R Package *'gradDescent'* Modification

Xingwei Ji, Zack Liu, Liya Li, Dandi Peng

March 20, 2019

1 Introduction

R package *'gradDescent'* is an implementation of various learning algorithms based on Gradient Descent for dealing with regression tasks.

This package includes basic gradient descent algorithm **(GD)** and variants of gradient descent algorithm like Mini-Batch Gradient Descent (MBGD), Stochastic Gradient Descent (SGD), Stochastic Average Gradient (SAG), Momentum Gradient Descent (MGD), etc. Gradient Descent is a first order optimization algorithm to find a local minimum of an objective function by searching along the steepest descent direction.

The core computational expensive part of gradient descent algorithm is the iterations (for-loop). Here we choose the basic gradient descent algorithm function **(GD)** in package *'gradDescent'* as the modification target, and write C++ functions as a replacement to successfully increase the execution speed.

2 Algorithm

This **GD** function build a prediction model using Gradient Descent (GD) method.

In regression tasks, Gradient Descent Algorithm is an iterative method to minimize Residual Sum of Squares (RSS):

$$RSS = \sum_{i=1}^{n}(y^{(i)} - \beta^T x^{(i)})^2$$

where n is the sample size (number of observations), $y^{(i)}$ and $x^{(i)}$ correspond to the i^{th} observation, β is a vector of coefficients.

The basic gradient descent algorithm in package *'gradDescent'* is Batch gradient descent, and its procedure is:

1. Assign values randomly to initial parameters (coefficients β) following the distribution Uniform(0,1).
2. Calculate the Residual Sum of Squares (RSS) and update each elements β_j ($j = 0, 1, 2, ..., p$) of the coefficient β based on the rule: $\beta_j = \beta_j - a\frac{\partial RSS}{\partial \beta_j}$, where a is learning rate (constant).
3. Repeat step 1, 2 **t** (=100, 1000, ...) times.

The rule in step 2 yields:

$$\beta_j = \beta_j - a\frac{\partial RSS}{\partial \beta_j} = \beta_j - a\frac{\partial \sum_{i=1}^{n}(y^{(i)} - \beta^T x^{(i)})^2}{\partial \beta_j} = \beta_j - 2a\sum_{i=1}^{n}(\beta^T x^{(i)} - y^{(i)})x^{(i)}$$

The below **Figure 1**[1] is an example of gradient descent to get the minimum of a cost function on 2D graph.

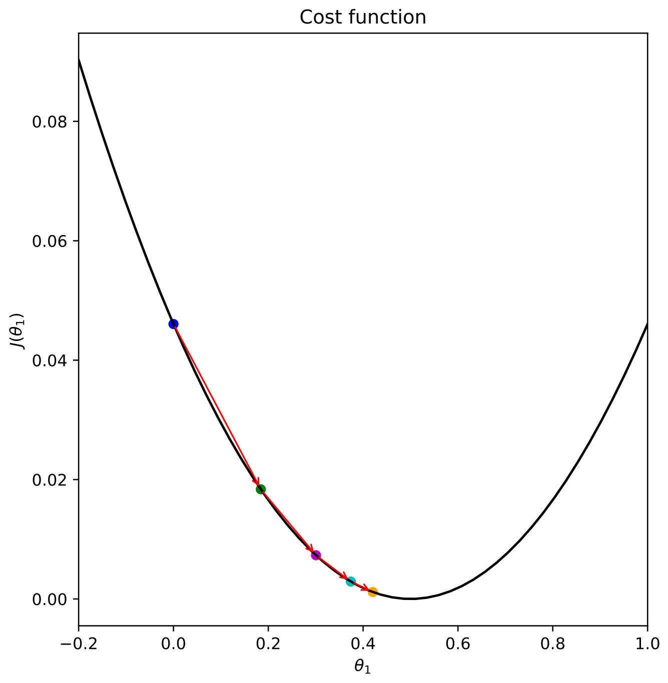


Figure 1: A Gradient Descent Example on 2D graph

[1]Hill, C. (June 5, 2016). Visualizing the gradient descent method. Retrieved from https://scipython.com/blog/visualizing-the-gradient-descent-method/

3 Modification with Rcpp

Since the third step in the gradient descent is the bottleneck, we decided to rewrite it in C++ by using Rcpp to speed up the **GD** function in `gradDescentR.Methods.R`. Rcpp provides many C++ classes to facilitate interfacing between R and C++, which made our rewriting process much easier than using the ordinary **.Call** function. The nested for-loop in the original GD function was replaced by

```
theta <- Cgd(maxIter,inputData,theta,rowLength,temporaryTheta,outputData,alpha)
```

where **Cgd** is a C++ function in file `GD.cpp`.

4 Modified Package Building Instruction[2]

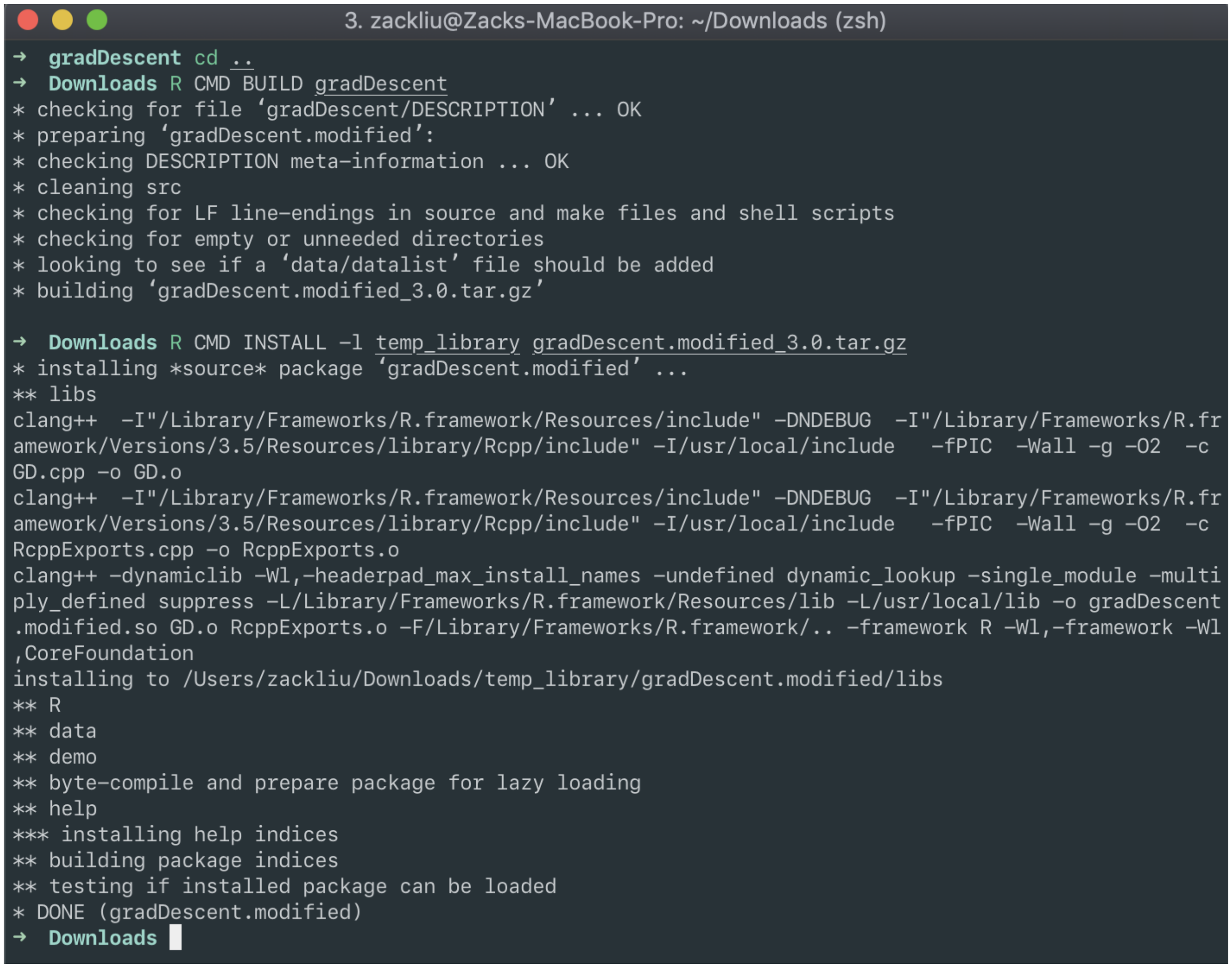
1. In terminal, change working directory to folder `gradDescent`
2. Rename the package name to `gradDescent.modified` in DESCRIPTION file
3. Open a R session
4. Run `devtools::use_rcpp()` , which creates a `src/` directory to hold our `.cpp` source file, and adds `"Rcpp"` to the `Linkingto` and `Import` fields in the DESCRIPTION file
5. add

   ```
   #’ @useDynLib gradDescent.modified, .registration = TRUE

   #’ @importFrom Rcpp sourceCpp

   NULL
   ```

   to the `gradDescentR.Methods.R` file and modified the for-loop in **GD** function as indicated in ***3 Modification with Rcpp*** to run partially in C++
6. Move our C++ source file `GD.cpp` into `src/`
7. Run `devtools::load_all()` , which loads and re-compiles the packages
8. Run `devtools:document()` , which updates `NAMESPACE` file
9. Run `cd ..` to go back to the parent directory
10. Run `R CMD BUILD gradDescent` to build the modified package, which generates `gradDescent.modified_3.0.tar.gz`
11. Run `R CMD INSTALL gradDescent.modified_3.0.tar.gz` to install

The final step of a successful installation is shown in **Figure 2**.

[2]Wickham, H. (2015, April). R Packages: Compiled code. Retrieved from http://r-pkgs.had.co.nz/src.htmlsrc



```
➜  gradDescent cd ..
➜  Downloads R CMD BUILD gradDescent
* checking for file ‘gradDescent/DESCRIPTION’ ... OK
* preparing ‘gradDescent.modified’:
* checking DESCRIPTION meta-information ... OK
* cleaning src
* checking for LF line-endings in source and make files and shell scripts
* checking for empty or unneeded directories
* looking to see if a ‘data/datalist’ file should be added
* building ‘gradDescent.modified_3.0.tar.gz’

➜  Downloads R CMD INSTALL -l temp_library gradDescent.modified_3.0.tar.gz
* installing *source* package ‘gradDescent.modified’ ...
** libs
clang++  -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG  -I"/Library/Frameworks/R.fr
amework/Versions/3.5/Resources/library/Rcpp/include" -I/usr/local/include   -fPIC  -Wall -g -O2  -c
GD.cpp -o GD.o
clang++  -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG  -I"/Library/Frameworks/R.fr
amework/Versions/3.5/Resources/library/Rcpp/include" -I/usr/local/include   -fPIC  -Wall -g -O2  -c
RcppExports.cpp -o RcppExports.o
clang++ -dynamiclib -Wl,-headerpad_max_install_names -undefined dynamic_lookup -single_module -multi
ply_defined suppress -L/Library/Frameworks/R.framework/Resources/lib -L/usr/local/lib -o gradDescent
.modified.so GD.o RcppExports.o -F/Library/Frameworks/R.framework/.. -framework R -Wl,-framework -Wl
,CoreFoundation
installing to /Users/zackliu/Downloads/temp_library/gradDescent.modified/libs
** R
** data
** demo
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (gradDescent.modified)
➜  Downloads
```

Figure 2: Installation

5 Test Result

Two test cases were applied to verify the modified R package. Both test cases use the same data set in **gradDescentRdata** from the **gradDescent** library.

The first test case focuses on a running time comparison between the original GD function and the modified GD function, where we used different number of rows of the data and fixed the number of iteration. Result comparison shown as **Figure 3**. The modified package executes properly as it shows in **Figure 4**.

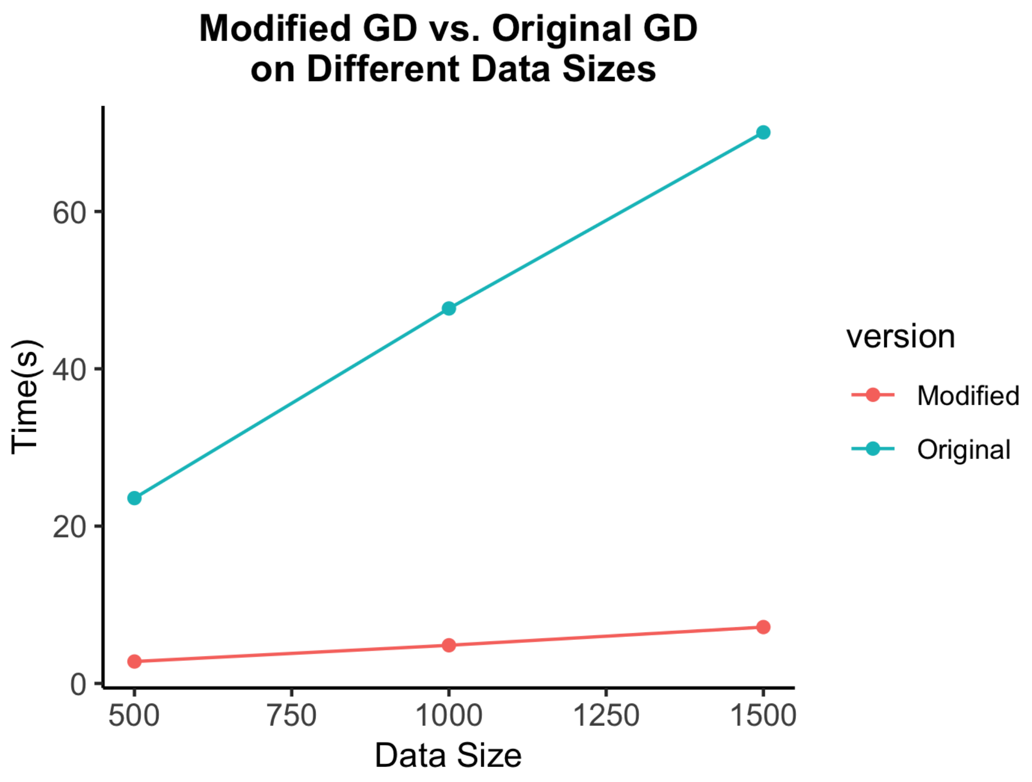
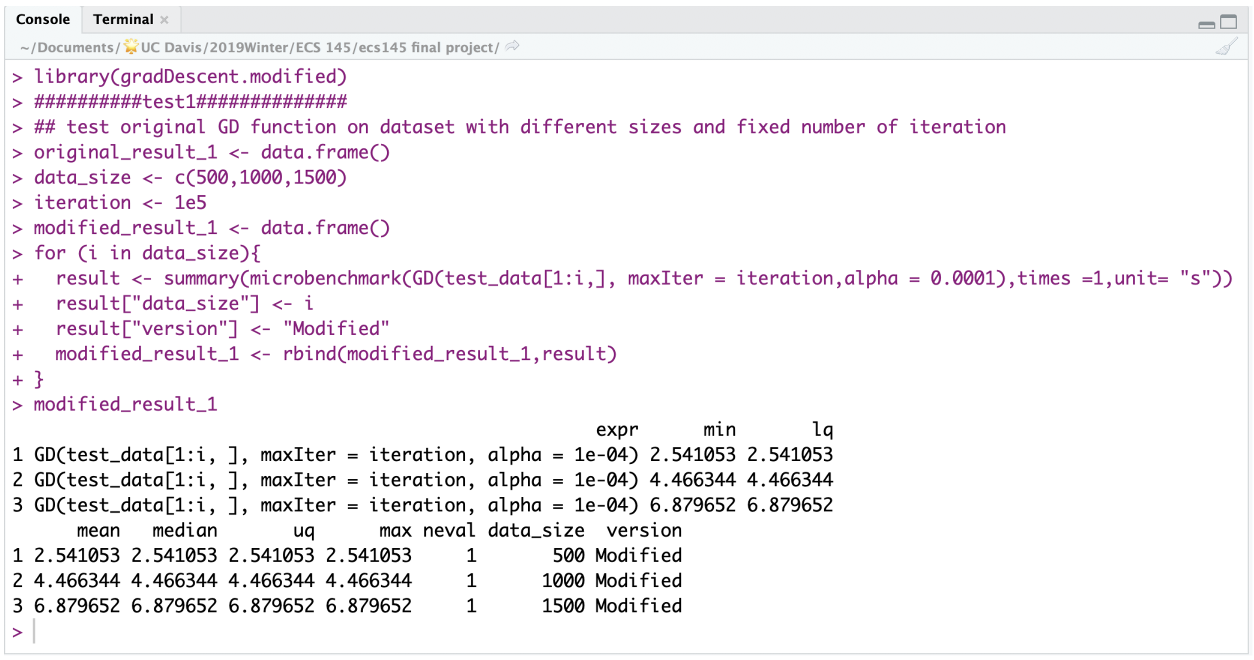


Figure 3: Results From Test Case 1


Console
Terminal
~/Documents/UC Davis/2019Winter/ECS 145/ecs145 final project/

```
> library(gradDescent.modified)
> #########test1##############
> ## test original GD function on dataset with different sizes and fixed number of iteration
> original_result_1 <- data.frame()
> data_size <- c(500,1000,1500)
> iteration <- 1e5
> modified_result_1 <- data.frame()
> for (i in data_size){
+   result <- summary(microbenchmark(GD(test_data[1:i,], maxIter = iteration,alpha = 0.0001),times =1,unit= "s"))
+   result["data_size"] <- i
+   result["version"] <- "Modified"
+   modified_result_1 <- rbind(modified_result_1,result)
+ }
> modified_result_1
                                                    expr      min       lq
1 GD(test_data[1:i, ], maxIter = iteration, alpha = 1e-04) 2.541053 2.541053
2 GD(test_data[1:i, ], maxIter = iteration, alpha = 1e-04) 4.466344 4.466344
3 GD(test_data[1:i, ], maxIter = iteration, alpha = 1e-04) 6.879652 6.879652
      mean   median       uq      max neval data_size  version
1 2.541053 2.541053 2.541053 2.541053     1       500 Modified
2 4.466344 4.466344 4.466344 4.466344     1      1000 Modified
3 6.879652 6.879652 6.879652 6.879652     1      1500 Modified
> 
```

Figure 4: Execution of Test Case 1

Meanwhile, the second test case focuses on a same procedure comparison yet with the used of fixed data size and different number of iteration. Results shown as **Figure 5**.

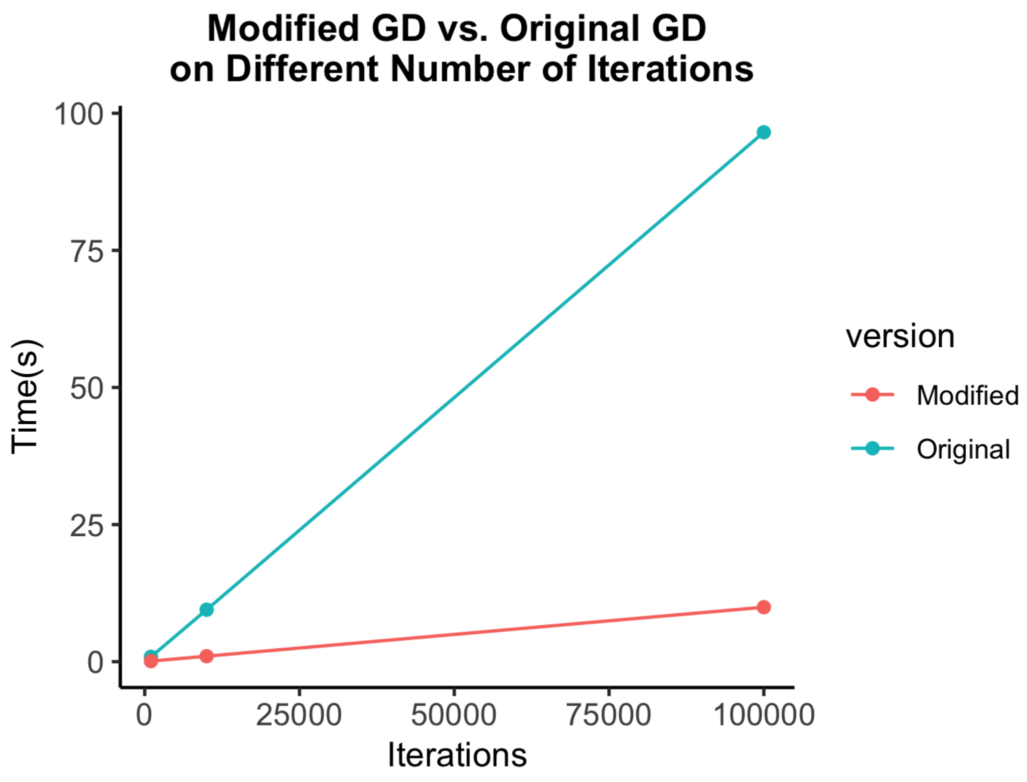


Figure 5: Results From Test Case 2

In summary, no matter applying for different data sizes or for iteration numbers, our modified GD function yields to much faster running time. On one hand, the larger data size or the larger iteration numbers, the faster running time our modified GD function results to than the original GD function. On the other hand, in the comparison on different iteration number, the range of running time increased increases obviously more when using the modified GD function.

6 Contribution

R package selection & modification discussion: All members

C++ code: Xingwei Ji

Modified Package Installation Instruction & test cases: Zack Liu & Dandi Peng

Report: Dandi Peng & Liya Li

Appendix

GD.cpp

```cpp
#include <Rcpp.h>
 using namespace Rcpp;

// [[Rcpp::export]]
NumericMatrix Cgd(int maxIter, NumericMatrix inputData,
   NumericMatrix theta, int rowLength, NumericMatrix
   temporaryTheta, NumericVector outputData,float alpha)
{
    for (int iteration = 0; iteration < maxIter; iteration++)
    {
        NumericMatrix tran_theta = transpose(theta);
        //inputData %*% t(theta)
        NumericVector y_hat(inputData.nrow());
        NumericVector temp(inputData.ncol());
        for (int i = 0; i < inputData.nrow();i++)
        {
            temp = inputData(i,_) * tran_theta;
            y_hat[i] = sum(temp);
        }
        NumericVector error = y_hat - outputData;
        for (int j = 0; j < theta.ncol(); j++)
        {
            NumericVector term = error * inputData(_,j);
            //calculate gradient
            float gradient = sum(term) / rowLength;
            //printf("gradient: %f\n",gradient);
            temporaryTheta(0,j) = theta(0,j) - (alpha *
               gradient);
        }
        theta = temporaryTheta;
    }
    return theta;
}
```

Running_time_test.R

```
library(microbenchmark)
library(ggplot2)
library(gradDescent)
data("gradDescentRData")

#data
test_data <- gradDescentRData[[3]]
data("gradDescentRData")

##########test1##############
## test original GD function on dataset with different sizes
   and fixed number of iteration
original_result_1 <- data.frame()
data_size <- c(500,1000,1500)
iteration <- 1e5
for (i in data_size){
  result <- summary(microbenchmark(GD(test_data[1:i,],
     maxIter = iteration,alpha = 0.0001),times =1,unit= "s"))
  result["data_size"] <- i
```

```r
  result["version"] <- "Original"
  original_result_1 <- rbind(original_result_1,result)
}
detach("package:gradDescent", unload=TRUE)

## test modified GD function on dataset with different sizes
   and fixed number of iteration for run of GD
library(gradDescent.modified)
modified_result_1 <- data.frame()
iteration <- 1e5
for (i in data_size){
  result <- summary(microbenchmark(GD(test_data[1:i,],
     maxIter = iteration,alpha = 0.0001),times =1,unit= "s"))
  result["data_size"] <- i
  result["version"] <- "Modified"
  modified_result_1 <- rbind(modified_result_1,result)
}
detach("package:gradDescent.modified", unload=TRUE)
#result1
test1<-rbind(original_result_1,modified_result_1)

##########test2##############
## test original GD function on dataset with same sizes and
   different number of iteration
library(gradDescent)
original_result_2 <- data.frame()
iteration <- c(1e3,1e4,1e5)
for (i in iteration){
  result <- summary(microbenchmark(GD(test_data, maxIter = i,
     alpha = 0.0001),times =1,unit= "s"))
  result["iterations"] <- i
  result["version"] <- "Original"
  original_result_2 <- rbind(original_result_2,result)
}
detach("package:gradDescent", unload=TRUE)

## test modified GD function on dataset with same sizes and
    different number of iteration
library(gradDescent.modified)
modified_result_2 <- data.frame()
iteration <- c(1e3,1e4,1e5)
for (i in iteration){
  result <- summary(microbenchmark(GD(test_data, maxIter = i,
     alpha = 0.0001),times =10,unit= "s"))
  result["iterations"] <- i
  result["version"] <- "Modified"
  modified_result_2 <- rbind(modified_result_2,result)
}
#result2
test2<-rbind(original_result_2,modified_result_2)
detach("package:gradDescent.modified", unload=TRUE)
```