Gradient Descent Algorithms

Muhammad Tamjid Rahman

Contents

Loading R packages

```
library(uuml)
library(ggplot2)
library(glmnet)
```

1 Implementation the gradient as a function

```
l_grad <- function(y, X, theta){
    grad <- t(y-(exp(X%*%theta)/(1+exp(X %*% theta)))) %*% X

    return(grad/nrow(X))
}
```

```
l_grad(y, X, theta = c(0,0,0))
```

```
##      (Intercept)     gre_sd     gpa_sd
## [1,]     -0.1825 0.08574746 0.08285471
```

```
l_grad(y, X, theta = c(-1,.5,.5))
```

```
##      (Intercept)     gre_sd      gpa_sd
## [1,]  0.02174332 -0.0395161 -0.04264481
```

1.1 Implementation of Gradient Descent

1)

```
l <- function(y, X, theta){
    lf<-t(y) %*% X %*% theta -sum(log(1+exp(X%*%theta)))

    return(lf)
}
```

```
l(y, X, theta = c(0,0,0))
```

```
##            [,1]
## [1,] -277.2589
```

```
l(y, X, theta = c(-1,.5,.5))
```

```
##            [,1]
## [1,] -244.5342
```

2) logistic regression

```
logit <- glm(admit~gre_sd+gpa_sd, data=binary, family = binomial(link="logit"))

MLE<-logit$coefficients
MLE
```

```
## (Intercept)      gre_sd      gpa_sd
##  -0.8097503   0.3108184   0.2872087
```

3) Implementation of gradient descent algorithms

a) batch gradient descent

```
batch <- function(y, X, eta,size){
    theta <- c(0,0,0)
    iter <- c()
    theta_v<-matrix(0,size,3)

    for (i in 1:size) {
        theta_r <- -t(l_grad(y, X, theta))
        theta <- theta - eta * theta_r
        iter[i] <- l(y, X, theta)
        theta_v[i,]<- theta
    }
    return(list("coef" = theta, "iter" = iter,"theta_v"=theta_v))
}
batch(y, X, .1, 10)
```

b) stochastic gradient descent

```
sgd <- function(y, X, eta,size){
    theta <- c(0,0,0)
    iter <- c()
    theta_v<-matrix(0,size,3)
    for (i in 1:size) {
        k <- sample(1:nrow(X),1)
        theta_r <- -t(l_grad(y[k], t(X[k,]), theta))
        theta <- theta - eta * theta_r
        iter[i] <- l(y, X, theta)
        theta_v[i,]<- theta
    }
    return(list("coef" = theta, "iter" = iter, "theta_v"=theta_v))
```

```
}
sgd(y, X, .1,10)
```

c) mini-batch gradient descent

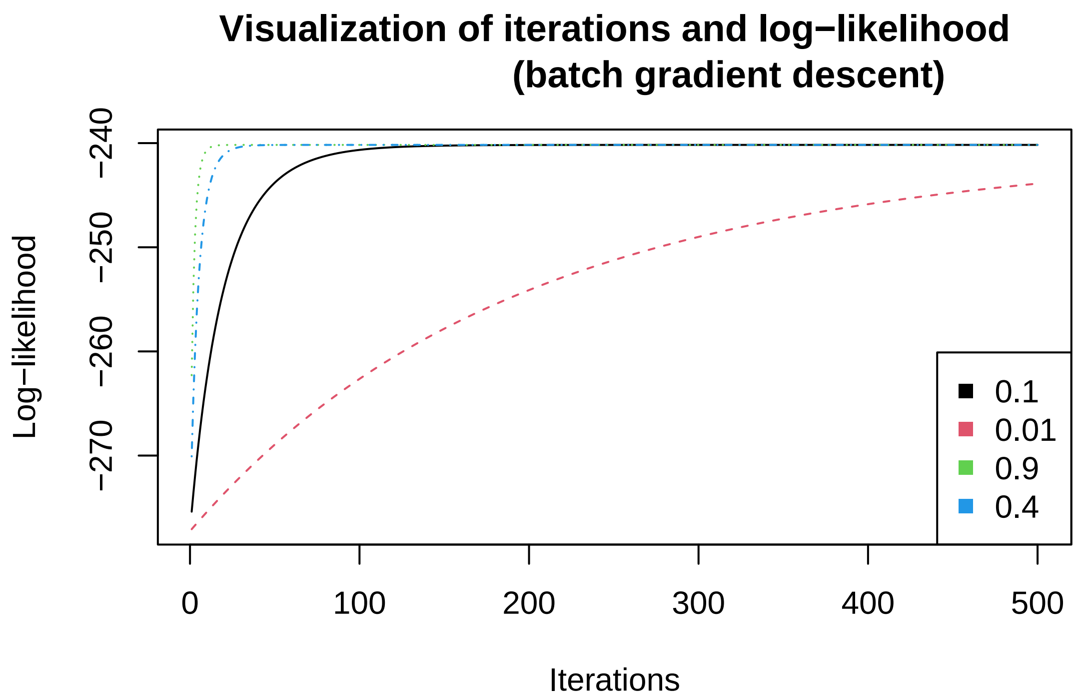
```
mgd <- function(y, X, eta,size){
    theta <- c(0,0,0)
    iter <- c()
    theta_v<-matrix(0,size,3)
    for (i in 1:size) {
        k <- sample(nrow(X), 10, replace = FALSE)
        theta_r <- -t(l_grad(y[k], X[k,], theta))
        theta <- theta - eta * theta_r
        iter[i] <- l(y, X, theta)
        theta_v[i,]<- theta
    }
    return(list("coef" = theta, "iter" = iter,"theta_v"=theta_v))
}
mgd(y, X, .1,10)
```

4)

a) different learning parameters η

batch gradient descent

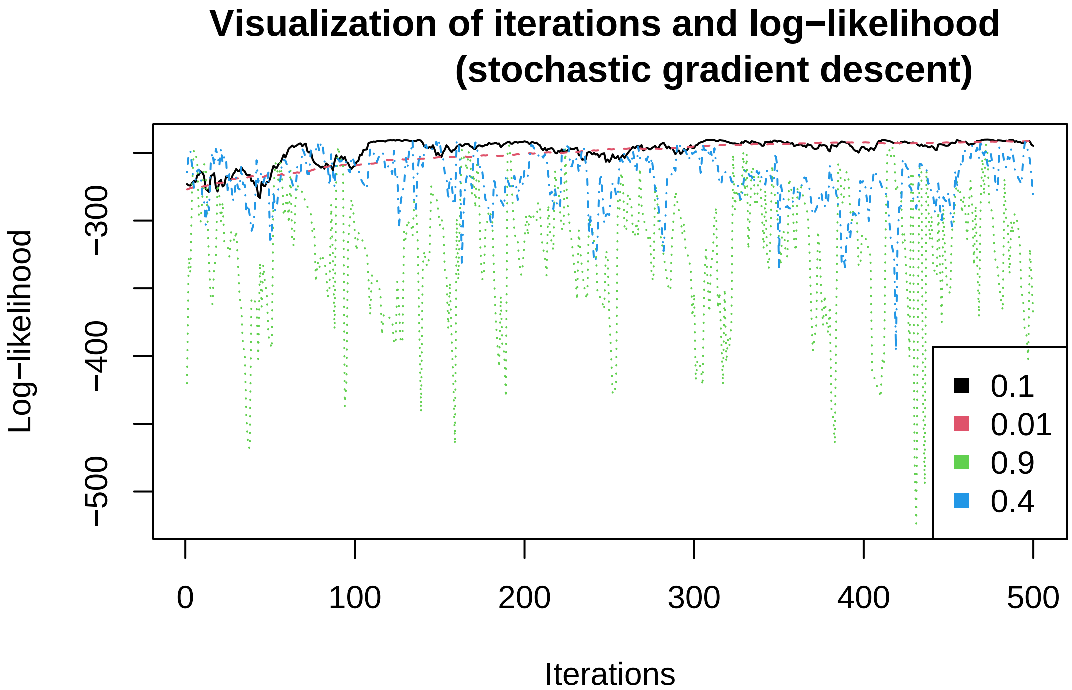
```
batch_eta1<-batch(y, X, .1,500)
batch_eta2<-batch(y, X, .01,500)
batch_eta3<-batch(y, X, .9,500)
batch_eta4<-batch(y, X, .4,500)
```



We can see from the plot that the algorithm converge after 120 iterations for learning rate .1. For learning rate .4 and .9 it converges after 25 iterations. For learning rate 0.01 need more iteration to converge.

stochastic gradient descent

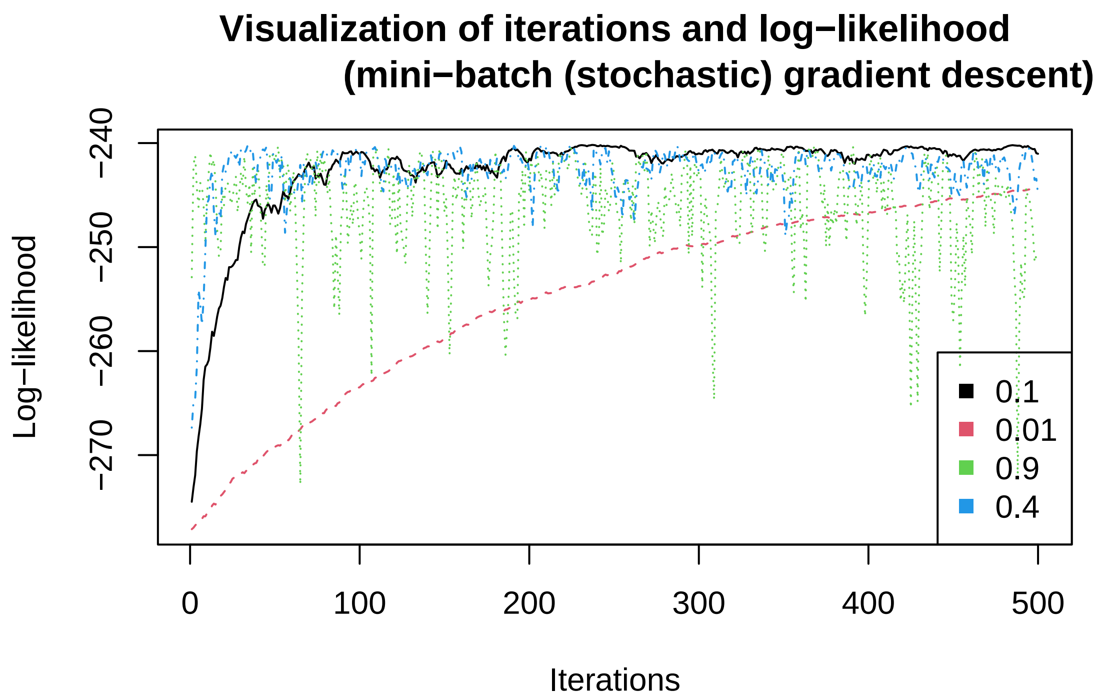
```
sgd_eta1<-sgd(y, X, .1,500)
sgd_eta2<-sgd(y, X, .01,500)
sgd_eta3<-sgd(y, X, .9,500)
sgd_eta4<-sgd(y, X, .4,500)
```



We can see from the plot that the algorithm converge after 130 iterations for learning rate .1 and .01. But But for .09 and .4 it hardly converge.

mini-batch (stochastic) gradient descent

```
mgd_eta1<-mgd(y, X, .1,500)
mgd_eta2<-mgd(y, X, .01,500)
mgd_eta3<-mgd(y, X, .9,500)
mgd_eta4<-mgd(y, X, .4,500)
```

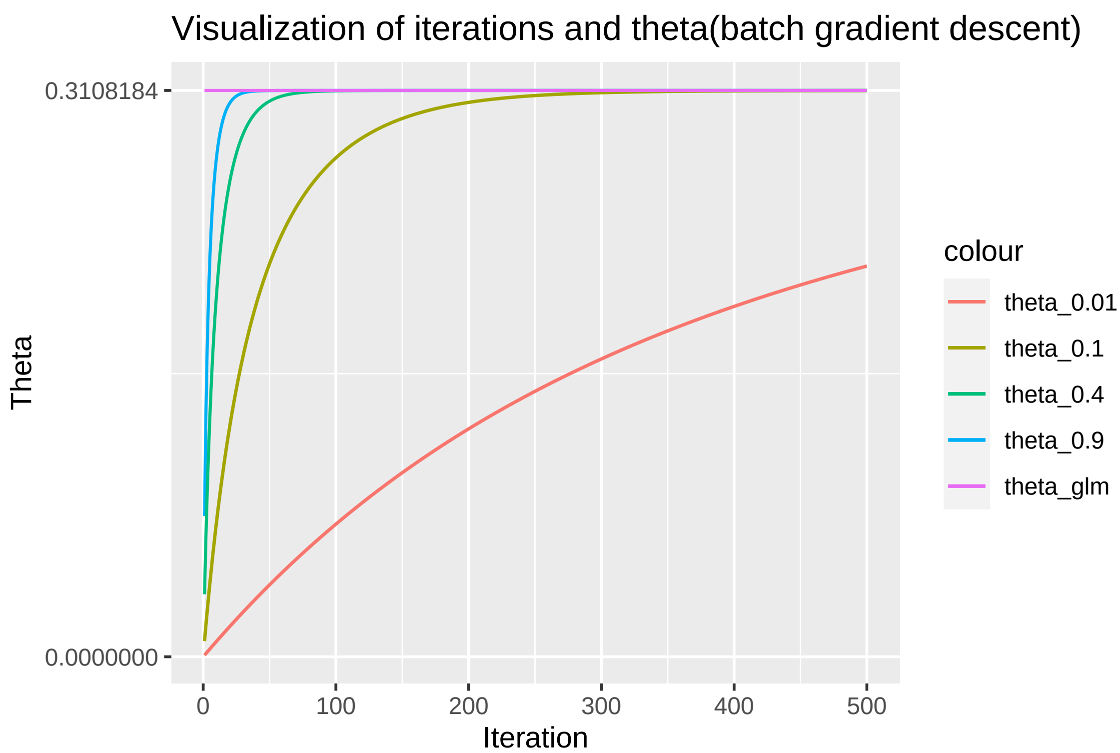


We can see from the plot that the algorithm converge after 90 iterations for learning rate 0.1. But for .01 and .4 it hardly converge. It became even harder for .9 .

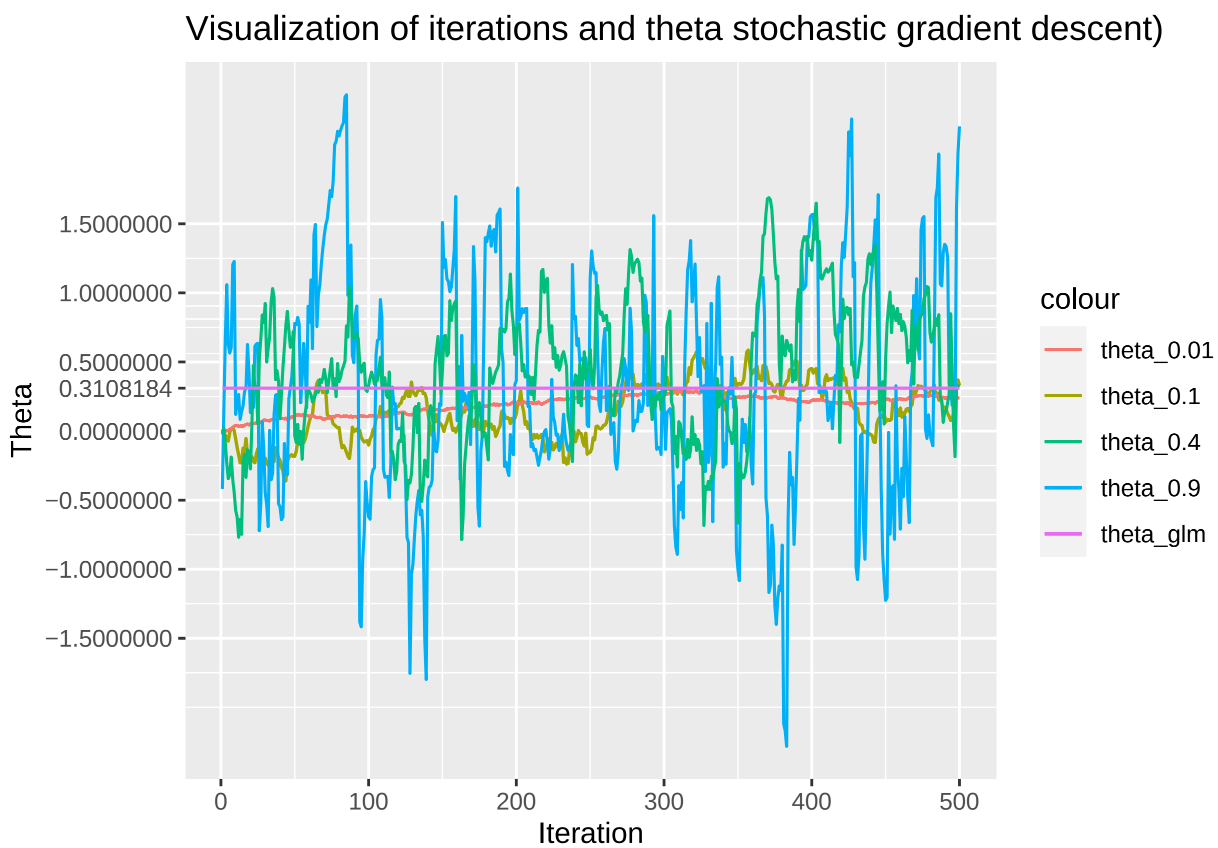
b)

```
sgd_f<-sgd(y, X, .1,610)
h=as.numeric( MLE[2])
```

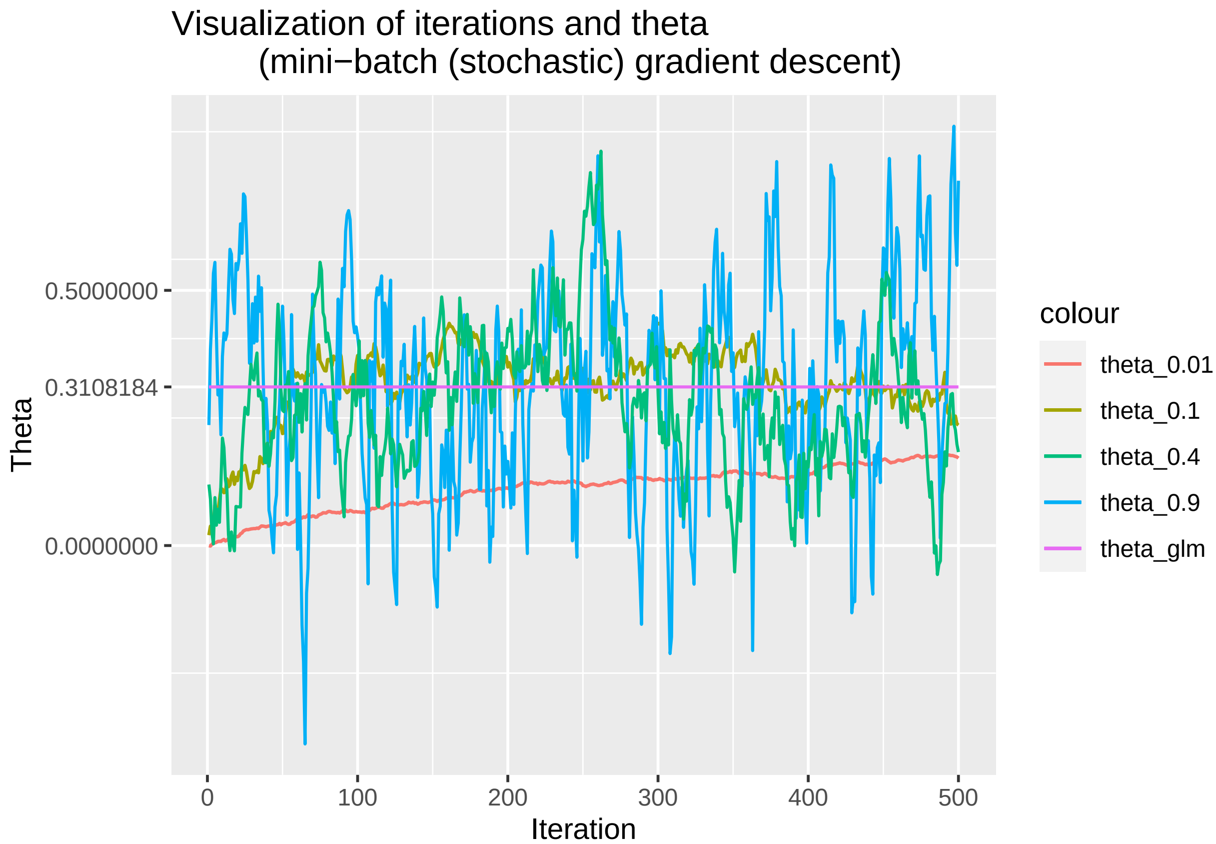
```
ggplot()+
    geom_line(mapping = aes(x=1:500, y = batch_eta1$theta_v[,2],color = "theta_0.1"),size=.59) +
    geom_line(mapping = aes(x=1:500, y = batch_eta2$theta_v[,2],color = "theta_0.01"),size=.6)+
    geom_line(mapping = aes(x=1:500, y = batch_eta3$theta_v[,2],color = "theta_0.9"),size=.55)+
    geom_line(mapping = aes(x=1:500, y = batch_eta4$theta_v[,2],color = "theta_0.4"),size=.55)+
    geom_line(mapping = aes(x=1:500, y = MLE[2],color = "theta_glm"),size=.55)+
    #geom_hline(aes(yintercept=h)) +
    scale_y_continuous(breaks =c(-1.5,-1,-0.5,0,.5,1,1.5, h))+
    labs(title = "Visualization of iterations and theta(batch gradient descent)",
         y='Theta',
         x='Iteration')+
    #geom_smooth(method = lm)+
    theme()
```




```
ggplot()+
    geom_line(mapping = aes(x=1:500, y = sgd_eta1$theta_v[,2],color = "theta_0.1"),size=.59) +
    geom_line(mapping = aes(x=1:500, y = sgd_eta2$theta_v[,2],color = "theta_0.01"),size=.6)+
    geom_line(mapping = aes(x=1:500, y = sgd_eta3$theta_v[,2],color = "theta_0.9"),size=.55)+
    geom_line(mapping = aes(x=1:500, y = sgd_eta4$theta_v[,2],color = "theta_0.4"),size=.55)+
    geom_line(mapping = aes(x=1:500, y = MLE[2],color = "theta_glm"),size=.55)+
    #geom_hline(aes(yintercept=h)) +
    scale_y_continuous(breaks =c(-1.5,-1,-0.5,0,.5,1,1.5, h))+
    labs(title = "Visualization of iterations and theta stochastic gradient descent)",
         y='Theta',
         x='Iteration')+
    #geom_smooth(method = lm)+
    theme()
```




```r
ggplot()+
    geom_line(mapping = aes(x=1:500, y = mgd_eta1$theta_v[,2],color = "theta_0.1"),size=.59) +
    geom_line(mapping = aes(x=1:500, y = mgd_eta2$theta_v[,2],color = "theta_0.01"),size=.6)+
    geom_line(mapping = aes(x=1:500, y = mgd_eta3$theta_v[,2],color = "theta_0.9"),size=.55)+
    geom_line(mapping = aes(x=1:500, y = mgd_eta4$theta_v[,2],color = "theta_0.4"),size=.55)+
     geom_line(mapping = aes(x=1:500, y = MLE[2],color = "theta_glm"),size=.55)+
    #geom_hline(aes(yintercept=h)) +
    scale_y_continuous(breaks =c(-1.5,-1,-0.5,0,.5,1,1.5, h))+
    labs(title = "Visualization of iterations and theta
         (mini-batch (stochastic) gradient descent)",
         y='Theta',
         x='Iteration')+
    #geom_smooth(method = lm)+
    theme()
```



2 Regularized Regression

```
data("prob2_train")

dim(prob2_train)
```

```
## [1] 200 241
```

1

A linear regression model is fit to the training data. But it couldn't estimate the coefficients and the standard errors. There are more predictors than the observations. This may be one of the reasons.

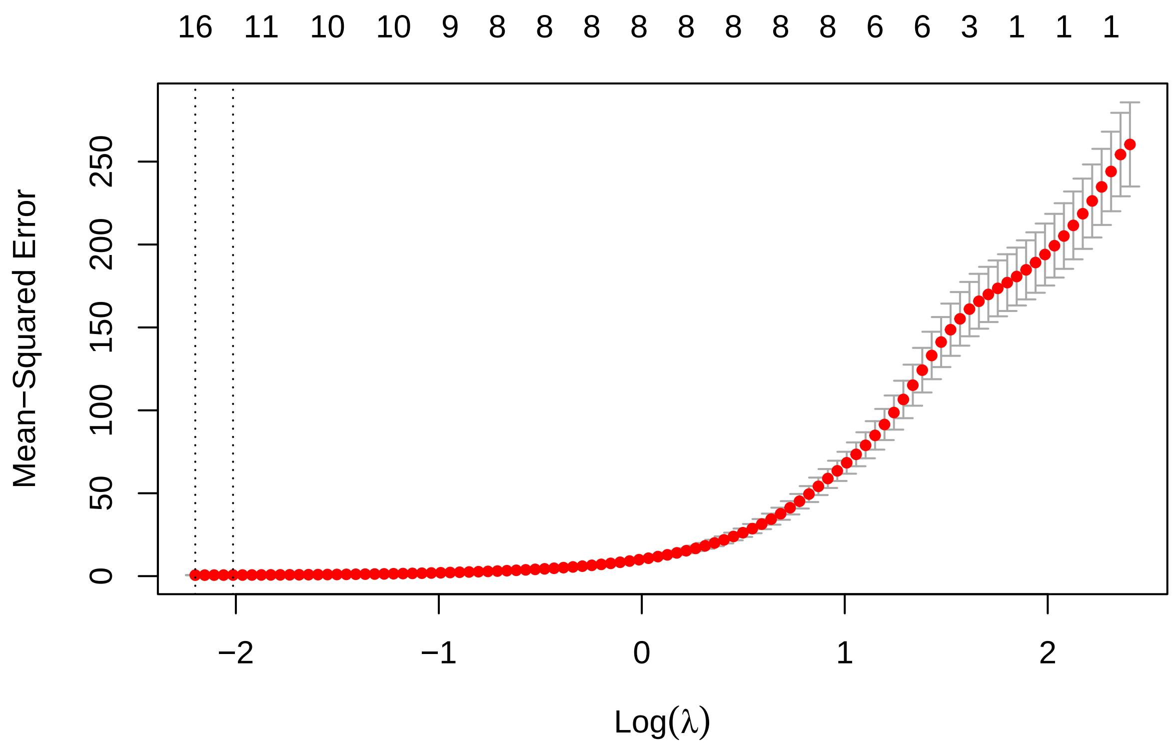
2

```
x<- as.matrix( prob2_train[,1:240])
y<- as.matrix(prob2_train$y)


cvfit <- cv.glmnet(x, y, alpha=1,nfolds = 10, intercept = T, standardize = T)
```

The cv.glmnet function is used to fit a lasso regression to the training data. Here we set alpha=1 to perform lasso regression. This function performs a 10 fold cross-validation to find the value of complexity parameter λ and estimates the predictors.

```
plot(cvfit)
```



From the plot we see that the MSE is roughly stable till log(λ)=0, but after it's increasing rapidly.

3

```
cvfit$lambda.min
```

```
## [1] 0.1108034
```

```
cvfit$lambda.1se
```

```
## [1] 0.133463
```

```
sum(coef(cvfit, s = "lambda.min") !=0)
```

```
## [1] 17
```

From the out we see, λ_{min} gives 16 non-zerro coefficients excluding the intercept.

```
sum(coef(cvfit, s = "lambda.1se") !=0)
```

```
## [1] 13
```

From the out we see, λ_{1se} gives 12 non-zerro coefficients excluding the intercept.

```
which(coef(cvfit, s = "lambda.1se") == 0 & coef(cvfit, s = "lambda.min") != 0)-1
```

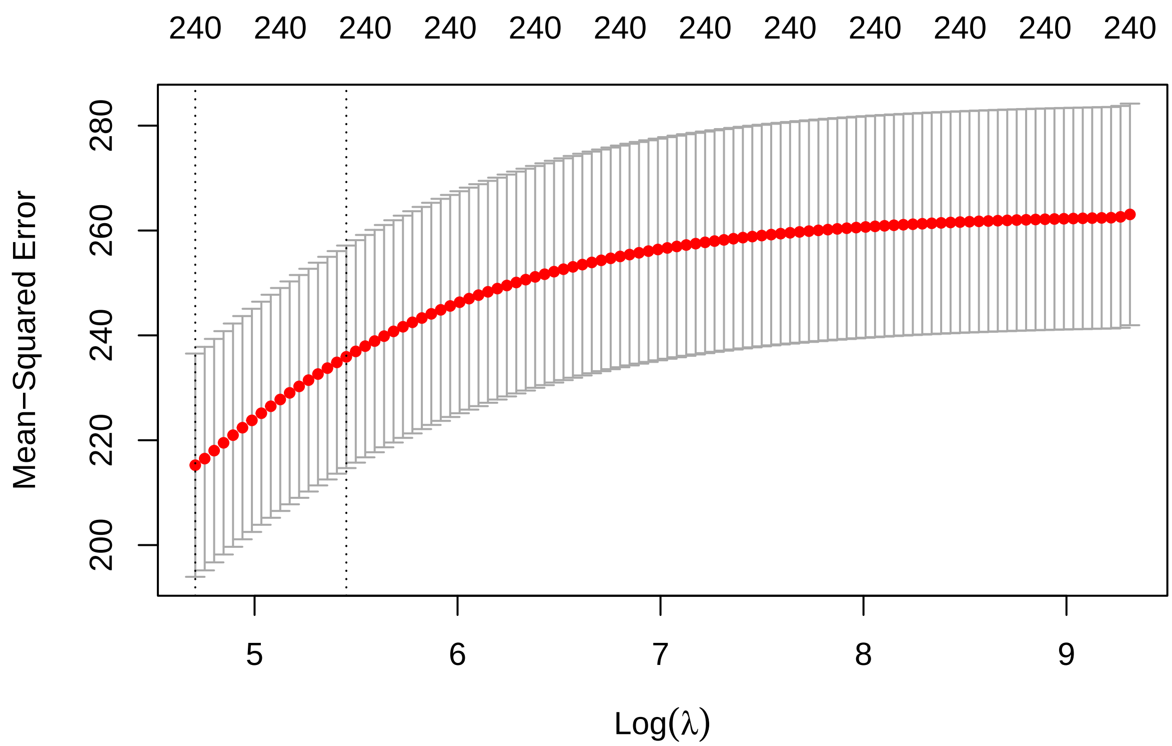
```
## [1]  75  88 168 231
```

The variables are in the above columns presents in both λ_{1se} and λ_{min} parameters.

4

The cv.glmnet function is also used to fit a ridge regression to the training data. Here we set alpha=0 to perform ridge regression.

```
ridge <- cv.glmnet(x=x, y=y, alpha  = 0,nfolds = 10, intercept = T, standardize = T)
plot(ridge)
```



5

```
data("prob2_test")

x_test<- as.matrix( prob2_test[,1:240])
y_test<- as.matrix(prob2_test$y)

#Prediction for Lasso
lasso_min<-predict(cvfit,s="lambda.min", newx = x_test)
lasso_1se<-predict(cvfit,s="lambda.1se", newx = x_test)

#MAE
mean(abs(prob2_test$y - lasso_min))
```

```
## [1] 0.6949652
```

```
mean(abs(prob2_test$y - lasso_1se))
```

```
## [1] 0.7228752
```

```r
#RMSE
sqrt(mean((prob2_test$y - lasso_min)^2))
```

```
## [1] 0.8772127
```

```r
sqrt(mean((prob2_test$y - lasso_1se)^2))
```

```
## [1] 0.9214398
```

```r
#Prediction for Ridge
ridge_min<-predict(ridge, s="lambda.min", newx = x_test)
ridge_1se<-predict(ridge, s="lambda.1se", newx = x_test)

#MAE
mean(abs(prob2_test$y - ridge_min))
```

```
## [1] 13.17532
```

```r
mean(abs(prob2_test$y - ridge_1se))
```

```
## [1] 13.84204
```

```r
#RMSE
sqrt(mean((prob2_test$y - ridge_min)^2))
```

```
## [1] 16.30217
```

```r
sqrt(mean((prob2_test$y - ridge_1se)^2))
```

```
## [1] 17.16112
```

	Lasso(λ_{min})	Lasso(λ_{1se})	Ridge(λ_{min})	Ridge(λ_{1se})
MAE	0.695	0.715	13.175	13.706
RMSE	0.877	0.909	16.302	16.983

The model with the least value of MAE & RMSE is the best model. Looking at the MAE & RMSE values of different models we can say that the lasso model with minimum lambda is the best model in our case.

3 Gradient Descent for penalized logistic regression

1 Derivation of the gradient for NLL$_r(\theta, y, X, \lambda)$ with respect to θ

The likelihood function,

$$l_r(\theta, y, X, \lambda) = \frac{1}{n} l(\theta, y, X) - \frac{\lambda}{2} \sum_{i=1}^{P} \theta_i^2$$

The gradient for NLL$_r(\theta, y, X, \lambda)$ with respect to $\theta =$

$$-\frac{\delta l_r}{\delta \theta} = -x_i(y_i - \frac{exp(x_i\theta_i)}{1 + exp(x_i\theta_i)}) + \frac{\lambda}{2} \sum_{i=1}^{P} 2\theta_i$$

2) Implementation the gradient as a function

```r
data('binary')
```

```
binary$gre_sd <- (binary$gre-mean(binary$gre))/sd(binary$gre)
binary$gpa_sd <- (binary$gpa - mean(binary$gpa))/sd(binary$gpa)
X <- model.matrix(admit ~ gre_sd + gpa_sd, binary)
y <- binary$admit
```

```
lr_grad <- function(y, X, theta, lambda){
    grad <- t(y-(exp(X%*%theta)/(1+exp(X %*% theta)))) %*% X

    return(grad/nrow(X)-(lambda)*c(0,theta[-1]))
}
```

```
lr_grad(y, X, theta = c(0,0,0),lambda=0)
```

```
##      (Intercept)     gre_sd     gpa_sd
## [1,]     -0.1825 0.08574746 0.08285471
```

```
lr_grad(y, X, theta = c(0,0,0),lambda=1)
```

```
##      (Intercept)     gre_sd     gpa_sd
## [1,]     -0.1825 0.08574746 0.08285471
```

```
lr_grad(y, X, theta = c(-1,0.5,0.5),lambda=1)
```

```
##      (Intercept)     gre_sd     gpa_sd
## [1,]  0.02174332 -0.5395161 -0.5426448
```

3 Implementation of the regularized log-likelihood l_r

```
lr <- function(y, X, theta,lambda){
    lf<- (t(y) %*% X %*% theta -sum(log(1+exp(X%*%theta))))

    return(lf/nrow(X)-(lambda/2)*(theta[2]^2+theta[2]^2))
}
```

```
lr(y, X, theta = c(0,0,0),lambda=0)
```

```
##            [,1]
## [1,] -0.6931472
```

```
lr(y, X, theta = c(0,0,0),lambda=1)
```

```
##            [,1]
## [1,] -0.6931472
```

```
lr(y, X, theta = c(-1,0.5,0.5),lambda=1)
```

```
##            [,1]
## [1,] -0.8613355
```

4 logistic regression with ridge penalty

```
mod <- glmnet(x=X[,-1], y=y, alpha  = 0, lambda=1)
mod$beta
```

```
## 2 x 1 sparse Matrix of class "dgCMatrix"
##                s0
## gre_sd 0.02444902
```

```
## gpa_sd 0.02339978
```

5 Implementation of the following gradient descent algorithms for the penalized logistic objective

a) ordinary (batch) gradient descent

```
batch_r <- function(y, X, eta,size,lambda){
    theta <- c(0,0,0)
    iter <- c()
    theta_v<-matrix(0,size,3)

    for (i in 1:size) {
        theta_r <- -t(lr_grad(y, X, theta,lambda))
        theta <- theta - eta * theta_r
        iter[i] <- lr(y, X, theta, lambda)
        theta_v[i,]<- theta
    }
    return(list("coef" = theta, "iter" = iter,"theta_v"=theta_v))
}
batch_r(y, X, .1, 10,1)
```

```
## $coef
##                    [,1]
## (Intercept) -0.16331666
## gre_sd       0.04880593
## gpa_sd       0.04703366
##
## $iter
##  [1] -0.6885345 -0.6844206 -0.6807125 -0.6773389 -0.6742447 -0.6713875
##  [7] -0.6687340 -0.6662580 -0.6639386 -0.6617590
##
## $theta_v
##              [,1]        [,2]        [,3]
##  [1,] -0.01825000 0.008574746 0.008285471
##  [2,] -0.03604379 0.015998829 0.015453647
##  [3,] -0.05339295 0.022426916 0.021655340
##  [4,] -0.07030891 0.027993017 0.027021149
##  [5,] -0.08680286 0.032813196 0.031664119
##  [6,] -0.10288575 0.036987942 0.035682067
##  [7,] -0.11856829 0.040604241 0.039159618
##  [8,] -0.13386089 0.043737372 0.042169963
##  [9,] -0.14877371 0.046452480 0.044776400
## [10,] -0.16331666 0.048805931 0.047033664
```

b) mini-batch gradient descent

```
mgd_r <- function(y, X, eta,size,lambda){
    theta <- c(0,0,0)
    iter <- c()
    theta_v<-matrix(0,size,3)
    for (i in 1:size) {
        k <- sample(nrow(X), 10, replace = FALSE)
        theta_r <- -t(lr_grad(y[k], X[k,], theta, lambda))
```

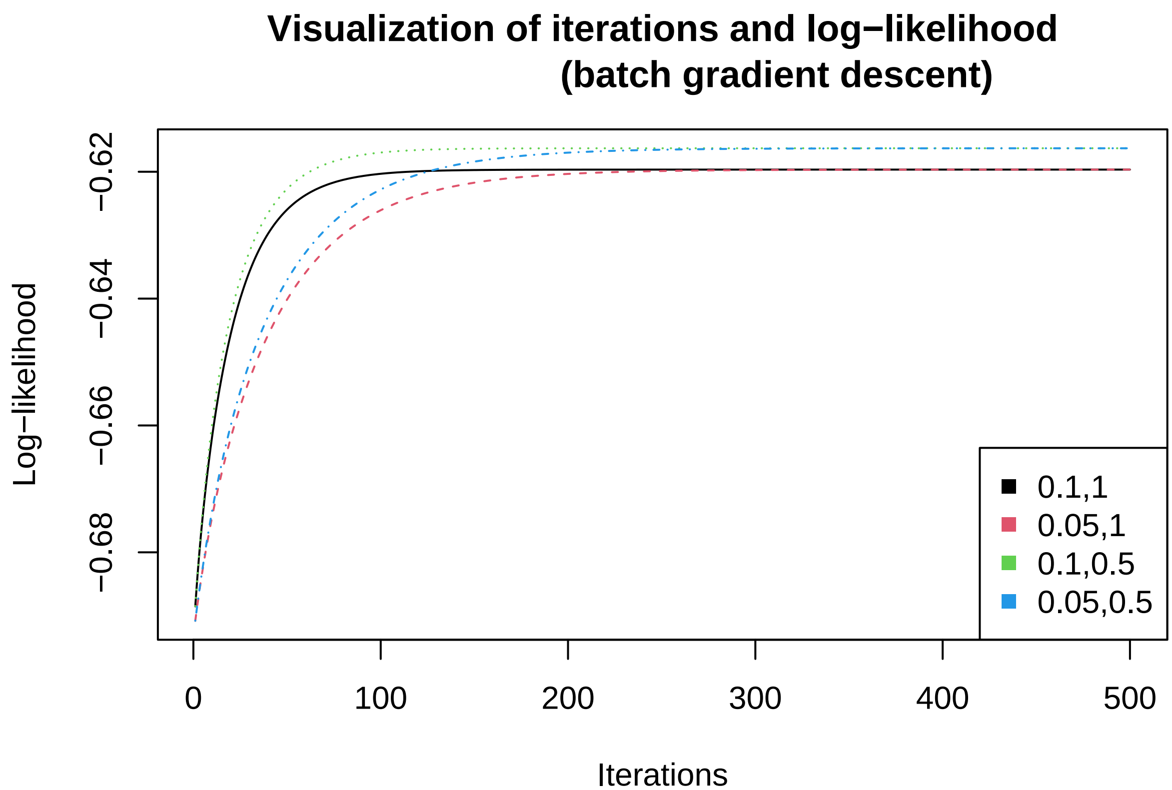
```
        theta <- theta - eta * theta_r
        iter[i] <- lr(y, X, theta,lambda)
        theta_v[i,]<- theta
    }
    return(list("coef" = theta, "iter" = iter,"theta_v"=theta_v))
}
mgd_r(y, X, .1,10,1)
```

```
## $coef
##                    [,1]
## (Intercept) -0.13471452
## gre_sd       0.09854938
## gpa_sd       0.02509372
##
## $iter
##  [1] -0.6871058 -0.6847663 -0.6861889 -0.6852320 -0.6835586 -0.6808116
##  [7] -0.6752185 -0.6763928 -0.6737020 -0.6715280
##
## $theta_v
##              [,1]       [,2]         [,3]
##  [1,] -0.03000000 0.02364164 -0.008810543
##  [2,] -0.02956284 0.05470607  0.023700960
##  [3,] -0.03839973 0.06555832 -0.008418437
##  [4,] -0.04761453 0.06865318 -0.013815533
##  [5,] -0.06771200 0.03117101 -0.044227301
##  [6,] -0.08589974 0.02056912 -0.042986607
##  [7,] -0.10405289 0.04847832 -0.015261886
##  [8,] -0.10099407 0.05246472 -0.022406042
##  [9,] -0.11835899 0.07872507 -0.002494274
## [10,] -0.13471452 0.09854938  0.025093718
```

6 different learning parameters η and λ

batch gradient descent

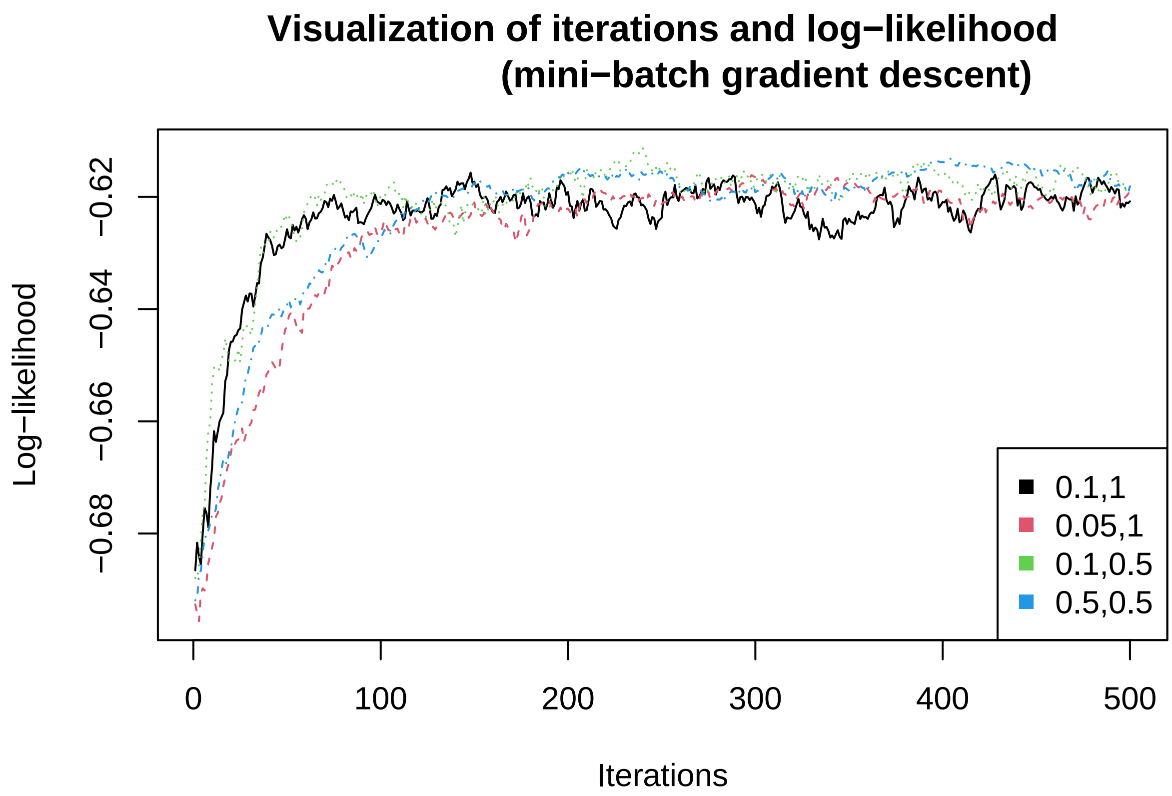
```
batch_eta1<-batch_r(y, X, .1,500,1)
batch_eta2<-batch_r(y, X, .05,500,1)
batch_eta3<-batch_r(y, X, .1,500,.5)
batch_eta4<-batch_r(y, X, .05,500,.5)
m_batch<-cbind(batch_eta1$iter,batch_eta2$iter,batch_eta3$iter,batch_eta4$iter)
plot_batch <- matplot(m_batch, type = c("l"),
                      main = "Visualization of iterations and log-likelihood
                      (batch gradient descent)",
                      ylab = "Log-likelihood", xlab = "Iterations")
legend("bottomright", legend = c("0.1,1","0.05,1","0.1,0.5","0.05,0.5"), col=1:4, pch=15)
```



η	λ	At iteration it converge
0.1	1	100
0.05	1	230
0.1	0.5	110
0.05	0.5	220

mini batch gradient descent

```
mgd_eta1<-mgd_r(y, X, .1,500,1)
mgd_eta2<-mgd_r(y, X, .05,500,1)
mgd_eta3<-mgd_r(y, X, .1,500,.5)
mgd_eta4<-mgd_r(y, X, .05,500,.5)
m_mgd<-cbind(mgd_eta1$iter,mgd_eta2$iter,mgd_eta3$iter,mgd_eta4$iter)
plot_mgd <- matplot(m_mgd, type =c("l"),
                    main = "Visualization of iterations and log-likelihood
                    (mini-batch gradient descent)",
                    ylab = "Log-likelihood", xlab = "Iterations", pch=15, col = 1:4)
legend("bottomright", pch=15, legend = c("0.1,1","0.05,1","0.1,0.5","0.5,0.5"), col=1:4)
```



η	λ	At iteration it converge
0.1	1	170
0.05	1	150
0.1	0.5	110
0.05	0.5	150