One vs ALL Logistic Regression using Stochastic Gradient Descent for Satellite Image Classification

Fangfei

November 8, 2018

Introduction

There are 5 different time series Landstat 6 bands images in trainning and test dataset, and in each dataset, there is a reponse column define the 30 different land type. The data size are over 12 million in each dataset. Since the data is huge, in this project, I will design a model for large data classification, the idea is using Stochastic Gradient Descent and Parrallel Processing to expediate the model processing, and using One vs All logistic regression for multi-class model

Enviroment preparing

```
#install.packages("raster")
#install.packages("rgdal")
#install.packages("rgeos")
#install.packages("ggplot2")
#install.packages("dplyr")
#install.packages("png")

library(raster)
library(rgdal)
library(rgeos)
library(ggplot2)
library(dplyr)
library(doParallel)
library(png)
```

Set parallel cores for parallel processing

```
cl=makeCluster(detectCores()-1)
registerDoParallel(cl)

setwd("Users/admin/Desktop/GDR")    ## Set the work direcory from the GDR file in des
ktop
```

Load data

```
train_data=list.files("/Users/admin/Desktop/GDR/classification_trial_data", pattern=
glob2rx("*train*.tif$"),full.names=TRUE)
test_data=list.files("/Users/admin/Desktop/GDR/classification_trial_data", pattern= g
lob2rx("*test*.tif$"),full.names=TRUE)
```

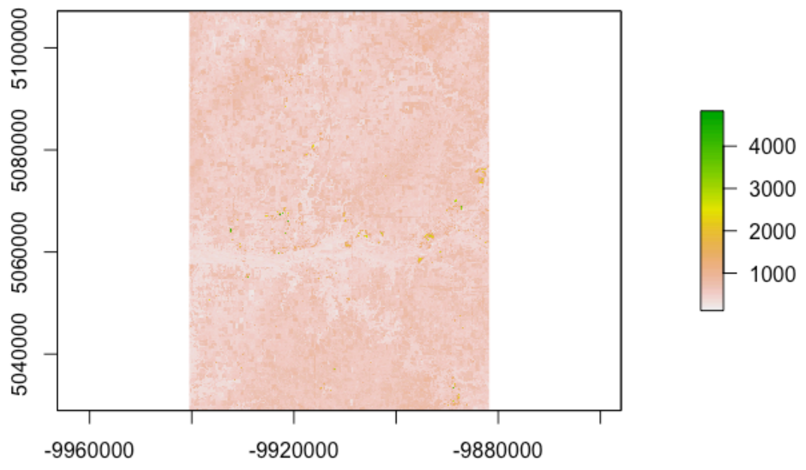
Data Preprocessing

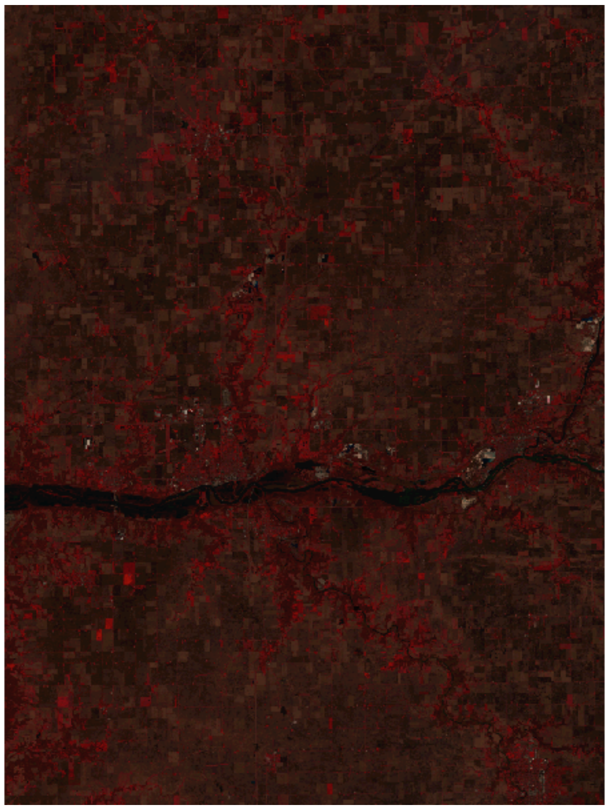
Use the *Summary* function to know the train and test data. Since Logistic regression will be used for classification, then normalize data is need before trainning model. Use the *Scale* function to normalize predictors in train and test data, and add 1 as intercept for model implemention.

```
train=stack(x=train_data)
train=brick(train)
train.y=matrix(getValues(train$train_classes),ncol=1)                  ## Get the respo
nse variable as train.y
train.x=matrix(getValues(train[[1:30]]),ncol=30)                       ## Get the predi
ctors as train.x
summary(train.x)
train.x=foreach(i=1:30,.combine = cbind)%do%scale(train.x[,i])         ## Normalize pre
dictors
train.x=cbind(rep(1,nrow(train.x)),train.x)                            ## Add intercept
for model implementation

test=stack(test_data)
test=brick(test)
test.y=matrix(getValues(test$test_classes),ncol=1)
test.x=matrix(getValues(test[[1:30]]),ncol=30)
summary(test.x)
test.x=foreach(i=1:30,.combine = cbind)%do%scale(test.x[,i])
test.x=cbind(rep(1,nrow(test.x)),test.x)
```

Plot

```
plotRGB(train, r=4,g=3,b=2)
plot(train$train_20170422_1,color=train$train_classes)
```





##Functions

1: Sigmoid Function:

Define the active fuction to trainsform input x into Logistic regression model

```
sigmoid=function(z)
{
  g=1/(1+exp(-z))
  return(g)
}
```

2:Logicost Function

Calculate the logistic regression Loss value

```
logicost=function(theta,x,y)                                    ##Theta is coeffecient
s for x
{
  if(is.null(nrow(x))==TRUE){m=1
  }else{
    m=nrow(x)}
  g=sigmoid(x%*%theta)
  J=(1/m )*(t(-y)%*%log(g)-t(1-y)%*%log(1-g))                  ## Logistic Loss Functi
on
  return(J)
}
```

3: SDGgradient Function

Random sample the data and choose the sample data for gradient update to converge

```
SGDgradient=function(x, y, alpha,iter)                         ## alpha: learning rate;
iter: iteration time
{
  result=foreach (i = 1:iter,.combine = c)%do%    ## Use "foreach" instead of "for" l
oop to parallel processing
  {
    m=nrow(x)                                                  ## m is the sample size
of input x
    k=sample(1:m,1)
    g=sigmoid(x[k,]%*%theta)
    J=logicost(theta,x[k,],y[k])
    value=list("J"=J,"theta"=theta)
    m_new=1                                              ## Since SGD is ramdom select
one sample, so sample size of new vairable is 1
    grad=x[k,]%*%(g-y[k])
    theta=theta-(alpha/m_new)*grad                       ## Gradient update for each st
ep
    return(value)
  }
  return(result)

}
```

4: oneVSall Function:

Since logistic regression usually deal with binary classification, in order to build multi-class model, I use one VS all algorithms to generate a new “binary”(set the one class as 1, the other as 0) response each time for logistic regression

```
onevsallpredict=function(x,y,alpha,iter)
{
  classes=unique(y)
  result=foreach(i=1: length(classes),.combine = c)%do%       ## Use "foreach" to do
parallel processing
  {
    y_new=ifelse(y==classes[i],1,0)   ## Create new response data, set current class
as 1, the other class is 0
    value=SGDgradient(x,y_new,alpha,iter)
    value_J=foreach(i=1:(length(value)/2),.combine = cbind)%do%{value[[1+(i-1)*2]]}
## Extract Loss value
    value_theta=foreach(i=1:(length(value)/2),.combine = cbind)%do%{value[[2+(i-1)*2]
]}   ## Exract theta value
    all_coef=value_theta[,which.min(value_J)]
    result2=list("Jcost"=value_J,"all_coef"=all_coef)
    return(result2)
  }
  ## The dimension of all Loss is(# of class * # of predictors) which save all the Lo
ss
  all_Jcost=foreach(i=1:(length(result)/2),.combine = rbind)%do%{result[[1+(i-1)*2]]}
  all_coef=foreach(i=1:(length(result)/2),.combine = rbind)%do%{result[[2+(i-1)*2]]}
## All the theta value
  H=sigmoid(x%*%t(all_coef))
  y_predict=max.col(H)                                      ## Classify each value by usi
ng the largest probability
  y_predict2=matrix(0,nrow=length(y_predict))
  for(i in 1:length(y_predict))                    ## Convert the response value index
to response classify label
  {
    y_predict2[i]=classes[y_predict[i]]
  }
  error=sum(ifelse(y_predict2!=y,1,0))/length(y)                                    #
# Calculate error
  value=list("y_predict"=y_predict2, "error"=error,"All_Jcost"=all_Jcost)
  return(value)
}
```

5 Learning rate Function

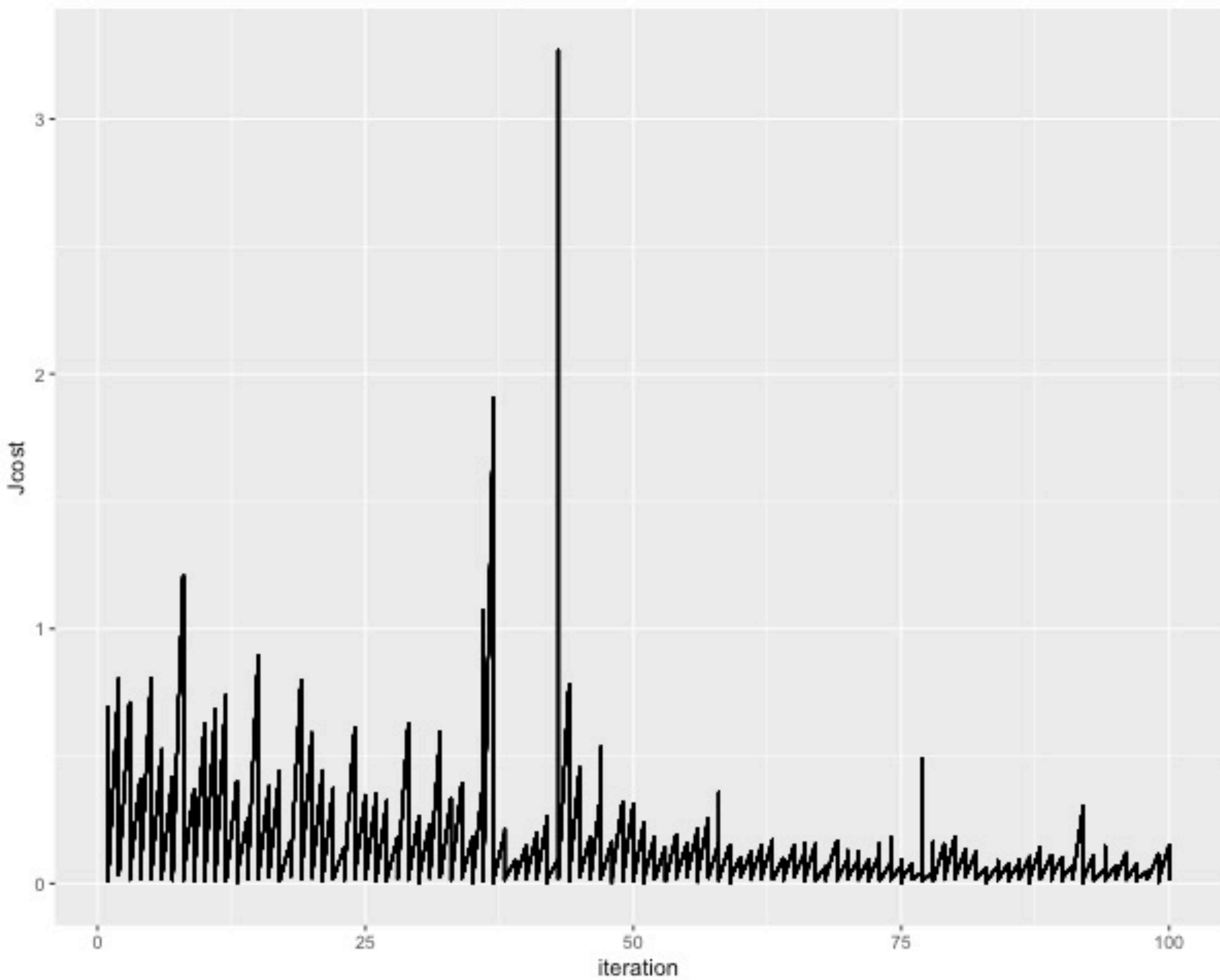
learning rate is very important in model fitting. If learning rate is too large, it is harder to find the global minimum since it may bounce around the global minimum; if learning rate is too small, it will take long time to approach global minimum.
In this case, I use small sample from trainning data to fit different learning rates and plot the change of Loss for picking the best learning rate.

```
find_learningrate=function(x,y)
{
  theta=as.matrix(rep(0,ncol(train.x)))
  alpha=c(0.1,0.01,0.001,0.0001,0.00001)            ## Set the learning rate from 0.1 to 0.00001 for interate
for (i in 1:length(alpha))
{
  value=onevsallpredict(x,y,alpha[i],500)                        ## Set iteration time as 500
  data=t(value$All_Jcost) %>% as.data.frame()                    ## Extract Loss value from result
  classes=unique(y)
  name=foreach(k=1:length(classes),.combine = c)%do%{name=paste("class",as.numeric(classes[k]))}
  colnames(data)=name
  data$index=rep(1:100)
  ## Create Plot of Loss change for each learning rate
  p=ggplot(data)+ggtitle( paste("Logistic cost of 30 classes for Learning rate =",  as.numeric(alpha[i]) ))+
    xlab("iteration")+ylab("Jcost")
  for(j in 1:30)
  {
    p=p+geom_line(aes(y=data[,j], x= index))
  }
  mypath3 <- file.path("/Users/admin/Desktop/GDR","result",paste("cost_error for alpha equal", as.numeric(alpha[i]), ".jpg", sep = "_"))                  ## Automated saved plot in local directory
  jpeg(file=mypath3, width = 600, height = 500)
  print(p)
  dev.off()
  print(i)
}
}
```

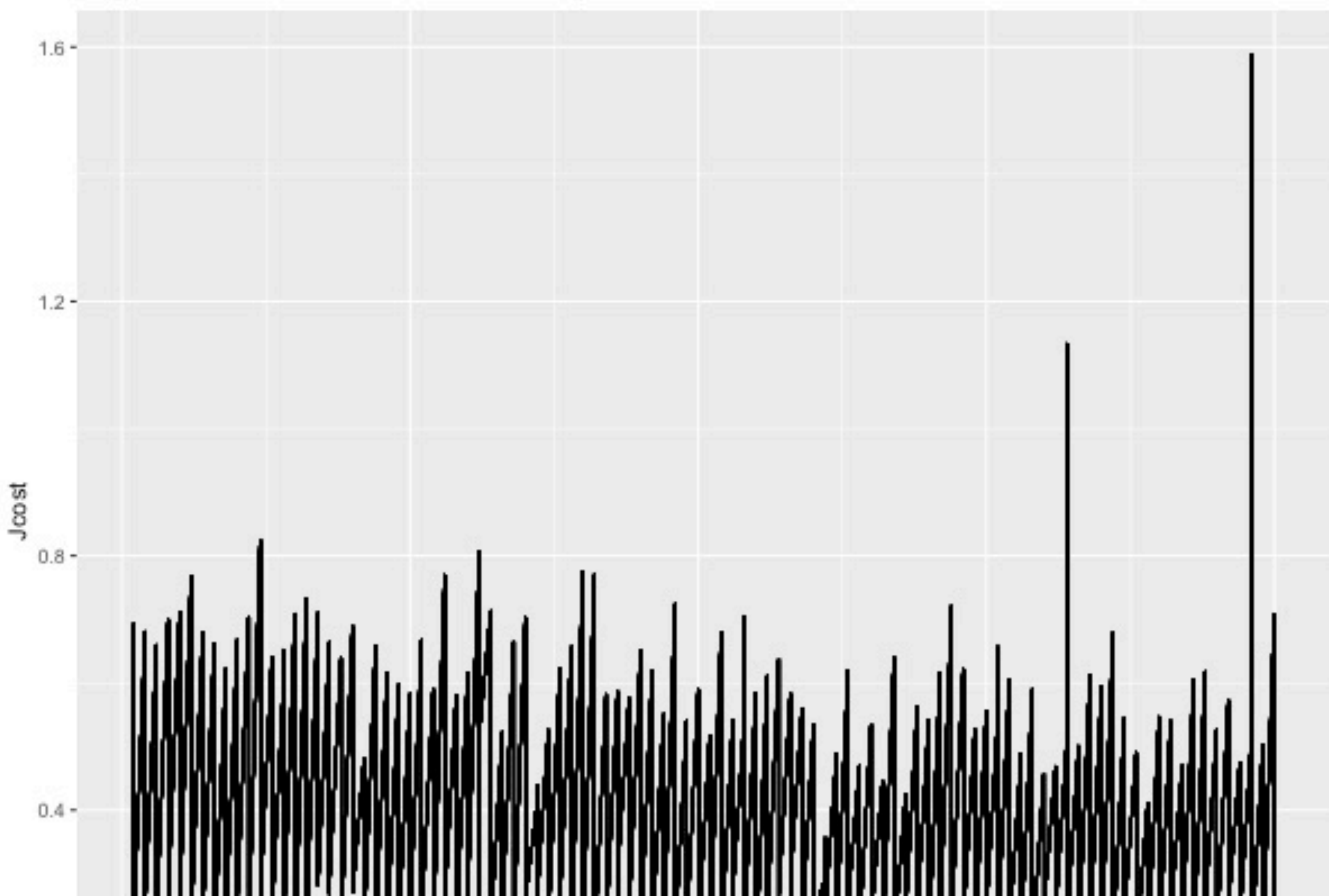
Sample data and select learning rate

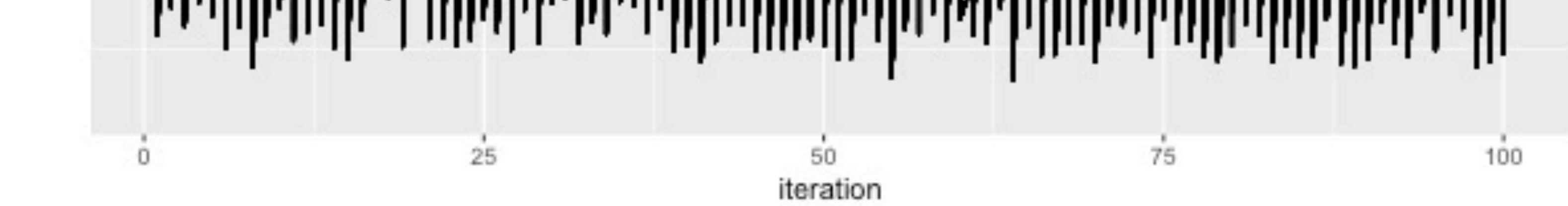
```
set.seed(3)
sample_index=sample(1:nrow(train.x),nrow(train.x)/10)       ## Choose sample data for choosing the learning rate
x_sample=train.x[sample_index,]
y_sample=train.y[sample_index,]
theta=as.matrix(rep(0,ncol(train.x)))
find_learningrate(x_sample,y_sample)
```

Logistic cost of 30 classes for Learning rate = 0.1

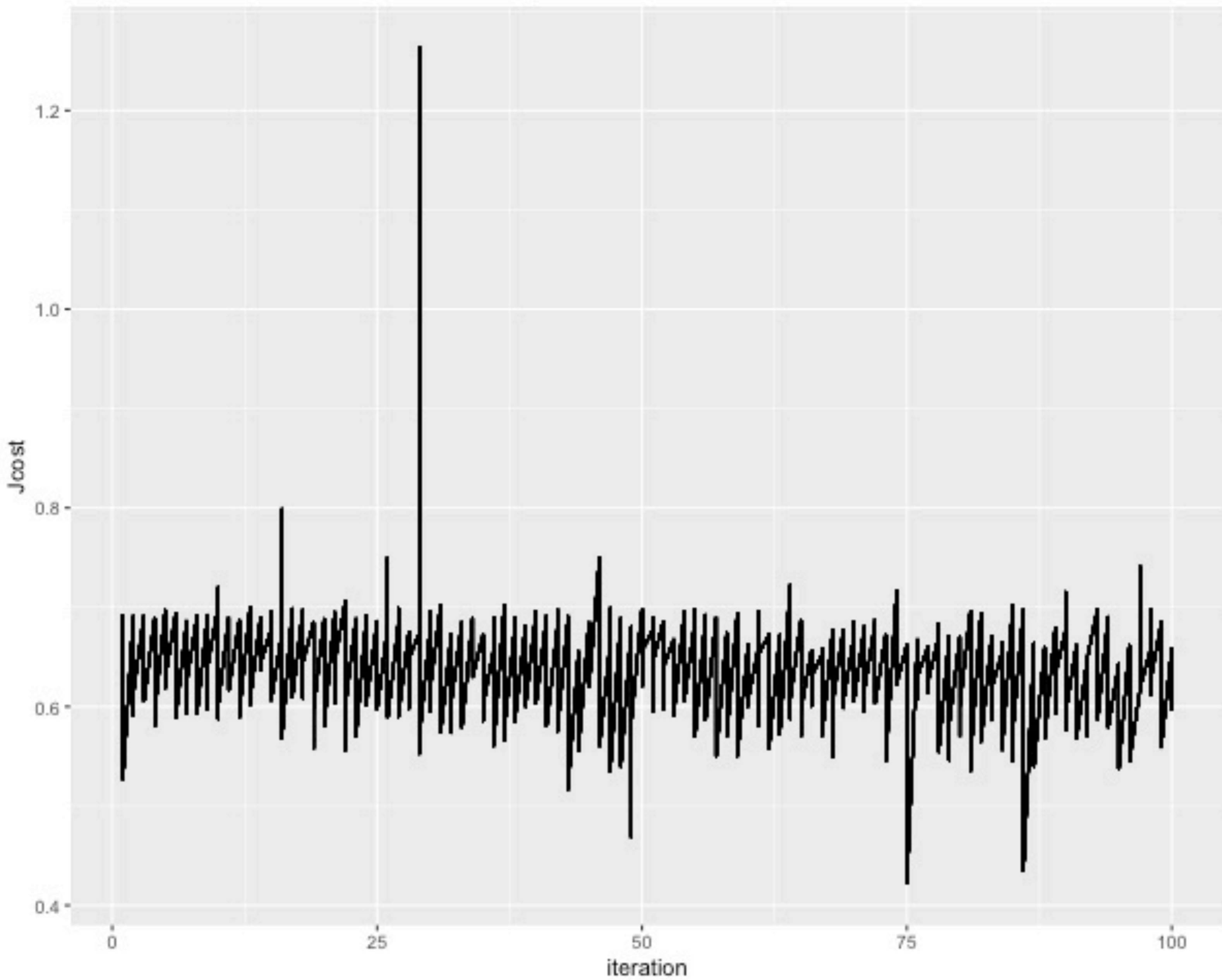


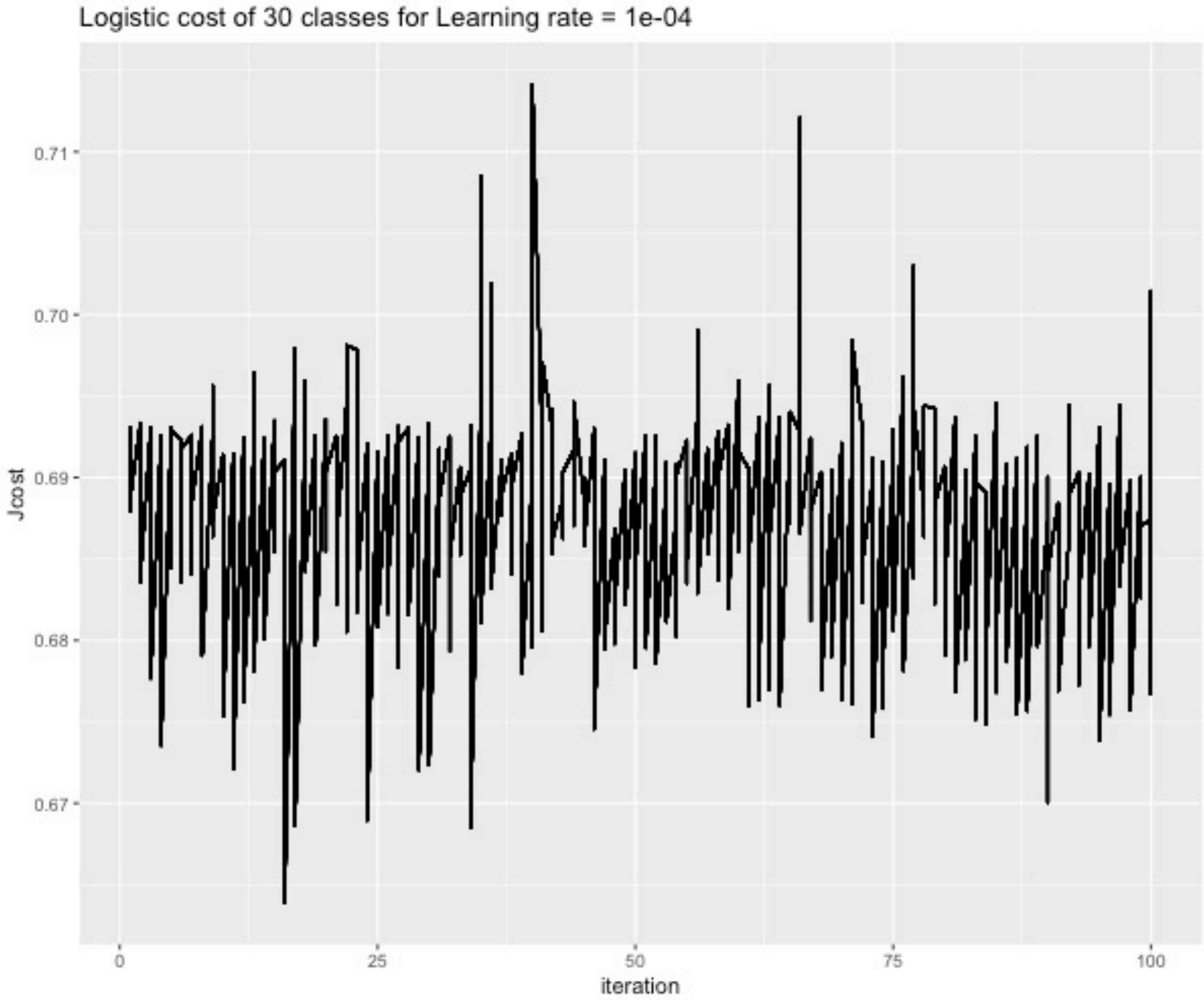
Logistic cost of 30 classes for Learning rate = 0.01



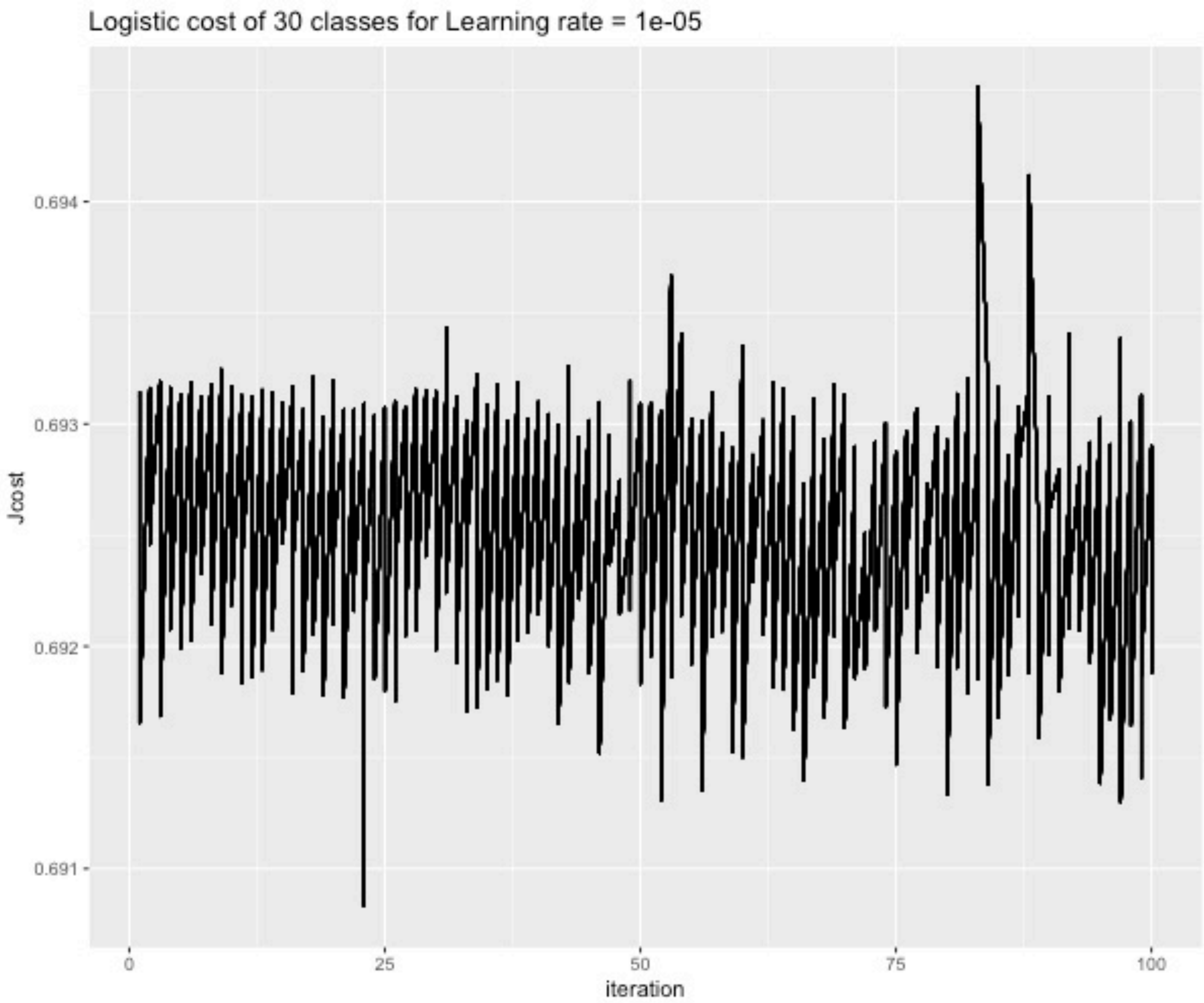


Logistic cost of 30 classes for Learning rate = 0.001





Logistic cost of 30 classes for Learning rate = 1e-04
0.71
0.70
0.69
0.68
0.67
Jcost
0
25
50
75
100
iteration



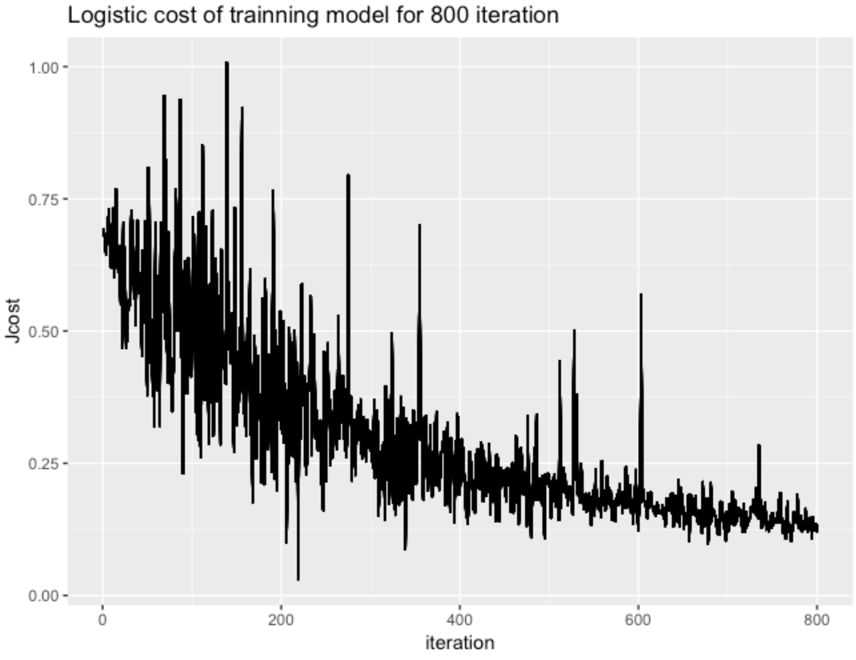
The learning rate of Stochastic Gradient Descent is noisy because SGD algorithm randomly select sample from data, so the Loss is not gradually decrease. From the plot above, the learning rate with 0.01 has the best trend for Loss, so I choose learning rate as 0.01 for logistic model

Model fitting in training

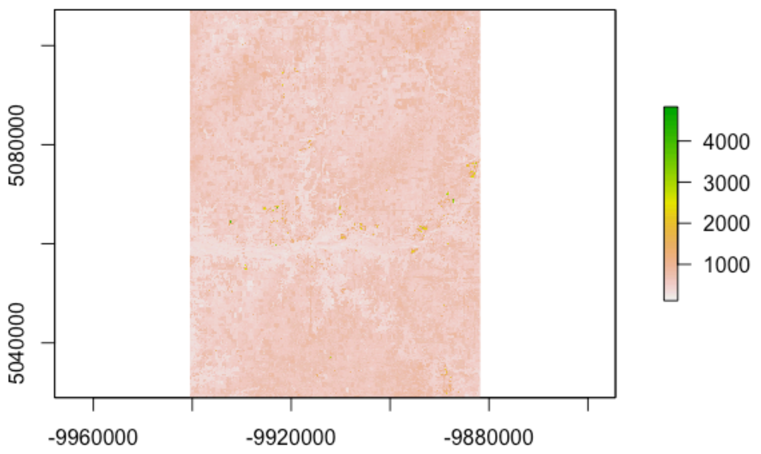
```
##innitialze
theta=as.matrix(rep(0,ncol(train.x)))
alpha=0.01
iter=1000
logimodel=onevsallpredict(train.x,train.y,alpha,iter)

model.error1=logimodel$error ##0.4693343   100iter  ##0.3612574 1000 iter
model.all_cost1=logimodel$All_Jcost
model.ypredict=logimodel$y_predict
```

Plot of training error

```r
data=t(model.all_cost1) %>% as.data.frame()
classes=unique(train.y)
name=foreach(k=1:length(classes),.combine = c)%do%{name=paste("class",as.numeric(clas
ses[k]))}
colnames(data)=name
data$index=rep(1:800)
p=ggplot(data)+ggtitle( "Logistic cost of trainning model for 1000 iteration ")+
  xlab("iteration")+ylab("Jcost")
for(j in 1:30)
{
  p=p+geom_line(aes(y=data[,j], x= index))
}
p
```



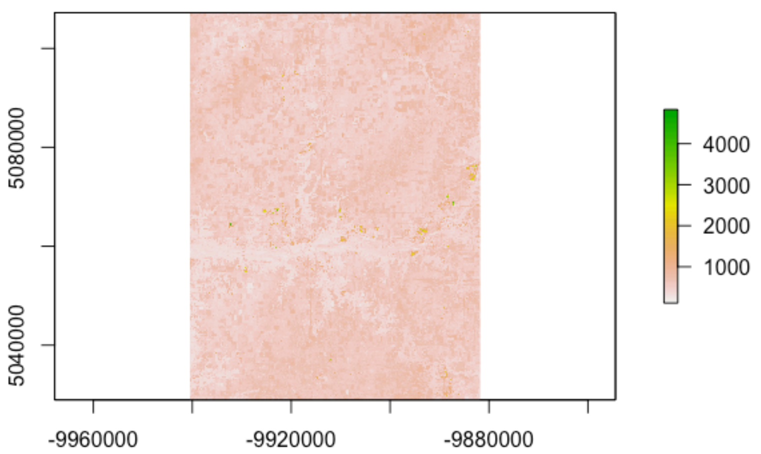
Plot of train error

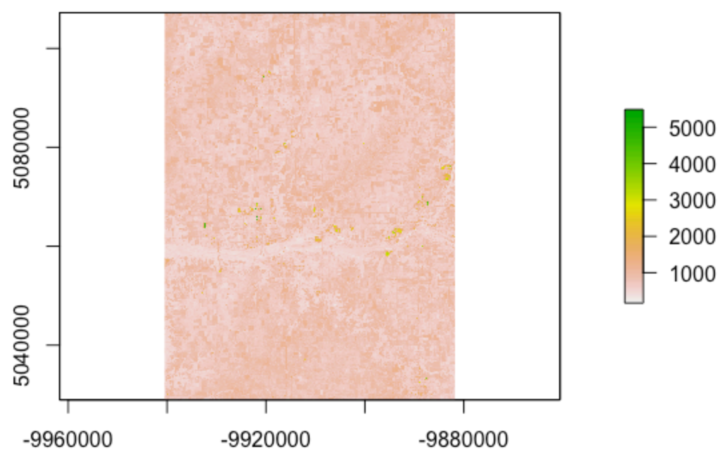


Plot of predict result

left is the original classfication, right is the predict result

```
plot(test$train_20170422_1,color=train$train_classes)
plot(test$train_20170422_1,color=logimodel_train$y_predict)
```





From the model trainning, the result shows the traing erro is 0.47 when iteration is 500, training error is 0.36 when iteration time is 1000 which means the trainning is 74% Accuracy, and when the iteration time becomes larger, the trainning error may decrease until converge to global minimum and then hit the highest accuracy.

Model testing in test data

```
##test
logimodel_test=onevsallpredict(test.x,test.y,alpha,iter)
```

close parallel processing

```
##close parrel
stopCluster(cl)
```

Summary

The finnal prediction accuracy is for this model. The bigest benefit of this model is suitable for large data processing since it combine the Stochastic Gradient Descent for converging to global minimum, besides, this model use parallel processing to save the time for model processing.