")
```