5-2018

Performance Enhancement of Logistic Regression for Big Data on Spark

Mengyao Wang
Purdue University

PERFORMANCE ENHANCEMENT OF LOGISTIC REGRESSION FOR BIG DATA ON SPARK

by

Mengyao Wang

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

Department of Computer and Information Technology

West Lafayette, Indiana

May 2018

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF COMMITTEE APPROVAL

Dedicated to my grandfather.

ACKNOWLEDGMENTS

I wish to gratefully acknowledge my thesis committee for their insightful comments and guidance and my family for their support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

GLM	generalized linear models
MLE	maximum likelihood estimators
ANN	artificial neural networks
RDD	resilient distributed datasets
GC	garbage collection
IRWLS	iterative re-weighted least square

GLOSSARY

Logistic regression – a regression model based on the “natural logarithm of an odds ratio,” and can be “well suited for describing and testing hypotheses about relationships between a categorical outcome variable and one or more categorical or continuous predictor variables.” (Peng, Lee, & Ingersoll, 2002)

Parallel computing – a computing technique that can “switch from sequential to modestly parallel computing” on multiple computing cores, which can enhance the efficiency of massive computations (Asanovic et al., 2009).

Spark – Apache Spark is “a general framework for distributed computing that offers high performance based on resilient distributed dataset (RDD).” (Zaharia, Chowdhury, Franklin, Shenker, & Stoica, 2010)

ABSTRACT

Author: Wang, Mengyao. M.S.
Institution: Purdue University
Degree Received: May 2018
Title: Performance Enhancement of Logistic Regression for Big Data on Spark
Major Professor: Baijian Yang

This research proposes a new fitting algorithm of logistic regression on IRWLS that utilizes the procedure of scanning data row-by-row and has the ability to acquire an exact result with only a few iterations. Furthermore, this research also realizes the distributed parallelization of the proposed method on Spark and conducts various experiments to manifest its memory-wise advantage over the traditional methods such as Spark MLlib package. The results show that the proposed method can provide an exact result rather than an approximated one within 5 or 6 iterations; achieve a satisfying accuracy for flight delay prediction within 1 or 2 iterations; has a better potential for parallelization and a better performance than MLlib with a 3-4x faster speed without full optimizations; and its performance is not undermined by an increasing data memory ratio.

CHAPTER 1. INTRODUCTION

This chapter provides an overview of the research study. It introduces the research by presenting a background of the problem area and research questions. In addition, it covers the research significance, assumptions, limitations and delimitations which define the extent of the study.

1.1 Background

Generalized linear regression models are fundamental and have been given much attention in machine learning area. The focus of this thesis, logistic regression, is one of the most commonly used training models. Although there are already many integrated packages or modules that can perform those regression models and are being applied to real-world applications, the first step is always loading the complete dataset into the memory, which has become unrealistic if the data size is too large for the memory size of a single computing system.

Because the computation and time complexity of such a regression algorithm can increase exponentially with the need to load and iterate data matrix from hard disks, and the speed of hardware improvement in memory can barely keep up with the rate of data growth nowadays, the researcher would like to conduct a research of the algorithm itself, and explore a much more efficient approach to perform it in a way that can achieve both better time and space efficiency. And such a new computational method will be applied using distributed parallel computing and a multi-core computing network, which will also be include in the scope of this research.

1.2 Scope

This research falls into two major domains of science: statistics, as in classical linear regression model; and computer science, as in applied machine learning. A better way to explain the goal is to adapt certain mathematical transformations in generalized linear models for the purpose of being utilized into practical computing that analyzes and extracts information out of a massive data input, with minimized loss of the truth, and an optimized efficiency.

From theoretical angle, this research will touch multiple classical linear regression models, especially logistic regression, while from technical angle, it will relates to commonly used techniques of big data analysis like Spark, MapReduce, parallel kernels, feature selection, etc.

1.3 Significance

Logistic regression has been regarded as one of the most commonly applied training models with already some integrated packages or modules that can perform it. However, such procedures always need their first step to load the massive dataset to the cache memory, which can be extremely hard nowadays due to the dramatic increase of data size.

Because the speed of hardware improvement in memory can barely keep up with the rate of data growth nowadays, if this research can explore a much more efficient approach to perform it both time and space efficiently, the classical logistic regression can break its bottleneck and be applied to more exciting areas. Also, parallel computing, the main technique that will be exploited, also has been paid much attention in the world of machine learning. Hopefully, this research work can demonstrate a general idea on how to transform classical statistic models into a form that can well adapt parallel realization for big data analysis.

1.4 Research Question

Can the performance of logistic regression be enhanced using distributed parallel computing, therefore it can overcome the difficulties of memory loading bottlenecks for big data, and achieve an exact results with limited number of iterations?

1.5 Assumptions

The study conducted for this project is done so assuming the follow:

- The expected prediction results will be representable in a binary form.
- The amount of data exceeds the size of RAMs.
- And for further proof of parallel efficiency, the size of data set can also exceed the storage of a single computer.
- A few times of iterations of the proposed Logistic Regression algorithm will be enough for an accuracy acceptable by the semantic meaning of the data.

1.6 Limitations

This research is conducted with the following limitations acknowledged:

- The focus on this study will be only on the Logistic Regression model.
- The dataset should be labeled or pre-processed for the training process.
- Only batch learning will be considered at the current stage.

1.7 Delimitations

This research is conducted with the following delimitations acknowledged:

- Other training models within the family of Generalized Linear Regression will not be focused on currently.
- The parallelization speedup will be bounded by the theoretical maximum potential due to the specific thread schedule setting.
- The time consumed by data pre-processing and system idling will not be considered or calculated into the total latency.

1.8 Summary

This chapter provided readers with the scope, significance, research question, assumptions, limitations, delimitations, definitions, and other background information for this research. The next chapter writes about a relevant review of the literature.

CHAPTER 2. REVIEW OF LITERATURE

Generalized linear regression models are fundamental and have been paid much attention in machine learning area. The focus of this thesis, logistic regression, is one of the most commonly used training models. Although there are already many integrated packages or modules that can perform those regression models, and being applied to real-world applications, yet the first step is always to load the complete dataset into the memory, which has become impossible now if the data size is too large for the memory size of a single computing system. While the computation and time complexity of such a regression algorithm can increase exponentially with the need to load and iterate data matrix from hard disks, the speed of hardware improvement in memory can barely keep up with the rate of data growth. This research aims to explore a much more efficient approach to perform it in a way that can achieve both better time and space efficiency. And most likely, such a new computational method will be applied using parallel computing and a multi-core computing network. So literature reviews have been conducted on current academic achievements of both logistic regression and parallel computing, as well as sub-topics related to them.

2.1 Logistic Regression

Speaking of linear regression models for machine learning there is one basic yet pragmatic member of the family that has been utilized for decades, and that is logistic regression. In this study, this algorithm realizing this specific regression model will be re-evaluated and hopefully re-designed with a new structure that can accommodate large scale parallel computing.

2.1.1 Definition

Logistic regression is “also called logit regression, is commonly used to estimate the probability that an instance belongs to a particular class,” where an instance can be predicted to be positive as 1 or negative as 0, making the whole process to be a binary classifier (Géron, 2017). Morgan and Teachman (1988) have emphasized that the key concept to understand the logistic regression models "is the odds ratio," which is "the ratio of the number of events to the number of nonevents," and should also has a clear semantic interpretation. More generally, logistic regression is a regression model "based on the natural logarithm of an odds ratio," and can be "well suited for describing and testing hypotheses about relationships between a categorical outcome variable and one or more categorical or continuous predictor variables." (Peng et al., 2002) Summarized from work of Géron (2017), The working mechanism of logistic regression can be basically interpreted as this: just similar to a linear regression model, a logistic regression model combines a bias term together with a computed weighted sum of the input features, but "instead of directly outputting the results like a linear regression model does, it outputs the logistic of those results using the logistic regression model estimated probability (vectorized form)," as shown in Eq. 2.1:

$$\hat{p} = h_\theta(\mathbf{x}) = \sigma(\theta^T \cdot \mathbf{x}) \tag{2.1}$$

where the logistic, noted $\sigma(\cdot)$, "is a sigmoid function (i.e., S-shaped) that outputs a number between 0 and 1," and is defined via a logistic function as Eq. 2.2 shows:

$$\sigma(t) = \frac{1}{1 + exp(-t)} \tag{2.2}$$

which yields a model prediction equation as Eq. 2.3 shows:

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5, \\ 1 & \text{if } \hat{p} \geq 0.5. \end{cases} \tag{2.3}$$

Using those fundamental equations, the logistic regression model can make the prediction easily, once it has estimated the probability $\hat{p} = {}_\theta(\mathbf{x})$, where it predicts 1 if an instance **x** belongs to the positive class, and 0 if negative.

2.1.2 Significance

But why is logistic regression important? And how it becomes essential and fundamental to machine learning area? According to Lemeshow and Hosmer Jr (1982), back to the 90s of 20th century, the usage logistic regression model was dominating for a decade and “regarded as a standard method for data analysis in epidemiologic studies,” which had its rapid development at that time; and as assumed by them, “its widespread application is probably due to its ease of interpretation as well as its relationship to log-linear discriminant function analysis.” And it was claimed by Schein and Ungar (2007) that “the last decade has also seen increased use of the logistic regression classifier in machine learning applications, though under different names: multinomial regression, multi-class logistic regression or the maximum entropy classifier.” From all those articles that was published ten years or even thirty years ago, it is shown that logistic regression has been rather a mature and fully researched method than a fresh and newly born terminology. However, its long history does not lessen its significance and usefulness over the decades, and in the opposite, the public source codes of many cutting-edge machine learning products leading the market these years have indicated their originating from logistic regression, or even still utilizing it as the main training model.

On the other hand, logistic regression is an outstanding representative of the family of generalized linear models (GLMs), which are fundamental and have been paid much attention to in statistic since they were proposed. GLMs represent a broad of statistical approaches corresponding to binary, binomial, multinomial, or Poisson data for a count response, and normal or gamma data for a continuous response. However, except for the normal case, maximum likelihood estimators (MLEs) of model parameters cannot be analytically resolved. Thus when using logistic regression, numerical methods must be used, including the Netwon-Raphson, the Fisher-scoring, and the iteratively re-weighted least square methods (IRWLS). Those methods mentioned above have been incorporated in many standard packages such as R and SAS. But to use these packages, the first step is always loading the complete dataset into the memory, which is not doable if the data size is much exceeding the memory size of the computing system. Therefore, the significance of this research can be revealed, since the goal is to propose a new approach which can overcome this difficulty, and to make logistic regression more compatible to modern large-scale machine learning projects.

2.1.3 Logistic regression in machine learning

To build a legal case, materials that proving a close relationship between logistic regression and machine learning are provided as the followings.

Artificial neural networks (ANN), one of the most popular machine learning methods nowadays, has been proved sharing many similarities with logistic regression. Mathematically, they “both provide a functional form and parameter vector to express as,” where the parameters “are determined based on the data set, usually by maximum-likelihood estimation,” and “as the functional form of differs for logistic regression and artificial neural nets, the former is known as a parametric method, whereas the latter is sometimes called semi-parametric or non-parametric.” (Dreiseitl & Ohno-Machado, 2002) So to some extends, connections can be drawn such that artificial neural networks are derived forms of the basic logistic regression model.

Also, from the angle of application, it shows the trace of usage of logistic regression in many popular fields, for example, the text categorization, or in another name, natural language processing. Genkin, Lewis, and Madigan (2007) claimed that in text categorization, it is often to use Ridge logistic regression “in combination with certain feature selection, producing sparser and more effective classifiers”; on the other hand, Lasso logistic regression also fits the criterion because it can provide “state-of-the-art text categorization effectiveness while producing sparse and thus efficient model,” and also because of its usefulness in “other high-dimensional data analysis problems, such as predicting adverse drug events.”

Combining logistic regressions relationship with artificial neural network, and their usages in medical field, there is also an interesting comment from Tu (1996), as the following:

> “Artificial neural networks are algorithms that can he used to perform nonlinear statistical modeling and provide a new alternative to logistic regression, the most commonly used method for developing predictive models for dichotomous outcomes in medicine. Neural networks offer a number of advantages, including requiring less formal statistical training, ability to implicitly detect complex nonlinear relationships between dependent and independent variables, ability to detect all possible interactions between predictor variables, and the availability of multiple training algorithms. Disadvantages include its ‘black box’ nature, greater computational burden, proneness to overfitting, and the empirical nature of model development.”

This comparison reveals the fact that, although artificial neural network is sometimes regarded as an alternative of logistic regression, the former can bring more unnecessary burden due to its complexity while the latter, if improved properly, has a potential to boost both the efficiency and accuracy.

2.1.4 Bottlenecks of large-scale logistic regression

Although logistic regression has been widely used in large-scale machine learning projects, it still has some bottlenecks that have been tried to tackle for years. One of them is overfitting. Liu, Chen, and Ye (2009) analyzed this difficulty in their research, by stating that if logistic regression is applied to applications with large amount of features but limited training data samples, then it tends to overfit the model, and then it usually needs further regularization to obtain a classifier that is more robust. They also proposed an algorithm called “Lassplore” to solve the large-scale sparse logistic regression. More specifically, they “formulate the sparse logistic regression problem as the ball constrained smooth optimization problem,” and they also proposed to “solve the problem by the Nesterovs method, an optimal first-order black-box method for the smooth convex optimizatio.” However, another critical issue along with their solution was the difficulty to estimate the “step size at each of the optimization iterations.”

Furthermore, in the study conducted by Mood (2010), he emphasized that when interpreting the estimates as cause-effect relationships when using linear regression such as logistic regression, researches should be more cautious because “it is difficult to control for all factors related to both independent and dependent variables, but it is of course even more difficult to control for all variables that are important for explaining the dependent variable.” Because of the mathematical mechanism behind logistic regression, if a small variance has been found and explained, an “unobserved heterogeneity is almost always present,” so the risk to conclude causal relationships should always be considered.

And finally, the efficiency of data loading is also an obvious barrier, which this research is aiming to solve. The trouble in classical techniques is caused by the two steps fitting procedure. Since statistical methods are not involved in the first step, the entire date set must be completely loaded to memory for further analysis. In other words, as the two steps are conducted separately and independently, classical techniques are not efficient in operations of computing resources. To solve the problem, the technique of scanning data by rows should be proposed. Since logistic regression only loads individual rows to memory, the technique can handle extremely large data with size exceeding the memory size of the computing system. After individual rows are loaded sequentially, a summary information set of previous records is obtained, so that the final result of the summary information should be obtained after the last row is loaded.

Since the estimates of model parameters and its variance-covariance matrix are computed from the summary information, such a technique of scanning data by rows will be extremely efficient in fitting a linear regression model, and therefore can overcome both the memory and computational efficiency barriers, which are the two most important to be addressed in big data analysis. And to be noted, this research tends to design a data scanning method that only accesses the entire data set once, which should be able to further enhance the efficiency.

2.2 Parallel Computing

The data nowadays expands dramatically on size, soon the data set can be too large to be accessed to a single or a few hard disks. And parallel computing is one of the major topics for exploring plausible solution to the bottlenecks of logistic regression mentioned above.

2.2.1 Significance of parallel computing in machine learning

A prospective view that "driven by the capabilities and limitations of modern semiconductor manufacturing, the computing industry is currently undergoing a massive shift towards parallel computing, and this shift brings dramatically enhanced performance to those algorithms which can be adapted to parallel computers" has been highlighted in the research work of Catanzaro, Sundaram, and Keutzer (2008). And by studying their research achievement around utilizing GPU computation on support vector machine method, it is found that GPU is a way with very low cost yet guarantees such high performances. Catanzaro et al. (2008) also pointed out that "new machine learning algorithms that can take advantage of this kind of performance, by expressing parallelism widely, will provide compelling benefits on future many-core platforms." And there are also many research work out there indicating parallel computing is an inevitable improvement for modern big data analysis.

2.2.2 Spark

Spark, a framework that "supports applications with working sets while providing similar scalability and fault tolerance properties to MapReduce," and according to Zaharia et al. (2010), to achieve those goals, Spark also introduces an abstraction which is called resilient distributed datasets (RDDs), a "read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost." Also, comparing to Hadoop, its main competitor, Spark "outperform by 10x in iterative jobs of machine learning, and also can be used to query a dataset with size of 39 GB with sub-second response time," which means for this research to boost performance of logistic regression, Spark will be the top on the preference list.

This research will utilize the Hadoop Distributed File System (HDFS) to store the input data, and partitioning functions of Spark to parallelize the input RDD over different cores of the cluster, thus realize the distributed computations.

2.2.3 MapReduce

MapReduce is an essential component in a machine learning pipeline. To optimize the classical logistic regression, map-reduce is a key part to concur, and provide related optimization on that as well. This concept firstly came from Google, who specializes it for clusters that generate unreliable communication and in which process individual computers may shut down in an unexpected way. However, map-reduce now has been widely utilized in various kinds of machine learning projects, where its basic steps can be concluded as the following: the master engine coordinates the sub-engines called mappers and reducers, and it is responsible for splitting the data and assigning them into different mappers and collects the intermediate data transferred back from the mappers, while the reducer will be asked by the master to process the data and then return final results (Chu et al., 2007).

Also, mentioned by Dean and Ghemawat (2008), MapReduce has an outstanding performance on easing the burden of network bandwidth, by "reducing the amount of data sent across the network: the locality optimization allows us to read data from local disks, and writing a single copy of the intermediate data to local disk saves network bandwidth." They also pointed out that "the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault tolerance, locality optimization, and load balancing." Therefore, it is preferred to use MapReduce on Spark to complete the data processing for this research.

2.2.4 Paralleling logistic regression

In the research work of Singh, Kubica, Larsen, and Sorokina (2009), they proposed an optimized algorithm for logistic regression that has been paralleled, which is "based on the map-reduce framework, for performing feature evaluation," and it "makes feature evaluation tractable on massive datasets," and furthermore, it "can trivially be applied to the SFO heuristic as well as other known heuristics." Although their work sounds promising, but it still has not resolved the data loading issue completely, where there is a great potential to make improvements using the same MapReduce procedure.

Because of the nice property of the technique of scanning data by rows, as proposed in the previous chapter, it is possible to extend GLMs and logistic regression for big data. The size of sufficient statistics in a logistic model does not depend on the sample size, but mostly the size of sufficient statistics in GLMs for non-Gaussion data cannot be lower than the size of the whole data, therefore the simply using of sufficient statistics cannot overcome the memory barrier in fitting GLMs as well as logistic regression for big data.

So to solve the problem, two scenarios must be addressed. In the first, it is assumed that the data size is lower than the storage capacity limitation of a single computer so that the proposed approach will be applied on a single processor. In the second, it is assumed the size of big data exceeds the limit of storage capacity of single computer, where multiple disks must be used, then the proposed approach will be applied to multiple processors.

For the multiple processors scenario, the implementation needs data-parallel computation executed on clusters of processors by a distributed file system, where MapReduce will be the pioneer solution. With MapReduce, identical computations are applied onto a enormous number of data records by a lot of processors. Different jobs will be specified as the Maps and Reduces, and divide the input data into independent small data sets that can be processed in a parallel pattern.

The major task in the initial parallelized implementation is the derivation of the coefficients and parameters. Once they are computed, the computation burden will be independent of the data size, and theoretically, the entire computation will need only $\mathscr{O}((p+1)^2)$ memory size, where is the number of processors. And as the computational time is linear to the data size, the usage of parallel computation can increase the speed of the overall process.

And finally, it is still important to study the theoretical relationship between the proposed approach with the classical approach in fitting GLMs for big data. After the results of classical fitting procedures provided by existing packages run on the same cluster and data set, it can be shown that results from the proposed approach are identical to those from the traditional methods, indicating that it is able to classify this new method as an exact approach.

2.3 Summary

This chapter provided a review of the literature relevant to Logistic Regression, Parallel Computing and their potentials.The next chapter provides the framework and methodology to be used in the research project.

CHAPTER 3. FRAMEWORK AND METHODOLOGY

The aim of this research is to enhance the performance of logistic regression, a popular approach of classification in machine learning. A full description of the methodology is provided in this chapter, including framework research, hypotheses, the proposed fitting algorithm on IRWLS, distributed parallelization on spark, assessment instrument, variables, and procedures of testing and analysis.

3.1 Framework of Research

The framework of research includes four stages:
First, algorithm design:

- Derive and refine the new method of scanning data row-by-row using Fisher Scoring and Iterative Reweighted Least Square (IRWLS).

Second, simulation:

- Realize algorithm with Python in serial programming with one iteration;
- Feed fake data for testing the validity of algorithm;
- Implement the iterative method;
- Set up Spark cluster;
- Distributively parallelize the program onto Spark using MapReduce and Aggregation method.

Third, data preparation:

- Feature selection;
- Data merging and cleaning;
- Handle categorical feature with one-hot encoding.

Fourth and the last, conducting experiments:

- Train the models;
- Design performance metrics and related experiments;
- Testing, scoring, summarizing.

So some key milestones in the research framework includes: combine Fisher Scoring and Iteratively Re-weighted Least Squares (IRWLS) in the proposed algorithm to change the data loading pattern from entirely to row-by-row; and on single CPU, deriving the first iteration of algorithm, then developing on further iterations to find a pivot for acceptable accuracy; utilizing distributed file system like Spark to further overcome the memory-wise and computational bottlenecks; finally, conducting experiment with a baseline of existing logistic regression packages. In general, the primary goal of the research is to complete an exact approach development by adapting the classical algorithm to parallelization via new data loading method and rules of iteration, making it well performed on multi-core environment.

3.2 Hypotheses

Since after certain amount of work in implementation, the new algorithm and model will be tested on data of big size and analyze the performance, this research is therefore mainly a quantitative one and has the following hypotheses:

- H_0: The proposed method cannot achieve an exact result with a better memory-wise performance than the baseline.
- H_1: The proposed method can achieve an exact result with a better memory-wise performance than the baseline.

3.3 The Proposed Fitting Algorithm on IRWLS

The IRWLS algorithm has an initial guess of the linear component using Eq. 3.1, and an initial guess of the weight using Eq, 3.2

$$z_{i,F}^{(0)} = \eta_i^{(0)} \tag{3.1}$$

$$w_{i,F}^{(0)} = b''[h(\eta_i^{(0)})][h'(\eta_i^{(0)})]^2 \tag{3.2}$$

And for iterative calculations, this research proposes the ways to obtain the weight β, the estimator σ_F^2, and the variance-covariance matrix $\mathbf{V}$ using Eq. 3.3, 3.4, and 3.5, respectively.

$$\beta^{(t+1)} = \{\mathbf{S}_{xx,F}^{(t)}\}^{-1}\mathbf{s}_{xz,F}^{(t)} \tag{3.3}$$

$$\{\sigma_F^2\}^{(t+1)} = \frac{1}{n}\left\{s_{zz,F}^{(t)} - \{\mathbf{s}_{xz,F}^{(t)}\}^{\top}\{\mathbf{S}_{xx,F}^{(t)}\}^{-1}\mathbf{s}_{xz,F}^{(t)}\right\}. \tag{3.4}$$

$$\mathbf{V}^{(t+1)} = (\mathbf{X}^{\top}\mathbf{W}_F^{(t)}\mathbf{X})^{-1}. \tag{3.5}$$

Algorithm 3.1 Fisher Scoring and IRWLS for $\hat{\beta}$ and $\hat{\phi}$ Based on A single Processor

Hence the proposed fitting algorithm on IRWLS can be designed as demonstrated in Algorithm 3.1.

```
Input: data read row-by-row from the hard disk
Output: $\hat{\beta}, \hat{\sigma}^2, \hat{V}(\hat{\beta})$
 1: procedure ITERATIVE ALGORITHMS WITH THE TECHNIQUE OF SCANNING DATA BY ROWS
    Initial Computation:
 2: Let $s^{(0)}_{zz,F}$, $\mathbf{s}^{(0)}_{xz,F}$, and $\mathbf{S}^{(0)}_{xx,F}$ be a value, a $p$-dimentional vector, and a $p \times p$-dimensional matrix, all equal to zero
 3: For each the $i$th row of data do
 4:     Define $z^{(0)}_{i,F}$ by Eq. 3.1 and $w^{(0)}_{i,F} by Eq.3.2$
 5:     Update $s^{(0)}_{zz,F} = s^{(0)}_{zz,F} + w^{(0)}_{i,F}\{z^{(0)}_{i,F}\}^2$
 6:     Update $\mathbf{s}^{(0)}_{xz,F} = \mathbf{s}^{(0)}_{xz,F} + w^{(0)}_{i,F}\{z^{(0)}_{i,F}\}x_i$
 7:     Update $\mathbf{S}^{(0)}_{xx,F} = \mathbf{S}^{(0)}_{xx,F} + w^{(0)}_{i,F}x_i x_i^T$
 8: end for
 9: Compute $\beta^{(1)}$ by Eq. 3.3, $\sigma^{2(1)}_F$ by Eq, 3.4, and $\mathbf{V}^{(1)}$ by 3.5
    Iterative Computation:
10: Let $s^{(t)}_{zz,F}$, $\mathbf{s}^{(t)}_{xz,F}$, and $\mathbf{S}^{(t)}_{xx,F}$ be a value, a $p$-dimentional vector, and a $p \times p$-dimensional matrix, all equal to zero
11: For each the $i$th row of data do
12:     Let $\eta^{(t)}_i = x_i^T\beta^{(t)}$, $\mu^{(t)}_i = g^{-1}(\eta^{(t)}_i)$, $w^{(t)}_{i,F} = b''[h(\eta^{(t)})][h'(\eta^{(t)}_i)]^2$ and $z^{(t)}_{i,F} = \eta^{(t)}_{F,i} - (y_i - \mu^{(t)}_i)/w^{(t)}_{F,i}$
13:     Update $s^{(t)}_{zz,F} = s^{(t)}_{zz,F} + w^{(t)}_{i,F}\{z^{(t)}_{i,F}\}^2$
14:     Update $\mathbf{s}^{(t)}_{xz,F} = \mathbf{s}^{(t)}_{xz,F} + w^{(t)}_{i,F}\{z^{(t)}_{i,F}\}x_i$
15:     Update $\mathbf{S}^{(t)}_{xx,F} = \mathbf{S}^{(t)}_{xx,F} + w^{(t)}_{i,F}x_i x_i^T$
16: end for
17: Compute $\beta^{(t+1)}$ by Eq. 3.3, $\sigma^{2(t+1)}_F$ by 3.4, and $\mathbf{V}^{(t+1)}$ by 3.5
18: Iterate Step 10 to Step 17 until convergence
19: end procedure
```

3.4 Distributed Parallelization on Spark

The implementation structure of the proposed method on Spark can be illustrated as Figure 3.1.

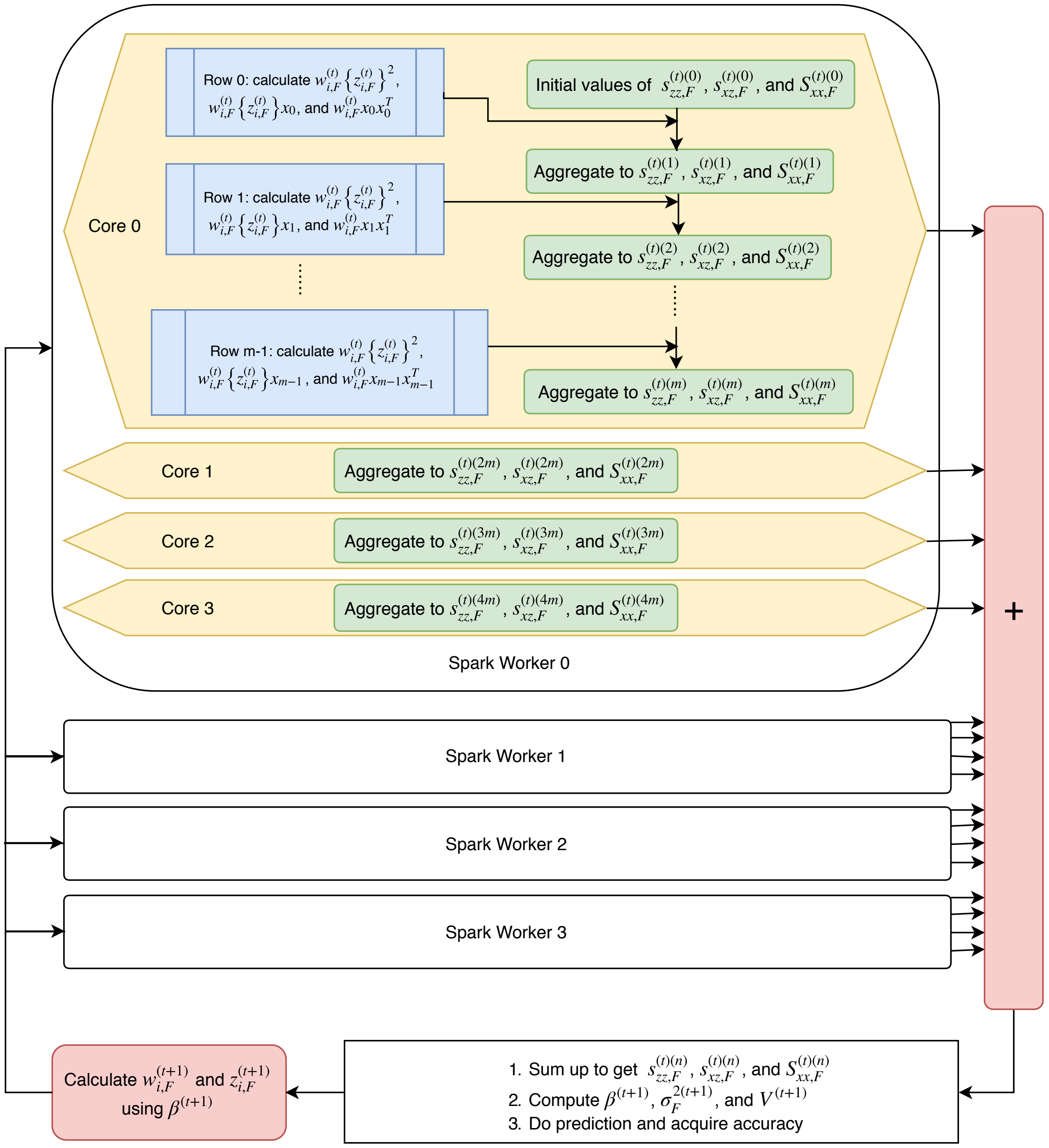
Row 0: calculate $w_{i,F}^{(t)}\left\{z_{i,F}^{(t)}\right\}^2$, $w_{i,F}^{(t)}\left\{z_{i,F}^{(t)}\right\}x_0$, and $w_{i,F}^{(t)}x_0x_0^T$
Row 1: calculate $w_{i,F}^{(t)}\left\{z_{i,F}^{(t)}\right\}^2$, $w_{i,F}^{(t)}\left\{z_{i,F}^{(t)}\right\}x_1$, and $w_{i,F}^{(t)}x_1x_1^T$
Row m-1: calculate $w_{i,F}^{(t)}\left\{z_{i,F}^{(t)}\right\}^2$, $w_{i,F}^{(t)}\left\{z_{i,F}^{(t)}\right\}x_{m-1}$, and $w_{i,F}^{(t)}x_{m-1}x_{m-1}^T$
Initial values of $s_{zz,F}^{(t)(0)}$, $s_{xz,F}^{(t)(0)}$, and $S_{xx,F}^{(t)(0)}$
Aggregate to $s_{zz,F}^{(t)(1)}$, $s_{xz,F}^{(t)(1)}$, and $S_{xx,F}^{(t)(1)}$
Aggregate to $s_{zz,F}^{(t)(2)}$, $s_{xz,F}^{(t)(2)}$, and $S_{xx,F}^{(t)(2)}$
Aggregate to $s_{zz,F}^{(t)(m)}$, $s_{xz,F}^{(t)(m)}$, and $S_{xx,F}^{(t)(m)}$
Core 0
Core 1
Aggregate to $s_{zz,F}^{(t)(2m)}$, $s_{xz,F}^{(t)(2m)}$, and $S_{xx,F}^{(t)(2m)}$
Core 2
Aggregate to $s_{zz,F}^{(t)(3m)}$, $s_{xz,F}^{(t)(3m)}$, and $S_{xx,F}^{(t)(3m)}$
Core 3
Aggregate to $s_{zz,F}^{(t)(4m)}$, $s_{xz,F}^{(t)(4m)}$, and $S_{xx,F}^{(t)(4m)}$
Spark Worker 0
Spark Worker 1
Spark Worker 2
Spark Worker 3
+
1. Sum up to get $s_{zz,F}^{(t)(n)}$, $s_{xz,F}^{(t)(n)}$, and $S_{xx,F}^{(t)(n)}$
2. Compute $\beta^{(t+1)}$, $\sigma_F^{2(t+1)}$, and $V^{(t+1)}$
3. Do prediction and acquire accuracy
Calculate $w_{i,F}^{(t+1)}$ and $z_{i,F}^{(t+1)}$ using $\beta^{(t+1)}$

Figure 3.1. Distributed parallelization implementation of the proposed method on Spark

The data is randomly split into p number of RDD partitions, each of which has m number of rows of records, and they are assigned to each core of each Spark worker as a parallelized task. Within each partition, each row is mapped with the function calculating the working sufficient statistics, and then the results of each row are reduced by aggregation into a final output. The tasks of different partitions can be distributed parallelized and executed no matter of the order. Eventually the partial working sufficient statistics will be collected and summed up to get the ultimate result for an iteration, based on which the weight β as well as other necessary values will calculated. Using $\beta^{(t+1)}$ of the t_{th} iteration, $z_{i,F}^{(t+1)}$ and $w_{i,F}^{(t)+1}$ can be updated and fed into the $(t+1)_{th}$ iteration. This circulation continues until manual stopping or automatic result convergence.

3.5 Assessment Instrument

In this section the experiment environment including both hardware and software will be introduced, as well as the dataset information and pre-processing methods.

3.5.1 Hardware environment

The spark cluster is built on four virtual machines with the following specifications:

Table 3.1. Hardware specifications of a single virtual machine

CPU model	AMD Opteron(TM) Processor 6272
Number of cores	4
CPU MHz	2100.025
Memory	8 G
Disk size	250 GiB

Hence the cluster has 16 cores and 32G of memory in total.

3.5.2 Software environment

Apache Spark 2.0.0 is installed over the four virtual machines, with one master nodes and worker nodes. The master node not only acts as a driver, but also has a secondary name node as a worker, so the cluster can be considered to have four workers with 16 CPU cores and 32 GB of memory in total. The program of new algorithm is written in Python 2.7 using some common packages such as Numpy and Pandas. Also the logistic regression package from MLlib, a machine learning library of Spark, is utilized as the baseline for performance comparison.

The baseline package is called `LogisticRegressionWithLBFGSF`, which is a logistic regression algorithm that has been already optimized using Limited-memory BFGS and Tree Aggregation, the latter of which can effectively shorten the time of result aggregation and communication cost among parallelized tasks. So readers should note that the advantage of the method proposed by this research is actually even more than it manifests in the experiments due to the normal aggregation method that has been used during the implementation.

3.5.3 Dataset and data pre-processing

For the purpose of testing the proposed logistic regression algorithm, this research selects to predict whether the arrival of a pre-scheduled flight will be delayed (“true” if delayed more than 15 minutes, and “false” otherwise), based on the historical on-time performance provided by Bureau of Transportation Statistics.

The data contains more than 100 columns, among which only 13 (12 features and one label) are used, which is shown in Table 3.2. `ARR_DEL15` is the label to be predicted while the 12 features will be used fully or partially according to the need of different data size or semantic meaning. The Bureau of Transportation Statistic database provides data from the year of 1988 - 2016, however data before the year of 1995 has significant differences such feature missing, old version of identification numbers, etc., so only data of the year 1995 - 2016 has been used. The total data size of this year length is about 35

GB, and after being trimmed to the selected 13 columns it has approximately 10 GB remained.

Although 8 GB of data can hardly be recognized as Big Data, the purpose of this research will not be undermined for the following two reasons: firstly, Spark allows users to specify memory usage for each node, enabling simulation of the scenario when data size is much larger than the total memory size even with dataset that is not actually huge; and secondly, One-hot Encoding, a well-known method of data pre-processing that enlarges an $1 \times n$ column of p number of distinct values to a $p \times n$ matrix, where each row contains $p-1$ 0s and one "1" to indicate which value this record has, can be applied to the several categorical features such as `FL_NUM`, `AIRLINE_ID`, `ORIGINAL_AIRPORT_ID`, and `DEST_AIRPORT_ID`. One-hot Encoding will not only make categorical feature more scientifically presented, but also can significantly enlarge the dataset that has limited size because of real-world collection for the purpose of manifesting the memory-wise advantage of the proposed algorithm in this research.

Table 3.2. Selected Columns (features/label) of Flight Data

`MONTH`	Month	1-12
`DAY_OF_MONTH`	Day of Month	1-31
`DAY_OF_WEEK`	Day of Week	1-7
`AIRLINE_ID`	Identification number of a unique airline (carrier)	e.g. 32575
`FL_NUM`	Flight Number	e.g. 1933
`ORIGIN_AIRPORT_ID`	Identification number of a unique origin airport	e.g. 14492
`DEST_AIRPORT_ID`	Identification number of a unique destination airport	e.g. 12266
`CRS_DEP_TIME`	CRS departure time (local time: hhmm)	e.g. 1350
`CRS_ARR_TIME`	CRS arrival time (local time: hhmm)	e.g. 1912
`DEP_DEL15`	Departure delay indicator, 15 minutes or more	1 or 0
`TAXI_OUT`	Taxi out time, in minutes	e.g. 21
`DISTANCE`	Distance between airports (miles)	e.g. 1042
`ARR_DEL15`	Arrival delay Indicator, 15 minutes or more	1 or 0

3.6 Variables

The independent variables of this research include the size of data input (decided by number of features and records across time), the assigned memory size of Spark nodes, the number of partitions of RDDs, and the number of iterations of the proposed IRWLS method. And the dependent variables that will be measured include accuracy predictions, predicted weights (β), running time, parallelization speedup, size of communication throughputs, and garbage collection time of Spark tasks,

3.7 Procedures of Testing and Analysis

After the proposed IRWLS method has been successfully tested to run in the serial version, i.e., with only one core making correct calculations for both training and predicting over multiple iterations, and giving results that are acceptable in both time and accuracy manner, the implementation will be adapted onto Spark cluster for further speedup with distributed parallelization using the schema described in section 3.3. Then once the numerical results such as accuracy and the weight β are identical to the serial version, it can be safe to say the algorithm is prepared for performance testing and analysis.

The experiments will be conducted in three stages as the followings:

- Parallelization performance analysis
- Accuracy and β convergence examination over iterations
- Performance analysis with different settings of data memory ratio

Firstly, the parallelization performance analysis experiment will be conducted on a relatively small data consisting of one year of records, five features and one label. In this research the flight data in year of 1995 has been used. The aim of using a dataset smaller than granted memory size is to exclude the effect of data loading difficulty and purely test the speed enhancement of the parallelization alone. The baseline package `MLlib` will be run on the same degree of parallelization and its results and metrics will be compared to the proposed logistic regression method.

Secondly, since the proposed method of this research emphasizes the need of only a small number of iterations and an exact result instead of an approximated one can be derived in the end of the algorithm, it is necessary to make an observation on running multiple iterations of the proposed method until the result converges, and evaluate its actual performance with the theoretical expectations. Since only the mathematical results will be examined and a larger number of iterations will be run, this part of experiment will also use the single year of dataset in year 1995 as well as the five features used in the previous experiment.

The third experiment has its focus on the main advantage about how reading data row-by-row can free the performance of logistic regression from growing data memory ratio that is happening in Big Data analysis. Dataset from the year of 1995 to 2015 will be used to compose training inputs with different size and all the 12 selected features will be introduced for a full performance analysis, and a part of data of the year 2016 of the same amount of features will act as the testing dataset. Both the proposed method and the baseline package will be tested in two scenarios: fixed small memory size with increasing data size, and fixed large data size with decreasing memory size.

A major advantage for using Spark is that it has a well-integrated web UI that monitors all the performance metrics of each stage of jobs submitted, and furthermore the results can be viewed even after stopping the applications in the history server that Sparks provides. Hence this research will combine the results from the web UI and outputs from the program itself for further analysis. And after the experiments are done, charts and tables will be demonstrated in a variable-controlling manner to manifest the performance enhancement of the proposed method.

3.8 Summary

This chapter provided the framework and methodology to be used in the research study.

CHAPTER 4. RESULTS AND DISCUSSION

In this chapter, the different stages of experiments designed in section 3.7 are conducted and results have been collected. Both the proposed IRWLS method and the baseline MLlib package `LogisticRegressionWithLBFGS` are tested using the same groups of data input and memory settings on Spark cluster whose specifications can be found in section 3.5. The first experiment analyzes the performance enhancement due to distributed parallelization; the second experiment examine the accuracy and result convergence over iterations of the proposed IRWLS method; and the third experiment records and compares performances of the two methods with different settings of data memory ratio to prove the memory efficiency of the proposed method. This research will provide further analysis and discussion at the end of each stage of experiments.

4.1 Parallelization Performance Analysis

Spark is well-known as a fast large-scale data processing engine due to many of its brilliant features including fault-tolerant RDD implementation, which provides a great platform for distributed parallelization. In this experiment, the serial version of the proposed IRWLS method will be adapted to run in parallel over cluster cores to achieve a significant speed up. And the baseline MLlib logistic regression package will perform the same tasks under the same conditions for comparison.

4.1.1 Testing parameters and input conditions

The basic information of this experiment is illustrated in Table 4.1. As discussed in section 3.7, for the purpose of testing parallelization performance alone, this research picks up a relatively small dataset compared to the assigned memory size for this stage.

Table 4.1. Parameters for single year test for parallelization performance analysis

Parameters	Value
Year of data	1995
Total size	1.54GB
Number of features	5
Features	`MONTH`, `AIRLINE_ID`, `FL_NUM`, `DEP_DEL15`, `DISTANCE`
Actual size being computed	122 MB for training + 34MB for testing
Number of records	4,177,444 for training + 1,044,361 for testing
Memory size	4 GB * 4 = 16 GB
Number of partitions	1, 2, 4, 8, 12, 16

The data of the year 1995 has been shuffled and split in a ratio of 0.8 for training and 0.2 for testing. The proposed IRWLS method will run for three iterations that would be enough for performance analysis at the current stage. And the RDD will be partitioned into 1, 2, 4, 8, 12, and 16 for different level of parallelization. Since the cluster is built on 4×4 cores, the task will be executed with a $task/executor$ ratio of $1/1$, $2/2$, $4/4$, $8/4$, $12/4$, and $16/4$, respectively.

4.1.2 Illustration of results

For a more precise analysis, each experiment element is conducted 10 times with the same parameters and environment, and a final result of average will be recorded. For this stage of experiment, the full set of original results are illustrated in Table 4.2, 4.3 and 4.4. Those tables list the training time results from 10 identical tests and the average is calculated and shown at the rightmost column. The rest of the experiments of this research use the same procedure to acquire the average of 10 identical tests. And for the reason of simplicity, only this stage shows the expanded table of full records while the later stages will only present the final average results.

The dependent variable, the number of partitions, i.e., the max number of active tasks running concurrently with distributed parallelization over the 16 cores, is noted as p, which spreads from 1 to 16, which is the maximum number of effective partitions due to the 16 cores provided by the Spark cluster. For the proposed IRWLS method, this experiment records the result of the first three iterations for each setting of p.

Table 4.2. Running Time of the proposed IRWLS method (part 1)

p	Iteration	Time for Training (s)					
		Test 1 Test 6	Test 2 Test 7	Test 3 Test 8	Test 4 Test 9	Test 5 Test 10	Average
1	1	327.51 330.22	334.10 324.94	331.53 326.62	314.36 332.59	335.58 331.68	328.91
	2	552.34 549.68	556.90 557.39	557.44 562.60	565.88 559.02	549.68 553.69	557.62
	3	557.63 560.79	559.72 562.17	548.41 564.36	554.01 550.78	568.47 558.65	558.50
2	1	165.32 170.50	157.06 158.71	162.29 165.22	160.62 165.48	172.83 164.08	164.21
	2	280.85 283.28	280.29 285.09	284.40 282.05	277.73 281.42	286.51 282.71	282.43
	3	279.16 276.25	280.73 278.79	280.13 274.65	274.11 275.55	277.78 280.10	277.72
4	1	92.08 92.30	87.95 90.48	90.04 93.17	91.65 89.04	91.12 86.73	90.46
	2	151.51 152.51	151.84 151.67	149.96 151.76	152.54 157.32	154.63 154.36	152.81
	3	151.58 150.86	153.14 153.51	153.34 150.90	153.42 150.90	153.89 149.92	152.10

Table 4.3. Running Time of the proposed IRWLS method (part 2)

p	Iteration	Time for Training (s)					
		Test 1	Test 2	Test 3	Test 4	Test 5	Average
		Test 6	Test 7	Test 8	Test 9	Test 10	
8	1	52.69	49.77	49.77	50.38	49.87	51.17
		51.26	53.23	54.57	50.06	51.70	
	2	87.61	87.61	83.03	81.02	84.81	83.67
		81.88	85.72	88.40	82.92	79.12	
	3	92.68	90.39	93.10	92.81	90.53	92.67
		91.69	95.25	93.66	93.70	92.89	
12	1	38.17	38.98	38.36	39.03	36.58	38.25
		40.13	37.99	37.97	36.58	38.74	
	2	64.18	61.43	58.70	64.70	61.40	61.83
		62.80	60.59	61.93	60.26	62.27	
	3	63.58	64.30	67.64	67.92	63.19	65.32
		64.34	63.63	66.80	65.22	66.61	
16	1	30.55	29.92	27.89	29.01	27.01	28.75
		29.78	29.03	28.77	28.81	26.77	
	2	48.26	47.41	48.00	47.05	48.44	47.74
		47.42	47.28	46.49	48.71	48.34	
	3	50.70	47.64	51.14	50.36	50.94	49.78
		49.78	49.26	49.35	50.30	48.33	

Table 4.4. Running Time of the MLlib Baseline Package

p	Time for Training (s)					
	Test 1 Test 6	Test 2 Test 7	Test 3 Test 8	Test 4 Test 9	Test 5 Test 10	Average
1	1467.82 1456.30	1464.28 1453.50	1445.59 1469.99	1449.43 1455.07	1465.07 1447.27	1457.43
2	790.22 772.01	763.49 779.96	774.11 798.91	783.66 782.59	790.74 776.02	781.17
4	499.95 502.93	506.05 512.12	506.83 502.52	510.33 497.85	501.54 522.18	506.23
8	320.84 320.23	320.83 322.00	323.18 333.37	314.90 321.83	317.54 321.40	321.61
12	245.73 240.04	242.86 245.61	242.81 239.84	237.81 238.01	243.98 238.82	241.55
16	196.87 195.72	198.38 197.13	196.00 187.82	195.26 195.32	197.93 197.66	195.81

To better show the statistics of the above results, box plots have been drawn on both the IRWLS method (first iteration) and the MLlib method, as shown in Figure 4.1 and 4.2.

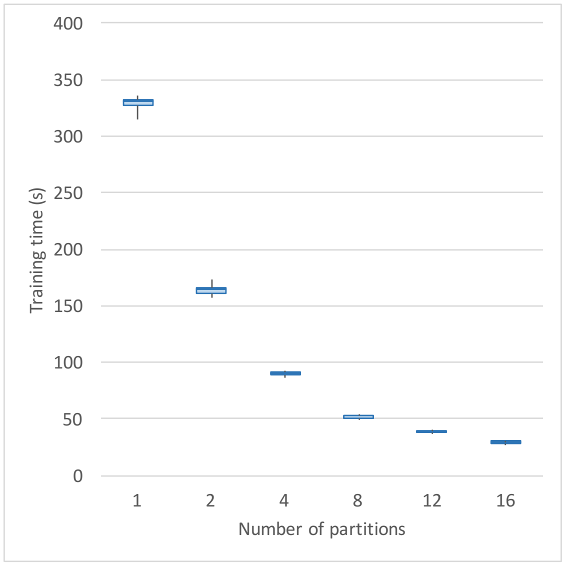

Figure 4.1. Box plot of training times using IRWLS (first iteration).

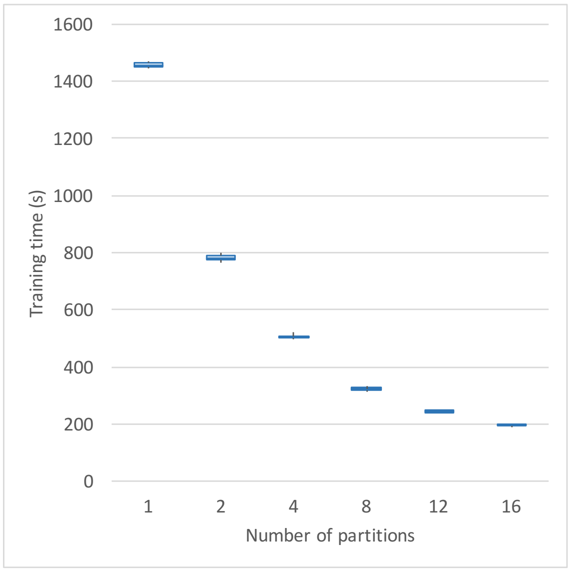

Figure 4.2. Box plot of training times using MLlib.

A significant running time advantage of the proposed IRWLS method can already be observed from the Table 4.2, 4.3, and 4.4. The proposed method finishes the slowest task (329.91 s for iteration 1) when $p = 1$ only roughly 1.5 times slower than how long MLlib package spends with its full potential (195.81 s) when $p = 16$.

A better observation can be illustrated using a bar chart as shown in Figure 4.3, where it shows training time of the first three iterations of the proposed IRWLS method and MLlib over different values of p. Although the speeds of the both methods are enhanced due to the increasing level of parallelization, major differences occur in the term the training time, where the first iteration of the proposed IRWLS method outperforms the MLlib package by about three or four times, and not much less regarding the second and the third iteration.

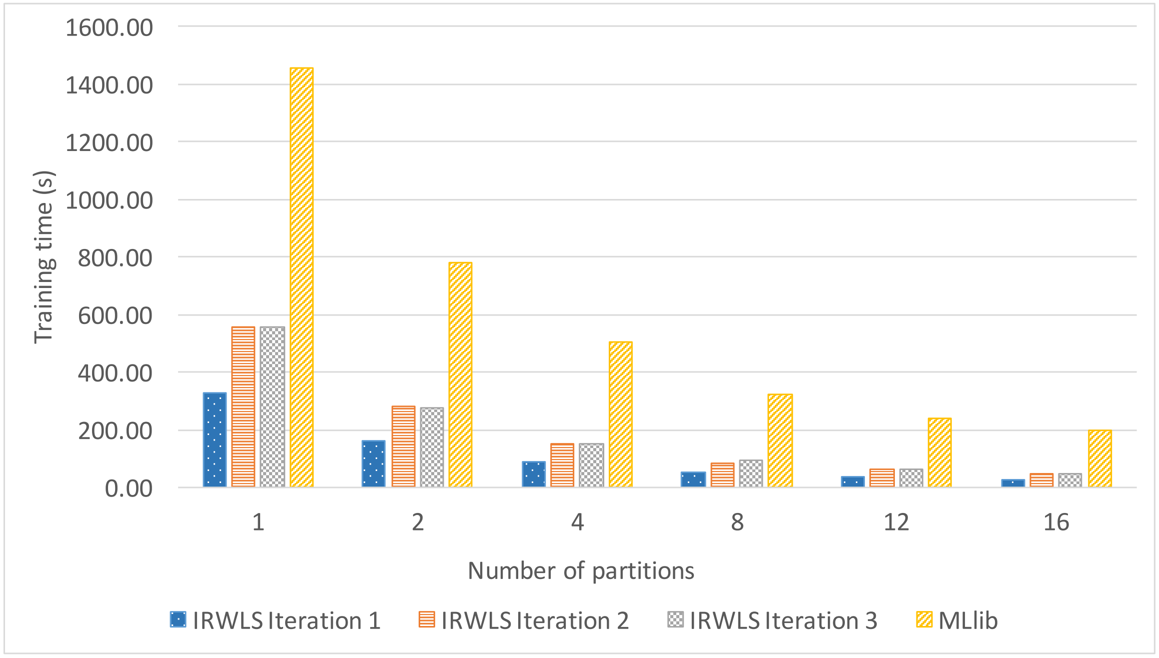

Figure 4.3. Training time summary using data of the year 1995.

For a more scientific analysis of parallelization performance, the system speed up, one of the commonly used metrics is calculated via Eq. 4.1, and its results are illustrated in Figure 4.4. As expected, the three curves of the proposed IRWLS method are much nearer than that of MLlib package to the theoretical linear line, which means the system speed up in perfect scenario without any additional communication cost, computing overhead, etc.. Another fact worth noticing is that the curve of IRWLS separates with that of MLlib at a very early stage, i.e., the difference already becomes obvious when p is small, and when communication costs should not be a large burden that undermines the performance of MLlib this much. The reason of this phenomenon will be analyzed in section 4.1.3 with more proof.

$$speed\ \ up = \frac{Time_{(serial)}}{Time_{(parallel)}} \tag{4.1}$$

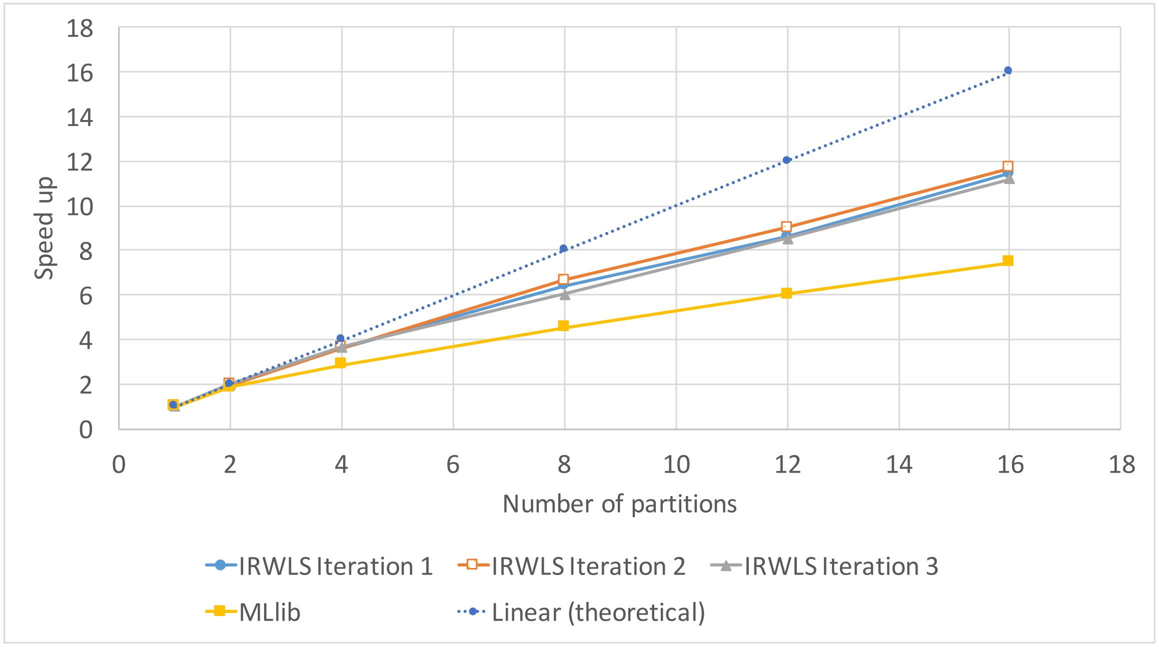

Figure 4.4. Speed up comparison using data of the year 1995.

To search for the reason behind the significant performance and speed up difference, this research demonstrates at an interesting finding that can be observed in the Spark Web UI - the garbage collection time. Garbage collection time refers to the time needed by Java Virtual Machine (JVM) when it has to evict existing objects for the purpose of making room for new ones. Basically the value of garbage collection time is proportional to the amount of Java objects created during a task, and it can be further raised by the increasing need of cleaning and refilling the memory for more space. This research has collected the garbage collection times of the same experiments and calculated the ratio shown in Eq. 4.2

$$GC\ \ ratio = \frac{Garbage\ \ collection\ \ time}{Total\ \ task\ \ time} \tag{4.2}$$

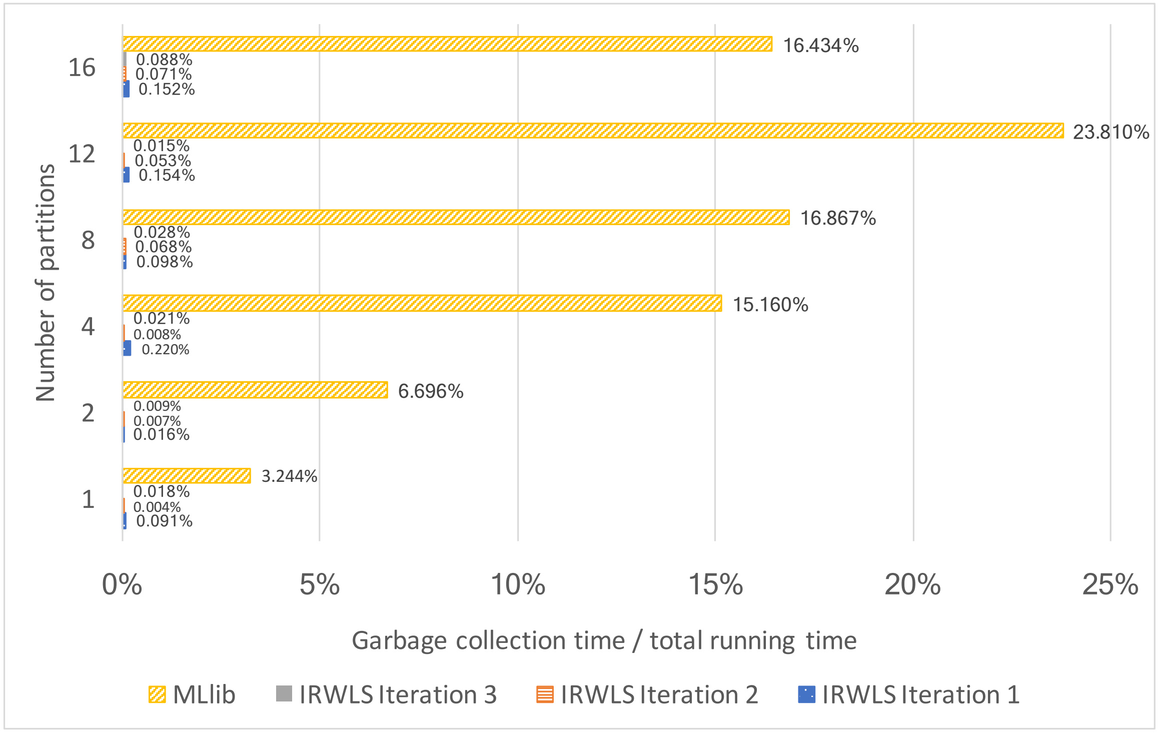

Figure 4.5. Garbage collection time ratio using data of the year 1995.

The GC ratio results are illustrated in Figure 4.5, where tremendous difference are found between the two methods. The MLlib package, in this observation, has a non-negligible GC ratio from about 0.03 to as high as about 0.21, whereas those of the proposed IRWLS method never exceed 0.003 and mostly lie under 0.001, which are basically ignorable when compared to those of MLlib.

4.1.3 Discussion

According to the results obtained by the current stage of experiments, the proposed IRWLS method manifests significant advantages against the baseline MLlib Logisic Regression package, in terms of both pure training time and parallelization speeding up performance.

The GC ratio illustrated in Figure 4.5 explains in one of the many possible ways why the proposed IRWLS method has its strength as manifested. As explained before, a large garbage collection time may be caused by a large amount of Java objects created during a task, i.e., the input size, or a frequent need to clean and refill the memory for making more room for new data. Since this experiment is conducted in away that the memory size much exceeds the input size, which is actually small, and all the tests are done with the same p for the both methods, the reason behind the tremendous difference of GC ratios can only be in the manner of algorithms themselves. As pointed out in section 3.3, the proposed method has a significant advantage because it only needs to do matrix algebra on the working sufficient statistics that has the largest size of $\mathscr{O}(m^2)$, where m is the number of features selected, and a better fact is that it does not need to reuse the calculated matrix once they are aggregated into the computation of the next row. This advantage in algorithm means the proposed method has a much lower need in memory space because the input data will be only scanned once row-by-row, and no further entries of processed data will be necessary, thus the garbage collection time is as low as ignorable. On the other hand, the traditional algorithm that MLlib uses has more complex mechanism that generates, stores, and communicates much larger mathematical results, which is the reason why it needs such a high GC ratio to clean and re-cache the memory for more space for new data coming into the computation.

And this need of a large Garbage Collection time also explains the speed up curves as shown in Figure 4.4. As noted in the previous section, the speed up of MLlib separates with that of IRWLS at a pretty early stage where communications cannot be the leading overhead yet. It can be assumed that the traditional algorithm behind MLlib package is not memory efficient enough to avoid addition cost caused by factors such us Garbage Collection, even when the granted memory size is much lager than the data input size. And this insufficiency further highlights the memory-wise advantage of the proposed IRWLS method.

Although the advantage in training time needs to be further examined due to the unknown number of iterations needed to achieve the expected accuracy and result convergence, it can already be concluded for now that if very few number iterations of the proposed IRWLS can guarantee an optimistic result, then the proposed algorithm is greatly promising from the angle of performance, because it has not been fully optimized yet in many aspects, as opposed to the already well-integrated MLlib package. And the next experiment will examine the accuracy level performance of the proposed method.

4.2 Accuracy and Result Convergence Examination over Iterations

As known to public, logistic regression needs multiple iterations to acquire a convergent β as the final weights for predictions, and so does the proposed method, one highlight of which is the ability to achieve an exact result with only a few number of iterations provided. Once the exact result can be acquired, an optimal accuracy of prediction comes naturally with it. In the previous stage of experiments, this research has demonstrated that the proposed method has a speed advantage when only a few of iterations are needed, so this current experiment will run it for 10 iterations to observe how many are necessary to achieve a convergent result and satisfying accuracy of prediction.

4.2.1 Testing parameters and input conditions

The dataset used in this stage is identical to the one used in the previous stage. The difference is that only running it using the number of partition $p = 16$, and set the number of iterations to 10. And the parameters used for this stage are listed in Table 4.5.

Table 4.5. Parameters for result convergence examination over iterations

Parameters	Value
Year of data	1995
Total size	1.54GB
Number of features	5
Features	`MONTH`, `AIRLINE_ID`, `FL_NUM`, `DEP_DEL15`, `DISTANCE`
Actual size being computed	122 MB for training + 34MB for testing
Number of records	4,177,444 for training + 1,044,361 for testing
Memory size	4 GB * 4 = 16 GB
Number of partitions	16
Number of iterations	10

4.2.2 Illustration of results

The results of running the proposed method for 10 iterations are shown in Table 4.6, where 10 sets of β value, indicating the calculated weights of `MONTH`, `AIRLINE_ID`, `FL_NUM`, `DEP_DEL15`, `DISTANCE`, and the intercept, are presented. The results show that though the value of β fluctuates in the first four iterations, the fluctuation decreases to none at the fifth iteration, after which the value of β converges and never changes again. This behaviors proves the ability of the proposed method for guaranteeing an exact result, and the number of iterations needed is incredibly low compared to the traditional method.

Table 4.6. Calculated β using the proposed IRWLS method over 10 iterations

Iteration	β (weights)					
	`MONTH`	`AIRLINE_ID`	`FL_NUM`	`DEP_DEL15`	`DISTANCE`	Intercept
1	-4.9283e-04	3.8551e-05	-6.2039e-06	1.3247e+00	6.8284e-05	-2.0421e+00
2	-1.6464e-03	1.2949e-04	-2.0423e-05	3.3464e+00	2.2402e-04	-4.7537e+00
3	-2.7872e-03	2.2325e-04	-3.3827e-05	3.9011e+00	3.6665e-04	-7.0157e+00
4	-3.1244e-03	2.5290e-04	-3.7445e-05	3.9776e+00	4.0314e-04	-7.6779e+00
5	-3.1369e-03	2.5409e-04	-3.7562e-05	3.9795e+00	4.0424e-04	-7.7037e+00
6	-3.1369e-03	2.5409e-04	-3.7562e-05	3.9795e+00	4.0424e-04	-7.7037e+00
7	-3.1369e-03	2.5409e-04	-3.7562e-05	3.9795e+00	4.0424e-04	-7.7037e+00
8	-3.1369e-03	2.5409e-04	-3.7562e-05	3.9795e+00	4.0424e-04	-7.7037e+00
9	-3.1369e-03	2.5409e-04	-3.7562e-05	3.9795e+00	4.0424e-04	-7.7037e+00
10	-3.1369e-03	2.5409e-04	-3.7562e-05	3.9795e+00	4.0424e-04	-7.7037e+00

And on the accuracy side, the results are even more promising. For the purpose of a more credible results, this research choses to show 12 digits for accuracy calculation, and the results of 10 iterations are listed in Table 4.7

Table 4.7. Accuracy using the proposed IRWLS method over 10 iterations

Iteration	Accuracy
1	0.903213543976
2	0.903214501499
3	0.903214501499
4	0.903214501499
5	0.903214501499
6	0.903214501499
7	0.903214501499
8	0.903214501499
9	0.903214501499
10	0.903214501499

According to the results, the proposed method can already achieve a steady accuracy of 0.903214501499 at as early as the second iteration, where the unchanged pattern means that the model already reaches the optimal accuracy. And another fact worth noticing is that even though the accuracy still gets better when moving from the first iteration to the second, but it only gets slightly improved on the sixth digit, which merely migrates from 3 to 4. So this accuracy sequence shows that at least for the purpose of predicting flight delay using the given dataset, the accuracy reaches its optimal at the second iteration, whereas that of the first iteration is already fair enough for real-world applications.

Then the test with the same parameters is conducted using the baseline MLlib logistic regression package, and results are listed in the same manner in Table 4.8 and 4.9. The results show that the acquired β is noticeably different with that of the proposed method, even though they are same on the exponent part. This finding means that unfortunately the MLlib method actually cannot provide an exact value of β as result, which again emphasizes the strength of the proposed method compared to it. As for the accuracy, MLlib gives a result of 0.903214501499 that is exactly the same with that of the proposed method, which further proves the accuracy acquired by IRWLS is indeed the optimal one.

Table 4.8. Calculated β using MLlib method

β (weights)					
`MONTH`	`AIRLINE_ID`	`FL_NUM`	`DEP_DEL15`	`DISTANCE`	Intercept
-4.6254e-03	-1.3393e-04	-3.6295e-05	3.9754e+00	4.1171e-04	-7.5707e+00

Table 4.9. Accuracy using MLlib method

Accuracy
0.903214501499

A closer look and a more thorough examination on convergence can be achieved by an additional experiment, where the training set of the year of 1995 has been shuffled for five times to calculate β using the both methods. And the results are shown in Table 4.10 and 4.11.

Table 4.10. Calculated β using the proposed IRWLS method on identical input shuffled for 5 times

shuffle	β (weights)					
	MONTH	AIRLINE_ID	FL_NUM	DEP_DEL15	DISTANCE	Intercept
1	-3.13695002e-03	2.54097311e-04	-3.75626882e-05	3.97953484e+00	4.04243698e-04	-7.70373001e+00
2	-3.13695002e-03	2.54097311e-04	-3.75626882e-05	3.97953484e+00	4.04243698e-04	-7.70373001e+00
3	-3.13695002e-03	2.54097311e-04	-3.75626882e-05	3.97953484e+00	4.04243698e-04	-7.70373001e+00
4	-3.13695002e-03	2.54097311e-04	-3.75626882e-05	3.97953484e+00	4.04243698e-04	-7.70373001e+00
5	-3.13695002e-03	2.54097311e-04	-3.75626882e-05	3.97953484e+00	4.04243698e-04	-7.70373001e+00

Table 4.11. Calculated β using the MLlib method on identical input shuffled for 5 times

shuffle	β (weights)					
	MONTH	AIRLINE_ID	FL_NUM	DEP_DEL15	DISTANCE	Intercept
1	-4.62540296e-03	-1.33939297e-04	-3.62950323e-05	3.97544934e+00	4.11710031e-04	-7.57071738e+00
2	-4.62539179e-03	-1.33939307e-04	-3.62949365e-05	3.97544882e+00	4.11710062e-04	-7.57071668e+00
3	-4.62540245e-03	-1.33939297e-04	-3.62950280e-05	3.97544932e+00	4.11710033e-04	-7.57071423e+00
4	-4.62540253e-03	-1.33939297e-04	-3.62950287e-05	3.97544932e+00	4.11710032e-04	-7.57071457e+00
5	-4.62546963e-03	-1.33939264e-04	-3.62950491e-05	3.97544935e+00	4.11709617e-04	-7.57071464e+00

As listed, the β results of the proposed method are all identical due to a steady convergence after the fifth iteration, while those of MLlib, though similar, still have observable fluctuations on the fifth digit and after. Since the training sets used are strictly identical except being shuffled using different random seed, results should be output as strictly the same if a method can achieve good convergence and an exact result, which is the case for the proposed method but not for MLlib.

4.2.3 Discussion

Result convergence and prediction accuracy have always been concerns of this research because only good results of those can prove the claim made for the proposed method being able to provide an exact result within limited number of iterations. And this stage of experiment has demonstrated promising results supporting this claim.

According to the recored results, the proposed method takes approximately five iterations to achieve a strictly converged value of β, which is an exact result due to the steadiness of it in the following iterations. And meanwhile the well-integrated and heavily optimized MLlib logistic regression package can only approximate the final result of β with much more time spent. This finding further supports the advantage of the proposed method over many traditional ones such as MLlib regarding the result convergence. Furthermore, the tests using the identical training set shuffled by five times show that only the proposed method can achieve perfect convergence and an exact result while MLlib cannot.

As for accuracy, it is also promising to observe that the proposed method can generally reach the strict optimal at the second iteration, whereas the accuracy of the first iteration only differs at the sixth digits, which means for real-world applications like predicting flight delay using the given dataset, running only one iteration of the proposed method can be regarded as sufficient. Hence the experiments after this stage will only use the first iteration for further performance analysis.

4.3 Performance Analysis with Different Settings of Data Memory Ratio

To manifest the advantage of loading data row-by-row when solving difficulties caused by large input data size compared to memory size. This experiment simulates the scenario when the input size is lower or higher than the assigned memory to observe the performance of both the proposed method and the baseline MLlib package. In this stage of experiments, two different ways of simulation are conducted: fixed memory size with various input size and fixed input size with various memory size. For the purposes above, this research introduces a concept of data memory ratio, which is calculated using Eq. 4.3.

$$Data\ memory\ ratio = \frac{Input\ size}{Memory\ size} \tag{4.3}$$

Performance tests will be conducted using different settings of data memory ratio in both ways of simulation, and results will be collected and analyzed.

Furthermore, the dataset finally expands from a single year to multiple, and also the number of features selected expands from the five used in previous experiments to the final 12 ones as listed in section 3.5.3.

4.3.1 Testing parameters and input conditions (fixed memory size)

The testing parameters of this experiment are listed in Table 4.12. Feature selection expands to the version of 12 ones.The dataset being used changes from the year of 1995 to the year of 1996 - 2001, plus the year of 2006 - 2015. The reason of not using data from the year of 2002 - 2005 is that there are much loss of records regarding some of the features selected in those years, which are voided for a more scientific result. And for testing, data of the first quarter of the year 2016 has been used. The maximum of the real input size is 5.1 GB in total, thus the memory size of this experiment is set to 512 MB per nodes and 2 GB in total to make the data memory ratio distributed more evenly on both side of the value of 1. Again, for the maximum potential of the proposed method, the partition number has been set to 16, and only one iteration will be tested.

There are 13 different values of data memory ratio used, corresponding to the usage of 13 years of data for training. So 13 performance tests of both the methods are conducted, whose results will be collected and analyzed.

Table 4.12. Parameters for data memory ratio experiment (fixed memory size)

Parameters	Value
Year of data	1996 - 2001, 2006 - 2015 (training); 2016 (testing)
Total size	24 GB
Number of features	12
Features	`MONTH`, `DAY_OF_MONTH`, `DAY_OF_WEEK` , `AIRLINE_ID`, `FL_NUM`, `ORIGIN_AIRPORT_ID`, `DEST_AIRPORT_ID`, `CRS_DEP_TIME`, `CRS_ARR_TIME`, `DEP_DEL15`, `TAXI_OUT`, `DISTANCE`
Actual size being computed	5.1 GB for training (maximum) + 64.6 MB for testing
Memory size	512 MB * 4 = 2 GB
Number of partitions	16
Number of iterations	1
Data memory ratio	0.15, 0.36, 0.60, 0.75, 0.95, 1.15, 1.35, 1.55, 1.80, 2.00, 2.20, 2.40, 2.55

4.3.2 Illustration of results (fixed memory size)

The training time results of the tests based on 13 different values of data memory ratio are illustrated in Figure 4.6. According to the chart, the training time of the proposed IRWLS method is generally lower than that of the baseline MLlib package, and the gap between two curves grows dramatically with increasing data memory ratio. To highlight where the input size equals to the assigned memory size, i.e., when data memory ratio equals to 1, this research has introduced a vertical reference dash line at $x = 1$. With the help of this reference line, analysis of both sides can be given clearly.

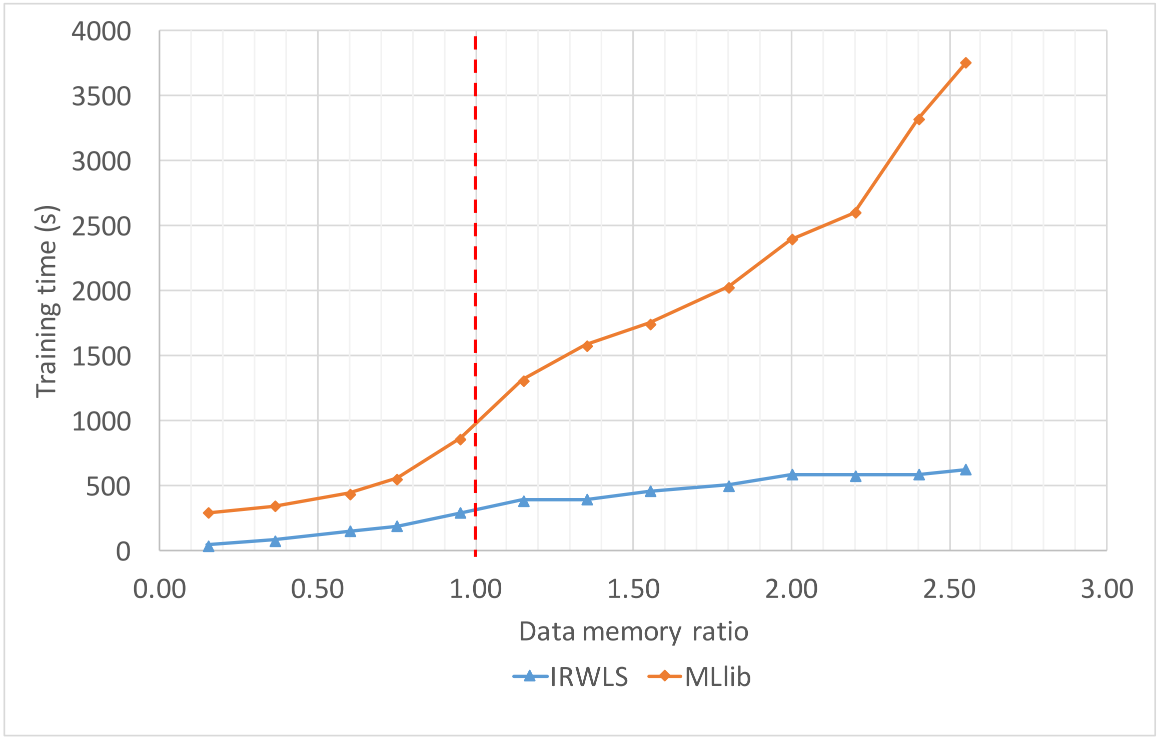

Figure 4.6. Training time of both methods with different data memory ratios (fixed memory size).

When data memory ratio is less than 1, at first both curves rise at a same speed linearly until when data memory ratio becomes nearer to 1, at which point the training time of MLlib suffers a dramatic jump across the reference line, making its curve no longer linear but concave upward. And after data memory ratio exceeds 1, the rising speed of MLlib curve does not drop down back to before, but drives the curve up continuously at a pace much faster than that of the proposed method, making the gap larger and larger. However on the side of the proposed method, whose curve only rises linearly before the $x = 1$ reference line, beyond which its rising speed becomes even lower than before, making the ascension noticeable but gentle.

With this observation, for now it can be assumed that with a data memory ratio growing nearer to 1, the MLlib method suffers a memory-wise burden that becomes larger and larger, which does not ease after passing the reference line, but even having sign of further growth since the curve becomes more concave with a data memory ratio larger than 2. But this burden on memory does not affect the proposed IRWLS method since passing the reference line does not boost the training time at all.

This research discovers another interesting findings in the Spark Web UI that seems to support this dramatical difference between the performances of the two methods - a metric called Input Size that accumulates over tasking process. After closer observations, this value of Input Size summarizes the amount of data transferring from a stage of computation to another, therefore the total Input Size can be regarded as a total data throughput over the whole process of training. The data throughput of the both method has been recored and illustrated in Figure 4.7

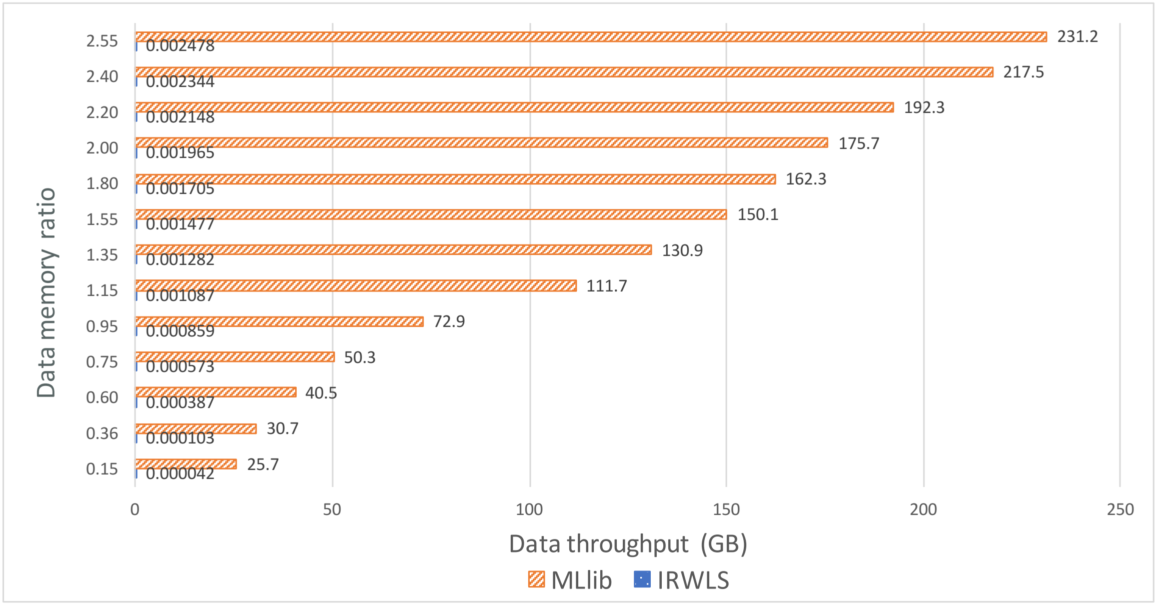

Figure 4.7. Data throughput of both methods with different data memory ratios (fixed memory size).

In the same way regarding the GC time ratio introduced in section 4.1, the MLlib method has a significantly large data throughput in unit of GB and growing fast corresponding to data memory ratio, while that of the proposed IRWLS method is between 0 - 3 MB, and nearly negligible compared to MLlib. As surprising as this finding is, it makes sense since the proposed method only scan the data once and has the small matrix of working sufficient statistics to transfer between stages, which only generates the data throughput less than several megabytes. However the traditional method used by MLlib has a much more complex mechanism that produces tremendous data throughput as high as 25 - 231 GB while the input data only has a maximum size of 5.1 GB in this experiment, which means a huge amount of data has been calculated back and forth, causing a heavy burden on both the computing power and memory resource.

To back up the relationship between the finding of data throughput and the performance difference, this research also illustrates the curve of data throughput over data memory ratio of the both methods, as shown in Figure 4.8 and 4.9.

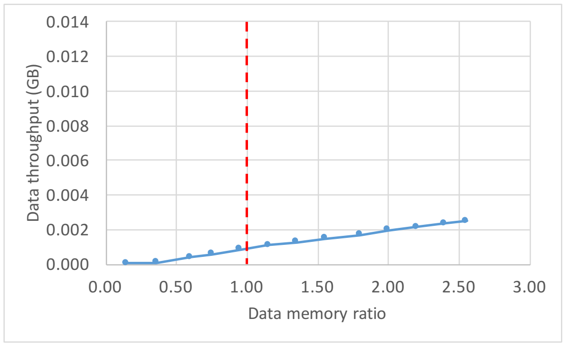

Figure 4.8. Data throughput using IRWLS with different data memory ratios (fixed memory size).

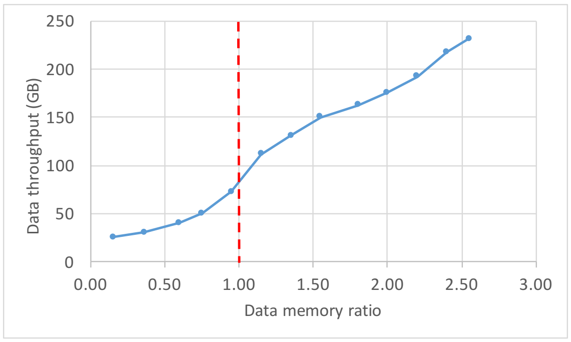

Figure 4.9. Data throughput using MLlib with different data memory ratios (fixed memory size).

It is not a surprise to find out that both curves share a similar shape to the training time curves shown in Figure 4.6. The growth rate of the data throughput of the proposed IRWLS method basically keeps as a constant, while that of the MLlib method suffers a sudden jump nearer and after the $x = 1$ reference line. Hence at this stage, it can be assumed that the huge difference between the data throughput of both methods is one of the reasons behind the significant performance gap.

4.3.3 Testing parameters and input conditions (fixed input size)

For a more thorough experiment, this research also conducts the data memory ratio test in a manner of using a fixed input size and various sizes of assigned memory to Spark nodes. The data of fixed size that being used is the one double merging the year of 1996 - 2001 and 2006 - 2015, which is 10.1 GB in total. The other parameters are similar except that the memory size varies from 2 GB to 16 GB to generate different and larger values of data memory ratios. The specific parameters are listed in Table 4.13

Table 4.13. Parameters for data memory ratio experiment (fixed input size)

Parameters	Value
Year of data	1996 - 2001, 2006 - 2015 (training, doubled); 2016 (testing)
Total size	48 GB
Number of features	12
Features	`MONTH`, `DAY_OF_MONTH`, `DAY_OF_WEEK` , `AIRLINE_ID`, `FL_NUM`, `ORIGIN_AIRPORT_ID`, `DEST_AIRPORT_ID`, `CRS_DEP_TIME`, `CRS_ARR_TIME`, `DEP_DEL15`, `TAXI_OUT`, `DISTANCE`
Actual size being computed	10.1 GB for training + 64.6 MB for testing
Memory size	16, 14, 12, 10, 8, 7, 6, 4, 3.5, 3, 2 GB
Number of partitions	16
Number of iterations	1
Data memory ratio	0.63, 0.72, 0.84, 1.01, 1.26, 1.44, 1.68, 2.53, 2.89, 3.37, 5.05

4.3.4 Illustration of results (fixed input size)

The tests are conducted in the same way as the previous ones except that the assigned memory size becomes the dependent variable at this stage resulting the different values of data memory ratio. The initial test is still the training time performance, whose result is illustrated in Figure 4.10.

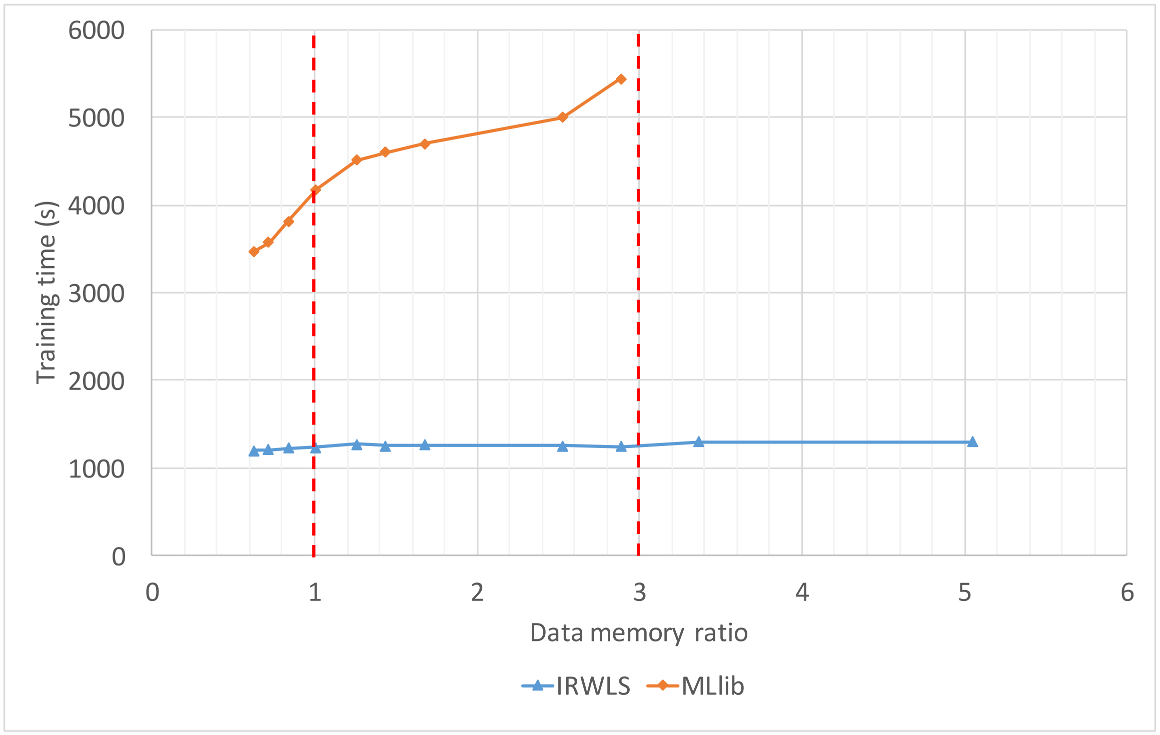

Figure 4.10. Training time of both methods with different data memory ratios (fixed input size).

The proposed IRWLS method still presents a promising performance that is relatively constant to the change of data memory ratio, which manifests the fact that the proposed method has the ability to keep its performance the same even with big data with limited memory resource. On the other hand, the MLlib package again suffers a sudden jump of training time when the data memory ratio comes near and exceeds 1. Furthermore, when the data memory ratio exceeds 3, the application running MLlib suffered a vital failure named `BlockFetchException`, which stopped the whole process, giving none result as output. It seems that MLlib has no ability to handle data that is too larger than the memory size and has to terminate the computation due to major data fetching failures.

In the same manner as before, this research also records the condition of data throughput and illustrates the results in Figure 4.11, 4.12, and 4.13.

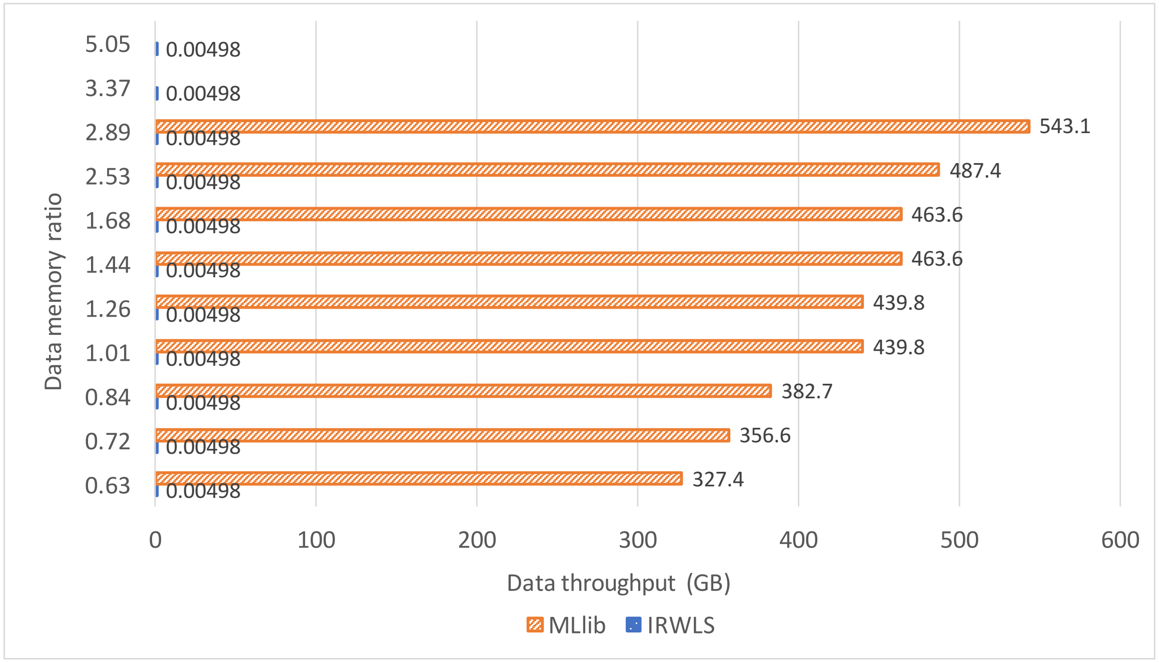

Figure 4.11. Data throughput of both methods with different data memory ratios (fixed input size).

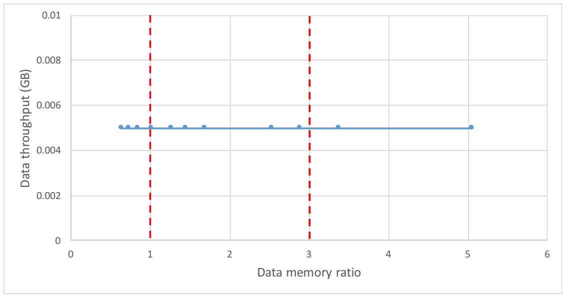

Figure 4.12. Data throughput using IRWLS with different data memory ratios (fixed input size).

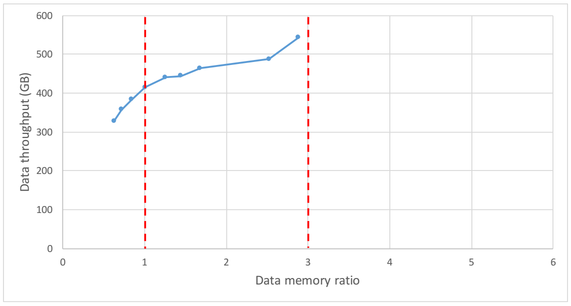

Figure 4.13. Data throughput using MLlib with different data memory ratios (fixed input size).

The curves of data throughput again support the performance difference in the ways of both the tremendous amount difference and similar shapes.

4.3.5 Discussion

This experiment summarizes the performances of both the proposed IRWLS method and the baseline MLlib package under situations when data memory ratio varies from less than 1 to larger than 1. With the previous experiments already showing the strength of the proposed method in manners of parallelization potential, result convergence, and speed of training, this experiments focuses more on how it performs with the changing of data memory ratio and aims to manifest the advantage of loading data row-by-row.

In the testing series of both ways of simulation (fixed memory size and fixed input size), the proposed method shows not only a fast speed in general, but also an “immunity” to changes of data memory ratio. However the baseline MLlib package always suffers a sudden jump in training time, tends to have a much higher cost when the data memory ratio grows near to 1 and beyond, and even encounter major memory fetching failures that terminate the who computation process, which demonstrates the limitation of the traditional method when handling dataset that has size similar or larger to the memory size.

This experiment also provides the finding of data throughput to support the observations of performance gap. The term of data throughput used here does not refer to the rate at which data transfers, but a accumulated sum of data transfered over Spark job stages. This criteria indicates the degree at which data being reused for calculation and additional intermediate results being generated, cached, and transfered. The difference in data throughput comes out surprisingly that an input size of 10.1 GB can cause more than 400 GB in data throughput using MLlib method, while merely less than 5 MB using the proposed method. Furthermore the curves of data throughput has similar shape with the performance curves, having same behaviors when data memory ratio comes near to and exceeds 1, which further proves the assumption that the proposed method benefits from the scanning row-by-row procedure and has a significant memory-wise advantage over the traditional method used by the baseline MLlib package.

The design of this experiment appropriately simulates and scales down the scenario when doing logistic regression training using real-world Big data on a limited memory resource. And the proposed method has again demonstrates its memory-wise strength and constant performance with various data memory ratios.

4.4 Summary

This chapter provides details of experiment results together with corresponding analysis and discussion. The advantage of the proposed method has been demonstrated in manners of parallelization performance, result convergence and accuracy, and performance with different data memory ratio. The next chapter provides the conclusion of this research.

CHAPTER 5. CONCLUSION

This research proposes a new fitting algorithm of logistic regression on IRWLS that utilizes the procedure of scanning data row-by-row and has the ability to acquire an exact result with only a few iterations. Furthermore, this research also realizes the distributed parallelization of the proposed method on Spark and conducts various experiments to manifest its memory-wise advantage over the traditional method used by Spark MLlib package. The following conclusions can be made out of the experiment results:

- The parallelization performance analysis demonstrates a much faster running speed and higher potential for parallelization in terms of the system speed up of the proposed method. And the analysis of Garbage Collection time further supports the performance enhancement, indicating the simplicity of the proposed method and advantage of scanning data row-by-row.

- The accuracy and β convergence examination over iterations shows that the proposed method takes approximately five iterations to achieve a strictly converged value of β, which is an exact result. And it can generally reach the optimal accuracy at the second iteration. And for real-world applications like predicting flight delay using the given dataset, running only one iteration of the proposed method can be already regarded as sufficient.

- Performance analysis with different settings of data memory ratio simulates the scenario when doing logistic regression training using real-world Big data on a limited memory resource. And the results show that the proposed method has constant performance to various data memory ratio and has extremely small data throughput between Spark job stages. This experiments further demonstrates the memory-wise strength of the proposed method

All the experiments show a tendency that the proposed IRWLS method is more parallelization friendly, memory efficient, performance-wise constant than the baseline MLlib package, and might also outperforms many other traditional methods.

The future work of this research includes more testing on large-scale data such as categorical data that is one-hot encoded, as well as further optimization in manners of data streaming, input vectorization, aggregation method, etc. It is promising to picture the fully optimized and integrated version of the proposed method solving the real-world Big Data machine learning problems with an even better performance of the next level.

REFERENCES

Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., ... others (2009). A view of the parallel computing landscape. *Communications of the ACM*, *52*(10), 56–67.

Catanzaro, B., Sundaram, N., & Keutzer, K. (2008). Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th international conference on machine learning* (pp. 104–111).

Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Olukotun, K., & Ng, A. Y. (2007). Map-reduce for machine learning on multicore. In *Advances in neural information processing systems* (pp. 281–288).

Dean, J., & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113.

Dreiseitl, S., & Ohno-Machado, L. (2002). Logistic regression and artificial neural network classification models: a methodology review. *Journal of biomedical informatics*, *35*(5), 352–359.

Genkin, A., Lewis, D. D., & Madigan, D. (2007). Large-scale bayesian logistic regression for text categorization. *Technometrics*, *49*(3), 291–304.

Géron, A. (2017). *Hands-on machine learning with scikit-learn and tensorflow: concepts, tools, and techniques to build intelligent systems.* OReilly Media, Sebastopol.

Lemeshow, S., & Hosmer Jr, D. W. (1982). A review of goodness of fit statistics for use in the development of logistic regression models. *American journal of epidemiology*, *115*(1), 92–106.

Liu, J., Chen, J., & Ye, J. (2009). Large-scale sparse logistic regression. In *Proceedings of the 15th acm sigkdd international conference on knowledge discovery and data mining* (pp. 547–556).

Mood, C. (2010). Logistic regression: Why we cannot do what we think we can do, and what we can do about it. *European sociological review*, *26*(1), 67–82.

Morgan, S. P., & Teachman, J. D. (1988). Logistic regression: Description, examples, and comparisons. *Journal of Marriage and Family*, *50*(4), 929–936.

Peng, C.-Y. J., Lee, K. L., & Ingersoll, G. M. (2002). An introduction to logistic regression analysis and reporting. *The journal of educational research*, *96*(1), 3–14.

Schein, A. I., & Ungar, L. H. (2007). Active learning for logistic regression: an evaluation. *Machine Learning*, *68*(3), 235–265.

Singh, S., Kubica, J., Larsen, S., & Sorokina, D. (2009). Parallel large scale feature selection for logistic regression. In *Proceedings of the 2009 siam international conference on data mining* (pp. 1172–1183).

Tu, J. V. (1996). Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology*, *49*(11), 1225–1231.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. *HotCloud*, *10*(10-10), 95.