Machine Learning, Spring 2021
Homework 3

Daniil Bulat, Jonas Josef Huwyler, Haochen Li,Giovanni Magagnin

3/18/2021

Exercise 1

Consider the Boston data set again, modeling *medv* in depdence of *lstat*. Fix the learning rage $eta = 0.000009$**. Set the initial values of the weights as** $(\beta_0, \beta_1) = (30, 0)$**.**

```
# Removing all variables/functions
rm(list = ls())

# Set the randam seed
set.seed(123)

# Learning rate (a smart guess)
eta = 0.000009

# Initial weights (a smart guess)
weights = c(30,0)

# Number of iterations (epochs) for the algorithm
NumIt = 20

# Data Preparation (Boston Data)
library(MASS)
# Keep only the features medv and lstat
Data = Boston[c("medv", "lstat")]
# Shuffle the data for (likely) better performance
Data = Data[sample(1:nrow(Data)),]
```

Question 1

Write a mini-batch gradient descent algorithm for batch size 32 with 20 epochs/iterations for the Boston data set.

```
# Question 1 -------------------------------------------------------------

# Number of batch size for the algorithm
BatSize = 32

# Number of batches for the Data
NumBat = floor(nrow(Data)/BatSize)+1

# Create lists to store results of each epoch
NumOfIt = rep(NA,NumIt)
NumOfRSS = rep(NA,NumIt)

# Set Start time of MBGD
MBGD.start = Sys.time()

# The mini-batch gradient descent algorithm
# Loops w.r.t. epochs (20 Iterations)
for (i in 1:NumIt){
  # Loops to update weights w.r.t. mini batches (16 batches) for specified epoch
  for (j in 1:NumBat){
    # Create Mini Batch regarding for index from 1 to 16
    if (j < NumBat){
      Data.Train = Data[((j-1)*BatSize+1):(j*BatSize),]
    } else {
      Data.Train = Data[((j-1)*BatSize+1):nrow(Data),]
    }

    # calculate stochastic descent algorithm for the specified mini batch
    # in specified epoch
    yhat = Data.Train$lstat*weights[2]+weights[1]
    error = Data.Train$medv - yhat
    weights[1] = weights[1] + 2*eta*sum(error)
    weights[2] = weights[2] + 2*eta*sum(error*Data.Train$lstat)
    # print(weights)
  }

  # Compute and store numbers of iterations and RSS for each epoch
  NumOfIt[i] = i
  yhat=Data$lstat*weights[2]+weights[1]
  error =  Data$medv  - yhat
  RSS = error%*%error
  NumOfRSS[i] = RSS
}

# Set end time of MBGD
MBGD.end = Sys.time()

# Calcualate error for the whole sample using the final weights
yhat=Data$lstat*weights[2]+weights[1]
error =  Data$medv  - yhat
```

```r
# Final output
sprintf("The final updated weights is (%.3f,%.3f)",weights[1],weights[2])
```

```
## [1] "The final updated weights is (30.157,-0.675)"
```

```r
RSS = error%*%error
sprintf("The corresponding RSS is %.3f",RSS)
```

```
## [1] "The corresponding RSS is 21847.045"
```

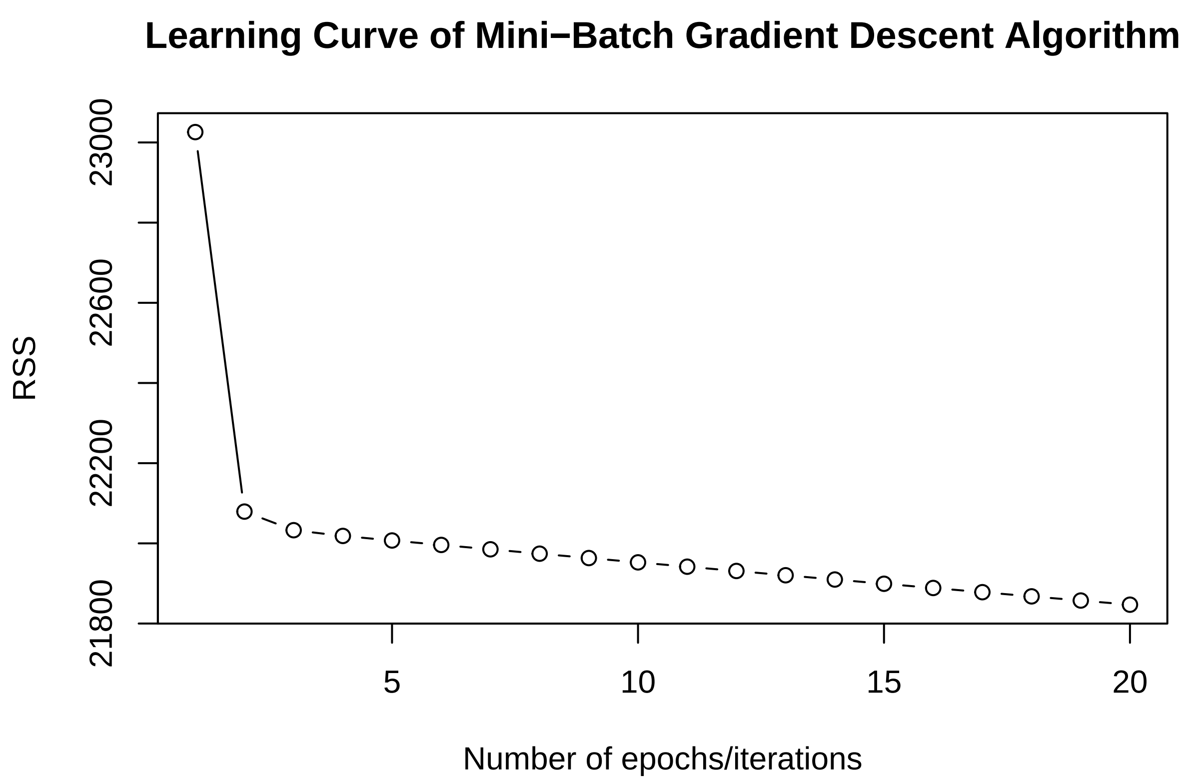
```r
RSE = sqrt(RSS/(nrow(Data)-2))
sprintf("The corresponding RSE is %.3f",RSE)
```

```
## [1] "The corresponding RSE is 6.584"
```

Question 2

Plot a learning curve for your mini-batch gradient descent algorithm, where the learning performance is measured by the RSS and the experience is given by the number of epochs/iterations, from 1 to 20.

```
# Question 2 -------------------------------------------------------------

#Plot the learning curve for the MBGD algo
plot(NumOfIt,NumOfRSS,
     type="b",
     main = "Learning Curve of Mini-Batch Gradient Descent Algorithm",
     xlab = "Number of epochs/iterations",
     ylab = "RSS")
```



Question 3

Plot a learning curve for the full gradient descent algorithm, where the learning Performance is measured by the RSS and the experience is given by the number of epochs/iterations, from 1 to 20.

```
# Question 3 ---------------------------------------------------------------

# Reset weights and lists to store
weights = c(30,0)
NumOfIt = rep(NA,NumIt)
NumOfRSS = rep(NA,NumIt)

# Set Start time of FGD
FGD.start = Sys.time()

# The full gradient descent algorithm (identical to the code example)
# Loops w.r.t. epochs (20 Iterations)
for (i in 1:NumIt){
  # The gradient descent algorithm
  yhat=Data$lstat*weights[2]+weights[1]
  error =  Data$medv  - yhat
  weights[1] = weights[1] + 2*eta*sum(error)
  weights[2] = weights[2] + 2*eta*(error%*%Data$lstat)
  # print(weights)

  # Compute and record numbers of iterations and RSS for each epoch
  NumOfIt[i] = i
  yhat=Data$lstat*weights[2]+weights[1]
  error =  Data$medv  - yhat
  RSS = error%*%error
  NumOfRSS[i] = RSS
}

# Set end time of FGD
FGD.end = Sys.time()

# Calcualate error for the whole sample using the final weights
yhat=Data$lstat*weights[2]+weights[1]
error =  Data$medv  - yhat

# Final output
sprintf("The final updated weights is (%.3f,%.3f)",weights[1],weights[2])
```

```
## [1] "The final updated weights is (30.166,-0.533)"
```

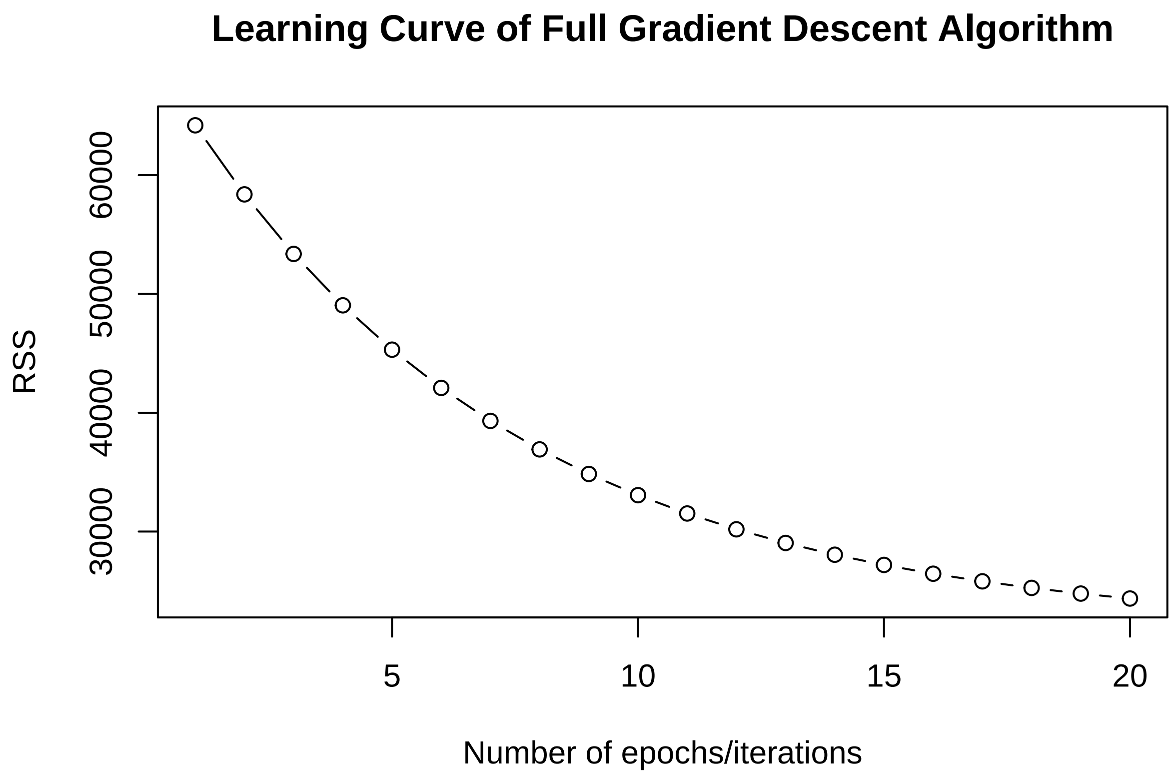
```
RSS = error%*%error
sprintf("The corresponding RSS is %.3f",RSS)
```

```
## [1] "The corresponding RSS is 24363.301"
```

```
RSE = sqrt(RSS/(nrow(Data)-2))
sprintf("The corresponding RSE is %.3f",RSE)
```

```
## [1] "The corresponding RSE is 6.953"
```

```r
#Plot the learning curve for the FGD algo
plot(NumOfIt,NumOfRSS,
     type="b",
     main = "Learning Curve of Full Gradient Descent Algorithm",
     xlab = "Number of epochs/iterations",
     ylab = "RSS")
```



Question 4

Compare your results. Which combination of batch size (32 or 506) and number of epochs/iterations would you recommend?

After running the two algorithms for the same number of epochs, we find the difference between the learning performances of two algorithms. The RSS of the Mini-Batch gradient descent algorithm is at around 21847, which is lower than the RSS of the full gradient descent algorithm which is at 24363. Based on the RSS we would recommend the mini-batch gradient descent algorithm since it has a better learning performance.

To find a good number of iterations we must find a point, where when we increase the iterations the RSS will only change by a little. The number of iterations depends on the application and the computing cost / capacity available, but since our data set is rather small this is not an issue for us.

If we increase the iterations, we can see when the learning curves start to get flat, that is where the RSS starts changing by less, per added iteration. In the graphs below we see that that point is at around 1000 iterations for a batch size of 32 and at around 30 iterations for batch size of 506.

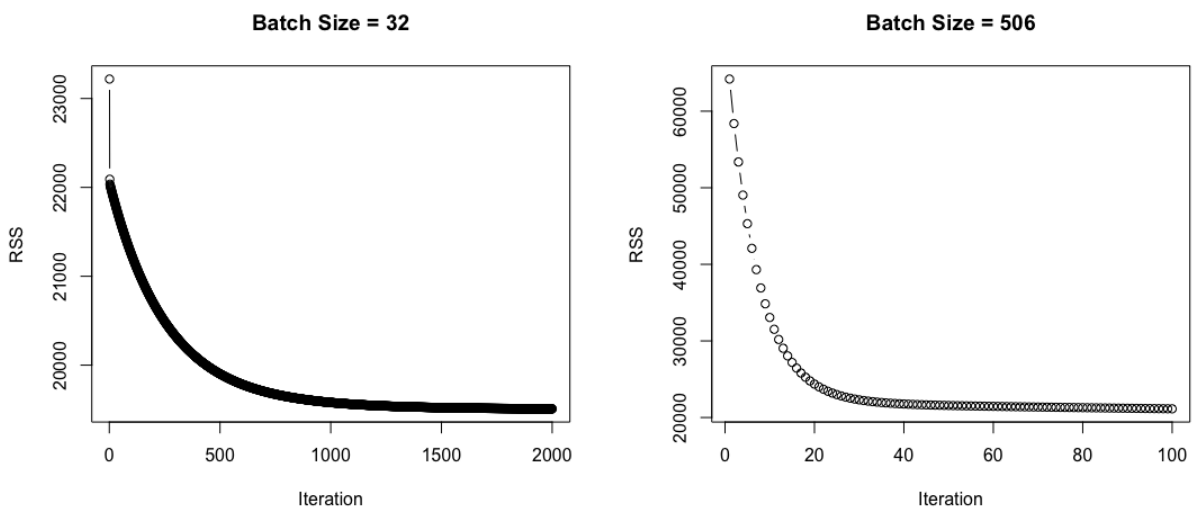


Figure 1: Running time v.s. Learning curves

If we compare the learning curve to the run times of the algorithms, we find

Number of Iterations	Batch Size	Run Time (secs)
1	32	0.0036
20	32	0.0254
1000	32	0.6355
1	506	0.0012
20	506	0.0046
30	506	0.0060

With increasing iterations, run times go up. The trade-off in run time vs. learning curve is much larger with full gradient descent than with mini-batch gradient descent. As computing power is not an issue with this data set we can use large numbers of iterations, but with larger data sets one would have to construct a model that is well suited for the specific project.

Question 5

Measure the run time of the full gradient descent algorithm (batch size 506) and the mini-batch gradient descent algorithm (batch size 32) for 20 epochs/iterations. Compare the two results.

```
# Question 5 -------------------------------------------------------------

MBGD.time = MBGD.end - MBGD.start
sprintf("The run time of MBGD for 20 epochs is %.3f secs",MBGD.time)
```

```
## [1] "The run time of MBGD for 20 epochs is 0.053 secs"
```

```
FGD.time = FGD.end - FGD.start
sprintf("The run time of FGD for 20 epochs is %.3f secs",FGD.time)
```

```
## [1] "The run time of FGD for 20 epochs is 0.017 secs"
```

```
sprintf("The time difference is %.3f secs",abs(FGD.time - MBGD.time))
```

```
## [1] "The time difference is 0.036 secs"
```

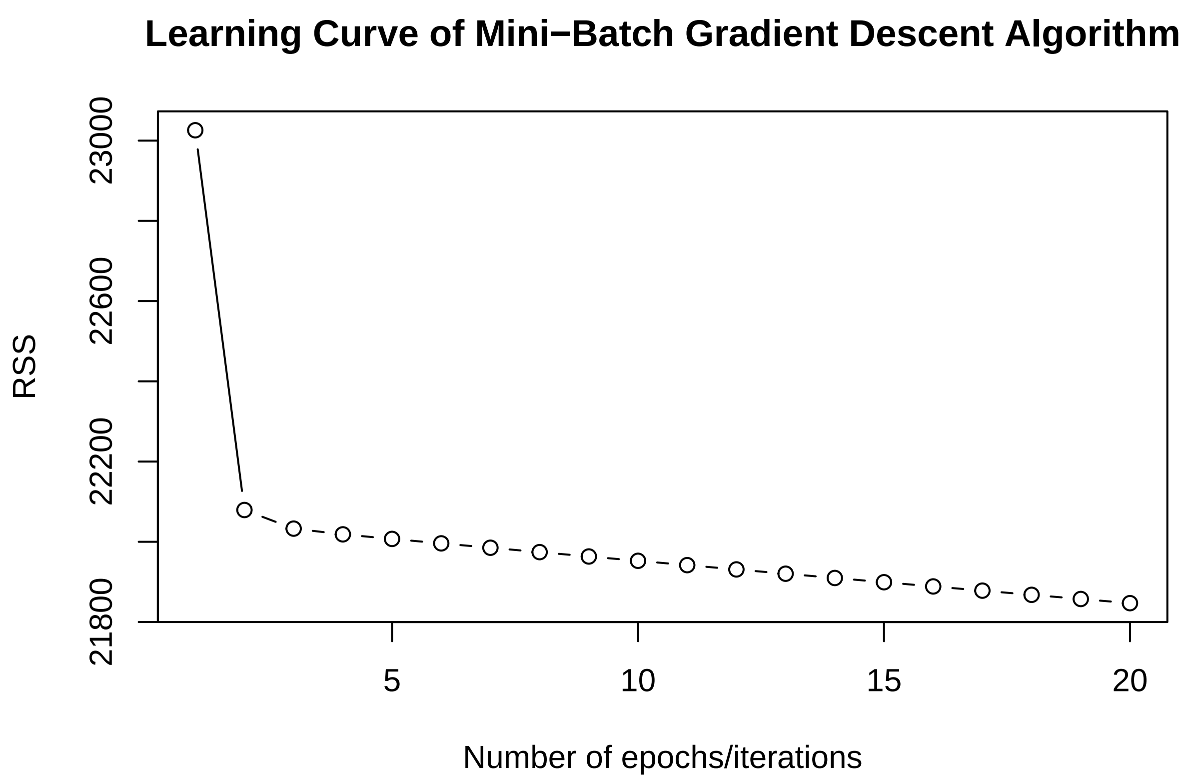
From the results above, we can find that the mini-batch gradient descent algorithm is slower than the full gradient descent algorithm when the epochs number is set to be 20. This is easy to be understood, the mini-batch gradient descent algorithm conducts more loops than the other algorithm in one epoch and loops cost time. Intuitively, the stochastic gradient descent algorithm would cost even more time the two algorithm we have discussed, since it will include more loops in one epoch.

Challenge Quesiton

Wrtie a general algorithm with a variable batch size and number of epochs.

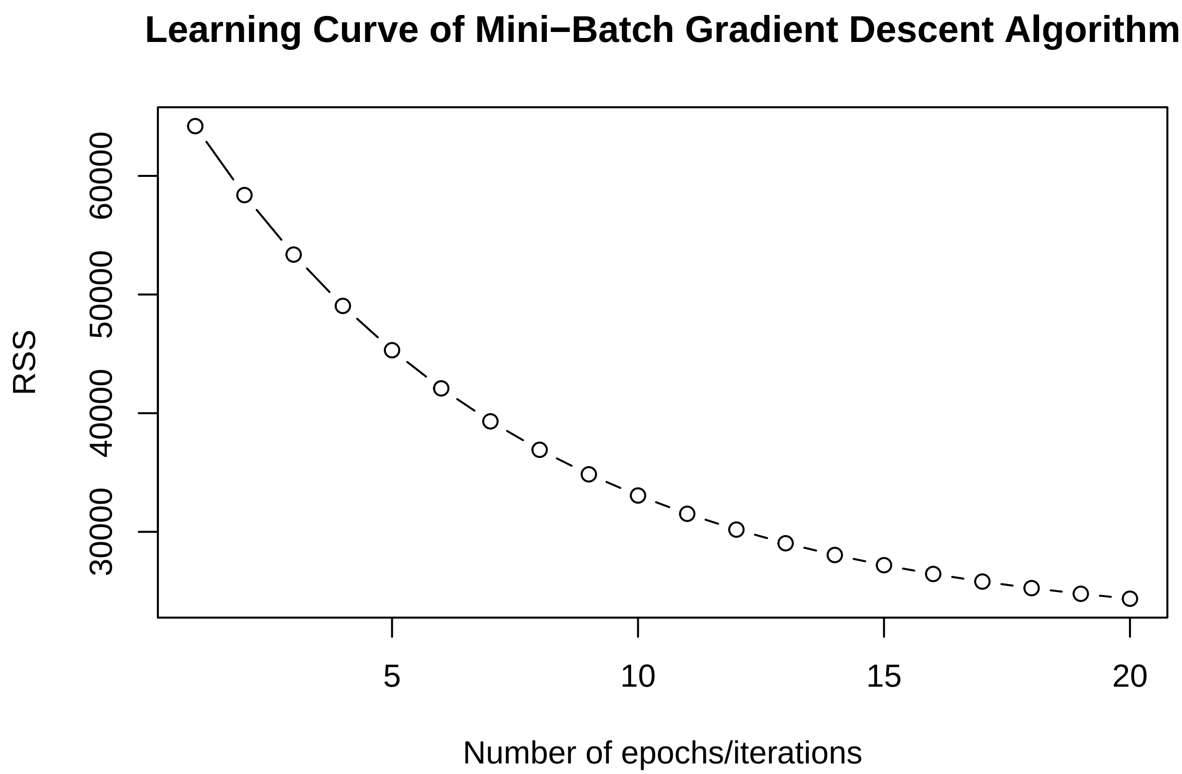
```
# Challenge ----------------------------------------------------------------

# Notice this algo is only for the Boston data since we don't allow other input
# to be variable

# reset weights
weights = c(30,0)

# Define the general algorithm with a variable batch size and number of epochs
algo <- function(BatSize,NumIt){
  # Number of batches for the Data (here I use the command of ceiling of floor
  # to avoid confusion between integer and non-interger)
  NumBat = ceiling(nrow(Data)/BatSize)

  # Create lists to store results of each epoch
  NumOfIt = rep(NA,NumIt)
  NumOfRSS = rep(NA,NumIt)

  # Loops w.r.t. the number of epochs
  for (i in 1:NumIt){
    # Loops w.r.t. the number of batches for specified epoch
    for (j in 1:NumBat){
      # Create Mini Batch regarding for index from 1 to 16
      if (j < NumBat){
        Data.Train = Data[((j-1)*BatSize+1):(j*BatSize),]
      } else {
        Data.Train = Data[((j-1)*BatSize+1):nrow(Data),]
      }

      # calculate stochastic descent algorithm for the specified mini batch
      # in specified epoch
      yhat = Data.Train$lstat*weights[2]+weights[1]
      error = Data.Train$medv - yhat
      weights[1] = weights[1] + 2*eta*sum(error)
      weights[2] = weights[2] + 2*eta*sum(error*Data.Train$lstat)
      # print(weights)
    }

    # Compute and store numbers of iterations and RSS for each epoch
    NumOfIt[i] = i
    yhat=Data$lstat*weights[2]+weights[1]
    error =  Data$medv  - yhat
    RSS = error%*%error
    NumOfRSS[i] = RSS
  }

  # Calcualate error for the whole sample using the final weights
  yhat=Data$lstat*weights[2]+weights[1]
  error =  Data$medv  - yhat
  RSS = error%*%error
  RSE = sqrt(RSS/(nrow(Data)-2))
```

```r
  # Final output
  sprintf("The final updated weights is (%.3f,%.3f)",weights[1],weights[2])
  sprintf("The corresponding RSS is %.3f",RSS)
  sprintf("The corresponding RSE is %.3f",RSE)

  # plot the learning curve
  #Plot the learning curve for the FGD algo
  plot(NumOfIt,NumOfRSS,
       type="b",
       main = "Learning Curve of Mini-Batch Gradient Descent Algorithm",
       xlab = "Number of epochs/iterations",
       ylab = "RSS")

  # output
  output= c("weights"=weights,
            "RSS"=RSS,
            "RSE"=RSE)
  return(output)
}

algo(32,20)
```




```
##      weights1      weights2           RSS           RSE
##     30.156899     -0.674513 21847.044734      6.583867
```

```r
algo(506,20)
```




```
##     weights1     weights2           RSS          RSE
##   30.1657489   -0.5326343 24363.3007214    6.9526889
```