Journal of The Korea Society of Computer and Information
Vol. 22 No. 4, pp. 9-16, April 2017
www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.04.009

# Performance Comparison of Logistic Regression Algorithms on RHadoop

Byung Ho Jung*, Dong Hoon Lim**

## Abstract

Machine learning has found widespread implementations and applications in many different domains in our life. Logistic regression is a type of classification in machine leaning, and is used widely in many fields, including medicine, economics, marketing and social sciences.

In this paper, we present the MapReduce implementation of three existing algorithms, this is, Gradient Descent algorithm, Cost Minimization algorithm and Newton-Raphson algorithm, for logistic regression on RHadoop that integrates R and Hadoop environment applicable to large scale data.

We compare the performance of these algorithms for estimation of logistic regression coefficients with real and simulated data sets. We also compare the performance of our RHadoop and RHIPE platforms.

The performance experiments showed that our Newton-Raphson algorithm when compared to Gradient Descent and Cost Minimization algorithms appeared to be better to all data tested, also showed that our RHadoop was better than RHIPE in real data, and was opposite in simulated data.

▸ Keyword : Big data, Hadoop, Logistic regression, R, RHadoop

## I. Introduction

빅데이터의 등장으로 데이터마이닝(data mining)의 중요성이 더욱 부각되고 있고 최근 들어 빅데이터와 관련하여 가장 주목을 받는 분야가 바로 머신러닝(machine learning)이다. 머신러닝은 인공지능의 한 분야로 기계가 스스로 학습하여 규칙을 생성하는 기술로 빅데이터의 핵심기술로 각광받고 있다.

빅데이터에서 로지스틱 회귀(logistic regression)는 머신러닝에서 분류 및 예측을 위한 모형으로 의료, 통신, 경제학, 마케팅 등 다양한 분야에서 폭넓게 사용되고 있다[1].

빅데이터 분석을 위해 기존의 전통적인 머신러닝 알고리즘을 사용하여 유용한 정보를 추출하는 것은 거의 불가능하다. 따라서 빅데이터에 적합한 분산처리 기반 머신러닝 알고리즘 개발이 필요하다[2].

Hadoop은 대용량 데이터를 분산처리 할 수 있는 자바 기반의 오픈소스 프레임 워크로서 분산파일 시스템인 HDFS(Hadoop Distributed Files System)에 데이터를 저장하고 분산처리 시스템인 MapReduce를 이용하여 데이터를 처리한다[3].

• First Author: Byung Ho Jung, Corresponding Author: Dong Hoon Lim
*Byung Ho Jung(root@korea.kr). Department of Information Statistics, Gyeongsang National University
**Dong Hoon Lim(dhlim@gnu.ac.kr). Department of Information Statistics, RINS, Gyeongsang National University
• Received: 2017. 01. 18, Revised: 2017. 01. 30, Accepted: 2017. 04. 13.
• This work was supported by the Gyeongsang National University Fund for Professors on Sabbatical Leave, 2016

R은 통계 계산, 데이터마이닝 그리고 데이터 가시화를 위한 오픈 소스 소프트웨어이다. R은 UNIX, 윈도우 그리고 MacOS 등 다양한 운영체제를 지원하며 Java, C, Fortran 등의 프로그래밍 언어와 연동이 가능하여 세계적으로 많은 사용자 들이 있다. 최근에 R은 구글, 페이스북, 야후, 아마존 등 많은 닷컴 기업에서 분석 플랫폼으로 사용되고 있다[4].

RHadoop은 R과 Hadoop의 통합환경으로 RHIPE[5, 6]와 같이 빅데이터 처리 및 분석을 위한 플랫폼으로 많이 사용되어 왔다[2,4,7-10]. 지금까지 RHadoop은 RHIPE에 비해 설치와 사용면에서 강점을 보인 반면에 처리속도면에서 약점을 보이는 것으로 나타났다[11].

본 논문에서는 RHadoop 플랫폼에서 로지스틱 회귀 추정을 위한 대표적인 알고리즘 즉, Gradient Descent 알고리즘, Cost Minimization 알고리즘과 Newton-Raphson 알고리즘[12]에 대해 MapReduce로 구현하고, 실제 데이터와 모의실험 데이터에서 이들 알고리즘 간의 성능을 비교하고자 한다. 또한, 두 플랫폼 RHadoop과 RHIPE간의 성능도 비교하고자 한다.

본 논문은 다음과 같이 구성되어 있다. 제 II 장에서는 관련연구로서 Hadoop과 RHadoop에 대해 간략하게 살펴보고, 제 III 장에서는 RHadoop 플랫폼에서 로지스틱 회귀 추정을 위한 알고리즘 간의 성능을 비교하고 두 플랫폼 RHadoop과 RHIPE간의 성능도 비교하고자 한다. 마지막 제 IV장에서 결론을 맺고 향후연구에 대해 논의하고자 한다.

## II. Related works

### 1. Hadoop

Hadoop의 HDFS는 물리적으로 네임 노드(name node)와 데이터 노드(data node)로 구성되어 있다. 네임노드는 파일의 디렉토리 구조, 권한과 같은 메타 정보를 저장하고, 실제 데이터는 여러 대의 데이터 노드에 분산되어 저장한다. MapReduce는 잡 트랙커(job tracker)와 테스크 트랙커(task tracker)로 구성되고 HDFS에 저장된 파일에서 데이터를 읽어서 가공하는 역할을 수행한다[13]. Fig. 1는 단어 세기(word count) 예를 가지고 MapReduce의 동작 과정을 보여주고 있다.

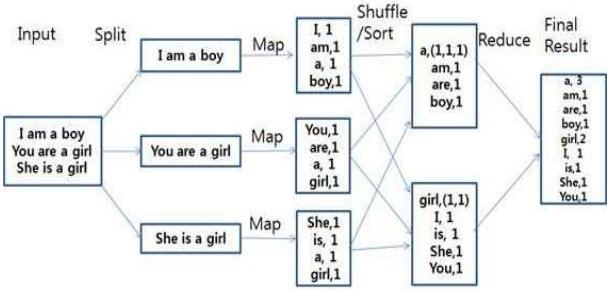


Fig. 1. MapReduce parallel processing

Fig. 1에서 보면 입력데이터는 블록 단위로 분할(Split)되고 각 블록은 <키, 값> 쌍으로 표현된다. 분할된 각 블럭에 대해 Map 함수 수행 후 다음 단계의 작업을 위해 <키, 값> 쌍으로 임시 저장된다. Shuffle / Sort 단계에서는 Map 함수의 결과를 키에 따라 섞고 다시 정렬한다. 각 키에 따라 값 들에 대해 병렬로 Reduce 함수를 적용한 후 <키, 값> 쌍으로 출력한다.

### 2. RHadoop

RHadoop은 3개의 주요한 R 패키지 즉, rmr, rhdfs, rhbase로 구성되어 있다. rmr은 R과 MapReduce 사이의 인터페이스 역할을 하는 패키지이다. 즉, R을 통해 Map과 Reduce를 호출할 수 있도록 해준다. rhdfs는 R에서 HDFS에 저장되어 있는 데이터를 이용할 수 있게 해주는 패키지이다. 이러한 패키지들은 각 노드들에 설치되어 이들 사이의 네트워크 통신을 통해 분산처리를 수행한다.

다음은 Fig. 1의 단어세기에 대한 예를 가지고 RHadoop에서 MapReduce 프로그램을 Map과 Reduce로 나누어 설명하고자 한다. 다음은 Map 스크립트를 나타내고 있다.

```
map <- function(NULL,v){
  words.list <- strsplit(v, split=" ")
  words <- unlist(words.list)
  keyval(words, 1)
}
```

위 스크립트에서 보듯이 map 함수는 HDFS에서 불러온 리스트 형태의 문장 데이터에 대해 공백을 기준으로 단어별로 분할하여 변수에 저장한다. 각 단어별로 구분된 데이터를 벡터 형태로 변환한 후, keyval 함수를 이용하여 <키, 값> 쌍의 집합을 생성하고 HDFS로 전송한다.

다음은 RHadoop에서 Reduce 스크립트를 나타내고 있다.

```
reduce <- function(k,v) {
  keyval(k, sum(v))
}
```

위 reduce 함수에서는 map 함수에서 생성된 <키, 값>에 대해 <키>를 기준으로 단어의 수인 <값>들에 대해 합을 연산하고 최종적으로 HDFS로 전송한다.

위에서 정의한 map 함수와 reduce 함수는 다음과 같이 mapreduce 함수를 통해 실행된다.

```
mapreduce( input, input.format, map, reduce, output, combine)
```

위 mapreduce 함수에서 input 인자, map 인자, reduce 인자, combine 사용여부 등 다양한 인자를 지정한다.

# III. Experimental Analysis of Logistic Regression Algorithms on RHadoop

## 1. Logistic Regression Algorithms

로지스틱 회귀모형에서 $k$개의 독립변수 $X_1, X_2, \ldots\ldots, X_k$에 대해 $Y$가 {0, 1}중 $Y=1$일 확률은 다음과 같이 표현할 수 있다.

$$P(Y=1|X_1,\ldots,X_k) = g(\sum_{i=0}^{k}\beta_i X_i) \quad (1)$$

여기서 $X_0=1$, $g(Z)=\frac{1}{1+e^{-Z}}$이다.

최우추정법(maximum likelihood estimation)에 의해 회귀계수 $\beta_0, \beta_1, \ldots\ldots, \beta_k$의 추정은 다음과 같다. 먼저, 우도함수(likelihood function)를 다음과 같이 정의한다.

$$L(\boldsymbol{\beta}) = P(Y_1, Y_2, \ldots, Y_n|X_1,\ldots,X_k) \quad (2)$$

따라서 로그-우도함수(log-likelihood)는 식 (2)에 로그를 취하면 다음과 같다.

$$\begin{aligned} l(\boldsymbol{\beta}) &= \log L(\boldsymbol{\beta}) \\ &= \sum_{i=1}^{n}\log g(Y_i Z_i) \end{aligned} \quad (3)$$

우리는 식 (3)에서 $l(\boldsymbol{\beta})$을 최대화하는 $\boldsymbol{\beta}$을 구하는 Gradient Descent 알고리즘, Cost Minimization 알고리즘 그리고 Newton-Raphson 알고리즘에 대해 설명하고자 한다.

1) Gradient Descent 알고리즘

Gradient Descent 알고리즘은 함수의 gradient에 비례하여 반복적인 방법으로 해를 찾는 방법으로 식 (3)에서 $\beta_k$에 대한 로그-우도 함수의 gradient를 다음과 같이 구한다.

$$\begin{aligned} \frac{\partial l(\boldsymbol{\beta})}{\partial \beta_k} &= \frac{\partial}{\partial \beta_k} log\, L(\boldsymbol{\beta}) \\ &= \sum_{i=1}^{n} Y_i X_{ik} g(-Y_i Z_i) \end{aligned}$$

따라서 최적해는 다음의 식에 의해 반복적으로 해를 찾는다.

$$\beta_k := \beta_k + \alpha \sum_{i=1}^{n} Y_i X_{ik} g(-Y_i Z_i)$$

여기서 $\alpha$는 학습률을 나타낸다. $\alpha$값이 너무 작으면 해에 대한 수렴속도가 늦고 $\alpha$값이 너무 크면 최적 해를 찾지 못하는 경우가 발생한다.

2) Cost Minimization 알고리즘

Cost Minimization 알고리즘은 로지스틱 회귀를 위한 Cost 함수를 정의하고 이 Cost 함수를 최소화하는 방법으로 먼저 $J(\boldsymbol{\beta})$을 로지스틱 회귀의 Cost 함수라 하고 다음과 같이 정의한다.

$$J(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} Cost(g(\boldsymbol{\beta}^t X_i), Y_i) \quad (4)$$

여기서 우리가 사용할 $Cost(g(\boldsymbol{\beta}^t X_i), Y_i)$는 다음과 같다.

$$Cost(g(\boldsymbol{\beta}^t X_i), Y_i) = \begin{cases} -\log(g(\boldsymbol{\beta}^t X_i)) & , \text{if } Y=1 \\ -\log(1-g(\boldsymbol{\beta}^t X_i)) & , \text{if } Y=0 \end{cases} \quad (5)$$

그리고 식 (4)은 식 (5)에 의해 다음과 같이 하나의 식으로 표현할 수 있다.

$$J(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n}[Y_i \log g(\boldsymbol{\beta}^t X_i) + (1-Y_i)\log(1-g(\boldsymbol{\beta}^t X_i))]$$

따라서 $J(\boldsymbol{\beta})$을 최소화하는 $\boldsymbol{\beta}$는 다음과 같이 구해진다.

$$\beta_k := \beta_k + \alpha \frac{\partial}{\partial \beta_k} J(\boldsymbol{\beta})$$

3) Newton-Raphson 알고리즘

Newton-Raphson 알고리즘은 학습률 $\alpha$가 필요없는 방법으로 먼저 $\beta_k$에 대한 로그-우도 함수의 gradient에 대해 $\beta_j$에 대한 로그-우도 함수의 2차 gradient는 다음과 같다.

$$\begin{aligned} \frac{\partial^2 l(\boldsymbol{\beta})}{\partial \beta_j \partial \beta_k} &= \frac{\partial}{\partial \beta_j}\sum_{i=1}^{n} Y_i X_{ik} g(-Y_i Z_i) \\ &= -\sum_{i=1}^{n} X_{ij} X_{ik} g(Y_i Z_i) g(-Y_i Z_i) \end{aligned} \quad (6)$$

$H(\boldsymbol{\beta})$를 로그-우도 함수의 2차 gradient 혹은 Hessian 행렬이라 할 때 식 (6)은 $H(\boldsymbol{\beta})$의 $j$번째 행과 $k$번째 열의 원소를 나타낸다. 따라서 Newton-Raphson 알고리즘은 $H(\boldsymbol{\beta})$를 사용하여 다음과 같이 표현할 수 있다.

$$\beta_k := \beta_k - H(\boldsymbol{\beta})^{-1}\sum_{i=1}^{n} Y_i X_{ik} g(-Y_i Z_i)$$

Newton-Raphson 알고리즘은 매번 반복과정에서 Hessian 행렬의 과중한 역행렬 계산을 요구하지만 수렴속도가 빠르다는 장점을 갖고 있다.

## 2. Experiment Environment

본 논문에서 사용된 실험환경은 6대의 개인용 PC를 사용하여 Fig. 2과 같은 클러스터를 구축하였다.

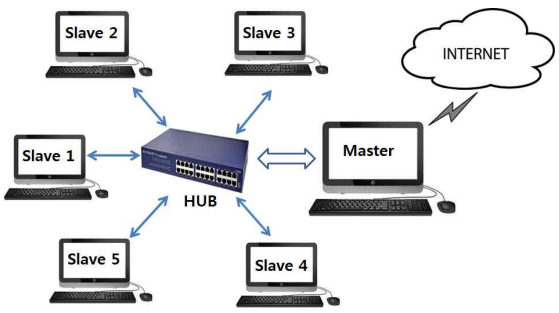


Fig. 2. Cluster configuration with 6 nodes

6대 중 1대 PC를 마스터(master) 노드, 나머지 5대 PC를 슬레이브(slaves) 노드로 설정하였으며 구축된 시스템의 하드웨어 및 소프트웨어 사양은 Table 1과 같다.

Table 1. Experiment environment

| | | | |
|---|---|---|---|
| Nodes | master | | 1 EA |
| | slaves | | 5 EA |
| Hardware specs | CPU | | Intel(R) Core2Duo E8300@2.83GHz |
| | RAM | master | 4GByte |
| | | slave | 4GByte |
| | NIC | | RealTeck 10/100Mbps |
| Software specs | OS | | Ubuntu 14.04 LTS |
| | Java | | 1.7.0 |
| | Hadoop | | 0.20.2 |
| | R | | 3.1.0 |
| | RHadoop | | rmr2(3.3.0), rhdfs(1.0.8) |
| Switch Hub | Cisco Catalyst 2960 (1G Ethernet) | | |

## 3. Experiment Data

실제 데이터는 2009년 ASA(American Standards Association:미국 규격 협회)에서 공개된 미국 항공기 운항과 관련된 데이터이다[14]. 이 항공기 데이터는 1987년부터 2008년까지 해마다 29개 변수에 대해 조사된 데이터이고 전체 데이터의 행은 123,534,970개이고 데이터의 크기는 12 GB정도이다. Wang et al.[15]과 같이 로지스틱 회귀분석을 위해 원래 데이터에서 5개의 변수만 고려하였으며 종속변수인 도착지연시간(ArrDelay)은 다음과 같이 얻었다.

$$\text{ArrDelay} = \begin{cases} 1, \ if \ \text{ArrDelay} > 15 \\ 0, \ otherwise \end{cases}$$

4개의 독립변수 중 출발시간(DepHour)은 원래 데이터의 출발시간(DepTime)에서 시(Hour)부분만 사용하였고, 비행거리(Distance)는 원래 데이터를 그대로 사용했다. 나머지 두 변수 밤(Night)과 주말(Weekend)은 아래와 같이 얻었다.

$$\text{Night} = \begin{cases} 0, \ if \ 0 \le \ DepHour \ \le \ 5 \text{ or } 20 \le \ \text{DepHour} \le \ 24 \\ 1, \ otherwise \end{cases}$$

$$\text{Weekend} = \begin{cases} 0, \ if \ 6 \le \ \text{DayOfWeek} \le \ 7 \\ 1, \ otherwise \end{cases}$$

여기서 DayOfWeek은 1(월요일)~7(일요일)값을 갖는다.

우리는 실제 데이터와 똑같이 4개의 독립변수들에 대해 정규난수를 발생시켜 모의실험을 실시하였다. 다음은 실험에 사용된 로지스틱 회귀모형이다.

$$g(z) = \frac{1}{1 + e^{-0.7 - 1.0X_1 - 1.3X_2 - 0.25X_3 - 0.05X_4}}$$

여기서 회귀계수들은 Rashid[16]에서 사용된 값들이다.

## 4. MapReduce Implementation

로지스틱 회귀 추정을 위한 Gradient Descent 알고리즘을 MapReduce로 구현하기 위한 의사코드(pseudo-code)는 Table 2와 같다.

Table 2에서 보는 바와 같이, 알고리즘은 크게 3 부분으로 나눈다. Map 정의 부분, Reduce 정의 부분, 그리고 반복 실행을 통해 수렴여부를 조사하는 부분이다. 먼저 Map의 내부 구조를 보면, 입력된 전체 데이터를 여러 개의 Map으로 쪼개어 gradient를 계산한 후 Reduce로 전송되고 Reduce에서는 각 Map에서 연산된 결과 값을 합산한다. 이러한 반복과정을 통해 로지스틱 회귀계수를 추정한다.

Cost Minimization 알고리즘을 MapReduce의 의사코드로 나타내면 Table 3과 같다.

Table 2. Pseudo-code for Gradient Descent algorithm

REQUIRE
- *Input (key, value) where key = null, and value = objects*
- *A set of objects of* $\{y_j, x_{j1}, x_{j2}, ..., x_{jk}\}$ *in each mapper*
- *initial set of weights* $\beta = \{\beta_0, \beta_1, \beta_2, \ ... \ , \beta_k\}$

```
1.  map phase
2.  {
3.        M ← {y_j, x_j1, x_j2, ..., x_jk}
4.        X ← {x_j1, x_j2, ..., x_jk}
5.        Y ← {y_j}
6.        for all y_i ∈ Y do
7.            if y_i = 0 then y_i = −1
8.        end for
9.
delta ← Y_j · X_jk · (1/(1+exp(−Y_j · (β_k^t × X_jk))))
10.       outputlist ≪ (NULL, delta)
11. }
12. reduce phase
13. {
14.       outputlist ← outputlist from mappers
15.    for all delta ∈ outputlist
16.       sumofdelta += delta
17.    end for
18.       outputlist ≪ (NULL, sumofdelta)
19. }
20. Repeat until convergence(ε = 10^−3)
21. {
22.       delta ← excute MapReduce
23.       β^(t+1) = β^(t) + α · delta
24. }
```

Table 3. Pseudo-code for Cost Minimization algorithm

REQUIRE
· *Input (key, value) where key = null, and value = objects*
· *A set of objects of* $\{y_j, x_{j1}, x_{j2}, ..., x_{jk}\}$ *in each mapper*
· *initial set of weights* $\beta = \{\beta_0, \beta_1, \beta_2, \ldots, \beta_k\}$

**pre-map phase**
{
  $M \leftarrow \{y_j, x_{j1}, x_{j2}, ..., x_{jk}\}$
  $Y \leftarrow \{y_j\}$
  $n_i \leftarrow length\ of\ Y$
  $outputlist \ll (NULL, n_i)$
}
*excute pre-Map phase*
for all $i \in no.pre-map\ tasks$
  $N \mathrel{+}= n_i$
end for
**map phase**
{
  $M \leftarrow \{y_j, x_{j1}, x_{j2}, ..., x_{jk}\}$
  $X \leftarrow \{x_{j1}, x_{j2}, ..., x_{jk}\}$
  $Y \leftarrow \{y_j\}$

$delta \leftarrow 1/N \cdot ((1/(1+\exp(\beta_k^t \times X_{jk}))) - Y_j) \cdot X_{jk}$
  $outputlist \ll (NULL, delta)$
}
**reduce phase**
{
  $outputlist \leftarrow outputlist\ from\ mappers$

  for all $delta \in outputlist$
    $sumofdelta \mathrel{+}= delta$
  end for
  $outputlist \ll (NULL, sumofdelta)$
}
**Repeat until convergence** $(\epsilon = 10^{-3})$
{
  $delta \leftarrow excute\ MapReduce$
  $\beta^{(t+1)} = \beta^{(t)} - \alpha \cdot delta$
}

Table 3에서 보듯이, Cost Minimization 알고리즘의 기본 골격은 Gradient Descent 알고리즘과 유사하다. Cost Minimization 알고리즘을 사용하기 위해서는 전체 행에 대한 개수가 필요하다. 따라서 MapReduce를 이용한 전체 행의 개수를 구하는 전처리 과정이 필요하다.

Newton-Raphson 알고리즘에 대한 MapReduce의 의사코드는 Table 4와 같다.

Table 4. Pseudo-code for Newton-Raphson algorithm

REQUIRE
· *Input (key, value) where key = null, and value = objects*
· *A set of objects of* $\{y_j, x_{j1}, x_{j2}, ..., x_{jk}\}$ *in each mapper*
· *initial set of weights* $\beta = \{\beta_0, \beta_1, \beta_2, \ldots, \beta_k\}$

**map phase**
{
  $M \leftarrow \{y_j, x_{j1}, x_{j2}, ..., x_{jk}\}$
  $X \leftarrow \{x_{j1}, x_{j2}, ..., x_{jk}\}$
  $Y \leftarrow \{y_j\}$
  $for\ all\ y_i \in Y\ do$
    $if\ y_i = 0\ then\ y_i = -1$
  $end\ for$
  $Hessian \leftarrow$

$X_{jk}^t \cdot 1/(1+\exp(-Y_j \cdot (X_{jk} \times \beta_k^t))) \cdot 1/(1+\exp(Y_j \cdot$

$delta \leftarrow Y_j \cdot X_{jk}^t \cdot 1/(1+\exp(Y_j \cdot (X_{jk} \times \beta_k^t)))$
  $outputlist \ll (NULL, (Hessian, delta))$
}
**reduce phase**
  $outputlist \leftarrow outputlist\ from\ mappers$
  for all $HessianDelta \in outputlist$
    $sumofHessianDelta \mathrel{+}= HessianDelta$
  end for

$outputlist \ll (NULL, sumofHessianDelta)$
}
**repeat until convergence** $(\epsilon = 10^{-3})$
{
  $resultofMR \leftarrow excute\ MapReduce$
  $Hessian, delta \leftarrow split\ resultofMR$
  $\beta^{(t+1)} = \beta^{(t)} - Hessian^{-1} \times delta$
}

Table 4에서 보듯이, 각 Map에서 Hessian 행렬과 gradient를 계산 한 후 그 결과를 하나로 묶어 Reduce로 전송한다. Reduce에서는 각 Map의 결과를 합산한다. 이러한 반복 과정을 통해 추정된 회귀계수를 얻는다.

## 5. Performance Comparison

로지스틱 회귀 추정을 위한 알고리즘의 성능은 회귀계수 추정하는데 걸린 시간과 추정된 회귀계수의 정확도를 가지고 비교하고자 한다.

Table 5은 실제 데이터에서 데이터 크기에 따른 알고리즘별 계산시간을 나타낸다.

Table 5. Comparisons of three algorithms with running time on real data

| Data Size | Gradient Descent | | Cost Minimization | | Newton-Raphson |
|---|---|---|---|---|---|
| | $\alpha$ | time | $\alpha$ | time | |
| 50 MB | 1.00e-6 | 506.96 | 4.38 | 436.02 | 273.01 |
| 100 MB | 5.000e-7 | 776.26 | 4.38 | 676.78 | 390.44 |
| 200 MB | 2.50e-7 | 887.31 | 4.38 | 745.98 | 439.28 |
| 300 MB | 1.25e-7 | 1321.09 | 4.38 | 773.22 | 442.63 |
| 500 MB | 1.00e-7 | 1092.29 | 4.38 | 942.20 | 512.16 |
| 1.00 GB | 4.36e-8 | 2490.92 | 4.38 | 1771.33 | 915.87 |
| 1.45 GB | 3.13e-8 | 3281.12 | 4.38 | 2494.40 | 1306.92 |
| 2.90 GB | 1.88e-8 | 4211.77 | 4.38 | 4156.46 | 2085.38 |
| 5.80 GB | 9.38e-9 | 8198.67 | 4.38 | 8096.13 | 3851.91 |
| 11.60 GB | 4.38e-9 | 17561.84 | 4.38 | 15214.70 | 7250.91 |

Table 5에서 보듯이 모든 데이터 크기에 대해 Newton-Raphson 알고리즘이 가장 빠르고, 다음으로 Cost Minimization, Gradient Descent 알고리즘 순으로 빠른 순서를 보였다.

Table 6은 glm 함수와 비교하여 알고리즘들의 추정된 회귀계수의 정확도를 나타낸 표이다.

Table 6. Comparisons of three algorithms to glm function with real data for estimated logistic regression equations

| Data Size | Algorithms | Estimated Logistic Regression Equations $g(\hat{Z}) = 1/(1+e^{-\hat{Z}})$ |
|---|---|---|
| 50 MB | glm | $\hat{Z}$ = -1.3780+0.0222$X_1$+0.1113$X_2$ +0.1967$X_3$-0.0735$X_4$ |
| | Gradient Descent | $\hat{Z}$ = -1.3774+0.0222$X_1$+0.1111$X_2$ +0.1965$X_3$-0.0732$X_4$ |
| | Cost Minimization | $\hat{Z}$ = -1.3775+0.0222$X_1$+0.1112$X_2$ +0.1966$X_3$-0.0732$X_4$ |
| | Newton-Raphson | $\hat{Z}$ = -1.3780+0.0222$X_1$+0.1113$X_2$ +0.1968$X_3$-0.0735$X_4$ |

Table 6에서 보듯이 Newton-Raphson 알고리즘은 다른 두 개의 알고리즘보다 glm 함수와 비교했을 때 추정된 회귀계수에서 차이가 거의 나지 않는 것을 알 수 있다. 그러나 Gradient Descent와 Cost Minimization 알고리즘은 $\beta_0, \beta_2, \beta_3, \beta_4$ 등에서 Newton-Raphson 알고리즘보다 차이가 큼을 보였다.

Table 7은 모의실험 데이터에서 데이터 크기에 따른 알고리즘별 계산시간을 나타낸다.

Table 7. Comparisons of three algorithms with running time on simulated data

| Data Size | Gradient Descent | | Cost Minimization | | Newton-Raphson |
|---|---|---|---|---|---|
| | $\alpha$ | time | $\alpha$ | time | |
| 50 MB | 1.25e-5 | 319.69 | 9.38 | 343.14 | 370.79 |
| 100 MB | 6.25e-6 | 411.15 | 9.38 | 439.69 | 467.19 |
| 200 MB | 3.13e-6 | 441.43 | 9.38 | 476.94 | 476.34 |
| 300 MB | 1.88e-6 | 436.74 | 9.38 | 475.66 | 503.05 |
| 500 MB | 1.25e-6 | 483.24 | 9.38 | 526.41 | 548.56 |
| 1.00 GB | 6.25e-7 | 847.99 | 9.38 | 934.63 | 934.80 |
| 1.45 GB | 3.75e-7 | 1164.47 | 9.38 | 1280.24 | 1257.91 |
| 2.90 GB | 1.88e-7 | 1846.85 | 9.38 | 2054.45 | 1991.98 |
| 5.80 GB | 9.38e-8 | 3515.02 | 9.38 | 3852.47 | 3793.39 |
| 11.60 GB | 5.00e-8 | 6567.56 | 9.38 | 7406.68 | 7101.42 |

Table 7에서는 Gradient Descent, Newton-Raphson, Cost Minimization 알고리즘 순으로 빠른 순서를 보였다. 그러나, 알고리즘의 정확한 성능 비교를 위해서는 처리속도 뿐 만 아니라 회귀계수의 정확한 추정도 중요하다. Table 8은 50 MB의 모의실험 데이터에서 glm 함수와 추정된 로지스틱 회귀 직선식을 나타낸다.

Table 8. Comparisons of three algorithms to glm function with simulated data for estimated logistic regression equations

| Data Size | Algorithms | Estimated Logistic Regression Equations $g(\hat{Z}) = 1/(1+e^{-\hat{Z}})$ |
|---|---|---|
| 50 MB | glm | $\hat{Z}$ = 0.7041+1.0045$X_1$+1.3036$X_2$ +0.2491$X_3$+0.0518$X_4$ |
| | Gradient Descent | $\hat{Z}$ = 0.7050+1.0057 $X_1$+1.3052$X_2$ +0.2494$X_3$+0.0518$X_4$ |
| | Cost Minimization | $\hat{Z}$ = 0.7051+1.0055$X_1$+1.3050$X_2$ +0.2494$X_3$+0.0518 8$X_4$ |
| | Newton-Raphson | $\hat{Z}$ = 0.7041+1.0045$X_1$+1.3037$X_2$ +0.2491$X_3$+0.0518$X_4$ |

Table 8에서 보듯이, Newton-Raphson 알고리즘은 glm 함수와 비교하였을 때 회귀계수의 차이가 발생하지 않으나 Gradient Descent와 Cost Minimization 알고리즘은 실제 데이터에 대한 Table 6과 비교했을 때 회귀계수 차이가 많이 나는 것을 알 수 있다. 이것은 Gradient Descent와 Cost Minimization 알고리즘은 적절하지 않은 학습률 사용으로 인해 정확한 회귀계수를 추정하지 못한 것으로 판단된다.

Table 9는 최적의 학습률 $\alpha$값을 사용하여 얻은 각 알고리즘별 처리 속도를 나타내는 표이다.

Table 9. Comparisons of three algorithms with simulation data in terms of computational time using optimal learning rate

| Data Size | Gradient Descent | | Cost Minimization | | Newton-Raphson |
|---|---|---|---|---|---|
| | $\alpha$ | time | $\alpha$ | time | |
| 50 MB | 7.7e-6 | 398.26 | 5.62 | 390.35 | 370.79 |

Table 9에서 보면 Table 7과는 달리 Newton-Raphson 알고리즘이 가장 빠르고 다음으로 Cost Minimization, Gradient Descent 알고리즘 순으로 빠른 결과를 보였다.

우리는 RHadoop 플랫폼의 성능을 평가하기 위해 R과 Hadoop의 또 하나의 통합환경인 RHIPE과 비교실험하였다. 여기서는 지면관계상 일부의 데이터 크기에서 Newton-Raphson 알고리즘에 대해서만 서술하고자 한다. Table 10은 실제 데이터에서 두 플랫폼의 계산시간을 나타낸 것이다.

Table 10. Comparison of RHadoop to RHIPE with real data using Newton-Raphson algorithm

| | 1.00 GB | 1.45 GB | 2.90 GB | 5.80 GB | 11.60 GB |
|---|---|---|---|---|---|
| RHadoop | 915.87 | 1306.92 | 2085.38 | 3851.91 | 7250.91 |
| RHIPE | 1048.42 | 1465.21 | 2408.63 | 4648.84 | 8817.24 |

Table 10에서 보면 RHadoop이 RHIPE보다 빠른 계산시간을 보였다. Table 11은 모의실험 데이터에서 두 플랫폼의 계산시간을 나타낸 것이다.

Table 11. Comparison of RHadoop to RHIPE with simulated data using Newton-Raphson algorithm

| | 1.00 GB | 1.45 GB | 2.90 GB | 5.80 GB | 11.60 GB |
|---|---|---|---|---|---|
| RHadoop | 934.80 | 1257.91 | 1991.98 | 3793.39 | 7101.42 |
| RHIPE | 786.24 | 1120.61 | 1687.67 | 3247.05 | 6221.43 |

Table 11에서 보면 Table 10의 결과와는 다르게 RHIPE이 RHadoop보다 빠른 계산시간을 보였다. 이것은 RHIPE의 표준 스트리밍 방식을 사용하지 않고 별도의 입력포맷 방식에 기인한다. RHadoop의 계산시간 문제는 노드추가로 극복할 수 있으나 Uskenbayeva et al. [11]에서 논의한 것처럼 RHIPE은 설치와 사용면에서 어려운 점을 갖고 있다.

## IV. Conclusions and Further Research

빅데이터 시대에 머신러닝의 중요성은 더욱 부각되고 있고 머신러닝에서 분류 방법의 하나인 로지스틱 회귀는 의료, 경제학, 마케팅 및 사회과학 전반에 걸쳐 널리 사용되고 있다.

오늘날 기하급수적으로 증가하는 대용량 데이터를 기존의 전통적인 플랫폼 상에서 처리 및 분석하는 것은 거의 불가능하다. 최근 R은 Hadoop의 분산처리 환경에서 빅데이터 분석도구로서 널리 사용되고 있다. R과 Hadoop의 통합환경으로 대표적 플랫폼이 RHadoop이다.

본 논문에서는 RHadoop 플랫폼에서 분산처리 기반의 로지스틱 회귀 추정을 위한 Gradient Descent, Cost Minimization 알고리즘과 Newton-Raphson 알고리즘을 MapReduce로 구현하고 실제 데이터와 모의실험 데이터에서 알고리즘 성능을 비교하였다.

3가지 알고리즘 성능비교 실험 결과, Gradient Descent 알고리즘은 적용이 용이하나 학습률에 크게 민감하게 반응하고 Cost Minimization은 학습률에는 민감하지 않으나 Gradient Descent 알고리즘과 마찬가지로 수렴성에 문제점을 갖고 있었다. 이에 반하여 Newton-Raphson 알고리즘은 사전 학습률이 필요없고 또한, 실제 데이터에서 Gradient Descent, Cost Minimization 알고리즘이 갖는 수렴성 문제도 없을 뿐만 아니라 모든 실험 데이터에 대해 좋은 성능을 보였다.

또한, 우리는 두 개의 플랫폼 RHadoop과 RHIPE의 성능을 비교한 결과 실제 데이터에서는 RHadoop이 빠르고 모의실험 데이터에서는 RHIPE이 더 빠른 것으로 나타났다. 물론, RHIPE은 설치와 사용면에서 RHadoop보다 어려움이 있다. RHadoop 플랫폼의 처리속도는 노드추가로 향상시킬 수는 있으나 한계를 갖고 있다. 따라서 오늘날 이를 해결하기 위해 인메모리 (in-memory) 기반인 Spark와 H2O 플랫폼에서 머신러닝에 관한 연구가 진행되고 있다. 향후 연구로서 이에 대한 연구가 진행되어 RHadoop 플랫폼과의 비교연구가 더욱 진척되길 희망한다.

## REFERENCES

[1] J. M. Hilbe, “Logistic Regression Models,” Chapman & Hall/CRC Press, 2009.

[2] B. Oancea, and R. M. Dragoescu, “Integration R and Hadoop for Big data analysis,” Romanian Statistical Review, No. 2. pp. 83-94, 2014.

[3] T. White, "Hadoop: The Definitive Guide," O'Reilly Media, Inc., Sebastopol, CA. 2012.

[4] V. Prajapati, "Big data analytics with R and Hadoop," Packt Publishing Ltd, Birmingham, UK. 2013.

[5] S. Guha, “Computing environment for the statistical analysis of large and complex data,” PhD thesis, Department of Statistics, Purdue University, West Lafayette, 2010.

[6] S. Guha, R. R., Hafen, J. Rounds, J. Xia, J. Li, B. Xi, W. S. Cleveland, “Large complex data: divide and recombine (D&R) with RHIPE,” Stat. 191, pp. 53-67, 2012.
[7] Y. Ko, and J. Kim, "Analysis of big data using Rhipe," Journal of the Korean Data & Information Science Society 24(5), pp. 975-987, 2013.
[8] B. H. Jung, J. E. Shin, D. H. Lim. "Rhipe Platform for Big Data Processing and Analysis," The Korean journal of applied statistics, 27(7), pp. 1171-1185, 2014.
[9] D. Harish, M.S. Anusha, Dr. K.V. Daya Sagar, "Big data analysis using Rhadoop," IJIRAE 4(2), 180-185, 2015.
[10] J. E. Shin, B. H. Jung, D. H. Lim. "Big data distributed processing system using RHadoop," Journal of the Korean Data & Information Science Society. 26(5), pp. 1155-1166, 2015.
[11] R. Uskenbayeva, A. Kuandykov, Y. I. Cho, T. Temirbolato a, S. Amanzholova, D. Kozhamzharova, “Integrating of data using the Hadoop and R”, Procedia Computer Science 56. pp.145 - 149, 2015.
[12] J. Wu, and S. Coggeshall. "Foundations of Predictive Analytics." Chapman and Hall/CRC. 2012.
[13] E. Sammer. "Hadoop Operations," O'Reilly Media, Inc., Sebastopol, CA. 2012
[14] ASA data expo. "Airline on-time performance," ASA Section on: Statistical Computing Statistical Graphics, http://stat-computing.org/dataexpo/2009/th e-data.html. 2009
[15] C. Wang, M. H. Chen, E. Schifano, J. Wu, J. Yan, "A survey of statistical methods and computing for Big Data." Cornell University Library. 2015.
[16] M. Rashid, “Inference on logistic regression models.” Doctor of philosophy thesis, Bowling Green State University. 2008.

## Authors

Dong Hoon Lim received the B.S., M.S. degrees in Computer and Statistics and Ph.D. degree in Statistics from Busan National University, Korea, in 1987, 1989 and 1993, respectively.
Dr. Lim is currently a Professor in the Department of Information and Statistics, Gyeongsang National University. He is interested in big data analysis.

Byung Ho Jung received the M.S. and Ph.D. degrees in Information and Statistics from Gyeongsang National University, Korea, in 2005 and 2017, respectively.
Dr. Jung is currently an official in the Information department, Gyeongsangnam-do Provincial Government. He is interested in big data analysis.