Homework 5 Neural Network

Xiongming Dai

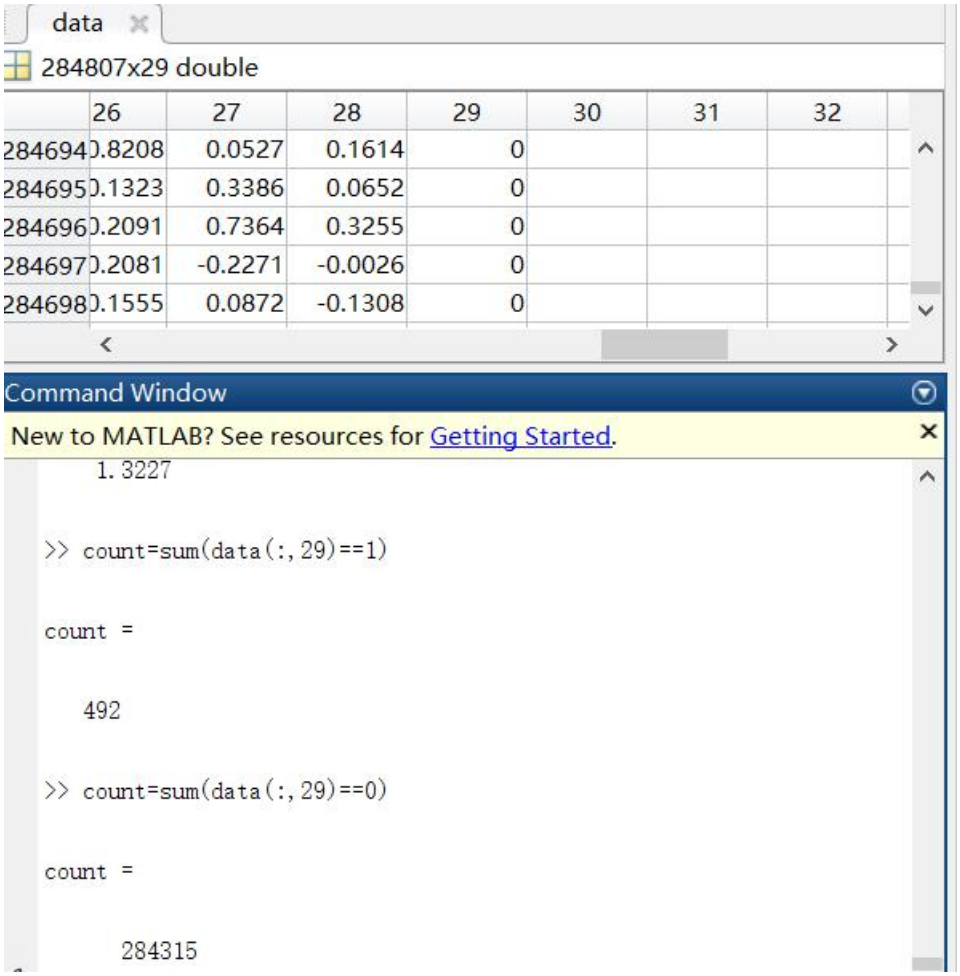
Due on April 10th , 2018

Problem 1:

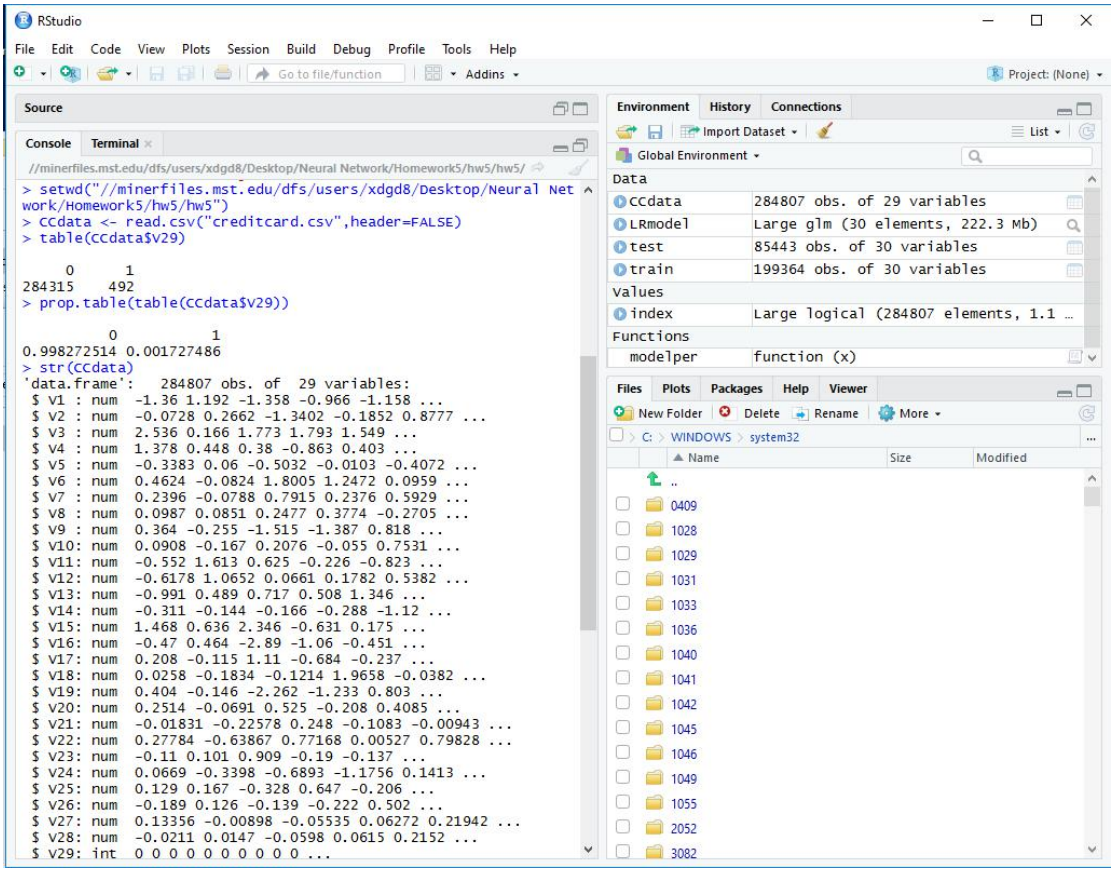
(1) As we can see from the following plot, there are 28 columns(V1,V2,…V28) expressing the mainly undefined features, the 29th column(V29) is the corresponding ground truth. And it takes value 1 in case of fraud and 0 otherwise.

This dataset presents transactions that occurred in this case, where we have 492 frauds out of 284,807 transactions and 284315 legal transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. Therefore, the data is skewed.

The result from the MATLAB workspace is show as follows:



The result from RStudio workspace is shown as follows:



The R code for this part(The line begins “#” used for notation):

```
#set up the working path of the code
setwd("//minerfiles.mst.edu/dfs/users/xdgd8/Desktop/Neural Network/Homework5/hw5/hw5")
CCdata <- read.csv("creditcard.csv",header=FALSE)  #extract the dataset
table(CCdata$V29)  #build a table to show
prop.table(table(CCdata$V29))
str(CCdata)
```

(2) “Under Sampling” were used for unbalanced data.

Method 1: I have referred to this article: “Calibrating Probability with Undersampling for Unbalanced Classification”. It is mainly based on K-means to extract important features or cases of transactions, then we obtain the equal distribution of both classes.

The R code for this part:

```
data=CCdata;    #copy the original dataset
#process the dataset for undersampling, mainly based on K-means, to extract the centre features
KMUS <- function(data, perc_maj = 50, perc_under = NULL, max_iter = 100L,
nstart = 10L, classes = NULL){
y <- data[[ncol(data)]]
class_levels <- levels(y)
if (is.null(classes)) classes <- extract_classes(y)
X_maj <- data[y == classes[["Majority"]], -ncol(data), drop = FALSE]
data_min <- data[y == classes[["Minority"]], , drop = FALSE]
maj_size <- nrow(X_maj)
min_size <- nrow(data_min)
#centre point extraction
num_centers <- maj_size - compute_undersample_size(majority_size = maj_size,
minority_size = min_size,
perc_maj = perc_maj,
perc_under = perc_under)
if (num_centers == maj_size) return(data)
X_centroids <- kmeans(X_maj,
num_centers = num_centers,
max_iter = max_iter,
nstart = nstart)$centers
data_centroids <- cbind(X_centroids, classes[["Majority"]])
colnames(data_centroids) <- colnames(data)
data <- rbind(data_centroids, data_min)
data[[ncol(data)]] <- factor(data[[ncol(data)]], levels = class_levels)
```

```
# output the reduced and balanced dataset to the original variable so that the following #train,valid and test can be implemented.

CCdata=data;
```

Method 2:

Using the function of R package, under-sampling reduces the number of observations from majority class to make the data set balanced:

The R code is shown as follows:

```
#sampling V29 for under sampling
trainUnder <- ovun.sample(V29~.,data = train,method = "under", p = 0.5)
CCdata <- trainUnder$data #extract dataset
table(CCdata$V29)    #make the table
```

(3)From (2),we got equally distributed classes of the dataset. Now, we will split the reduced data into train, valid and test datasets.

Generally, we will use 60% data as training, 20% data as the validated and 20% data as the tested. Sometimes, if we just need two processes: train and test, we might split the dataset into 70% data for training, the remaining for test.

The corresponding R code is shown as follows for this part:

```
library(caTools)    #introduce this library so that we can use the function
set.seed(999)
#split the dataset
index <- sample.split(CCdata$V29,SplitRatio = 60/100)
train <- CCdata[index,]    #for training
CCdata = CCdata[!index,]
index <- sample.split(CCdata$V29,SplitRatio = 50/100)
valid <- CCdata[index,]      #for validation
```

```
test <- CCdata[!index,]      #for test
train$V29 <- as.factor(train$V29)
valid$V29 <- as.factor(valid$V29)
test$V29 <- as.factor(test$V29)
```

Now train the neural network classifier, we choose the first input layer have 28 neuros and hidden layer have 20 neuros. The activation function is logistic function. Based on these, we do the training for the under sampled dataset.

The R code is shown as follows:

```
library(neuralnet) #introduce neural network package
train$fraud = trainset$factor == 1
train$nonfraud = trainset$factor == 0
Iteration=50;
#introduce logistic function model
LRmodel <- glm(V29~.,family = "binomial",data = train)
#build neural network with associated property and train
network = neuralnet(fraud + nonfraud ~ Sepal.Length + Sepal.Width + Petal.Length +
Petal.Width,train,LRmodel,hidden = 3,inputLayer=28,hiddenLayer=20,OutputLayer=1,iteration)
network$result.matrix
head(network$generalized.weights[[1]])
$result.matrix
network$result.matrix
library(neuralnet)
par(mfrow=c(2,2))
#validation
LRprobab <- predict(LRmodel,newdata = valid,type = "response")
LRpred <- ifelse(pred >= 0.5,1,0)
gwplot(network,selected.covariate = "Petal.Width")
gwplot(network,selected.covariate = "Petal.Length")
gwplot(network,selected.covariate = "Sepal.Length")
gwplot(network,selected.covariate = "Sepal.Width）
#test
net.predict = compute(network,test[-5])$net.result
net.prediction = c("fraud","nonfraud")[apply(net.predict, 1, which.max)]
predict.table = table(test$factor,net.prediction)
#show confusion Matrix
confusionMatrix(predict.table)
```

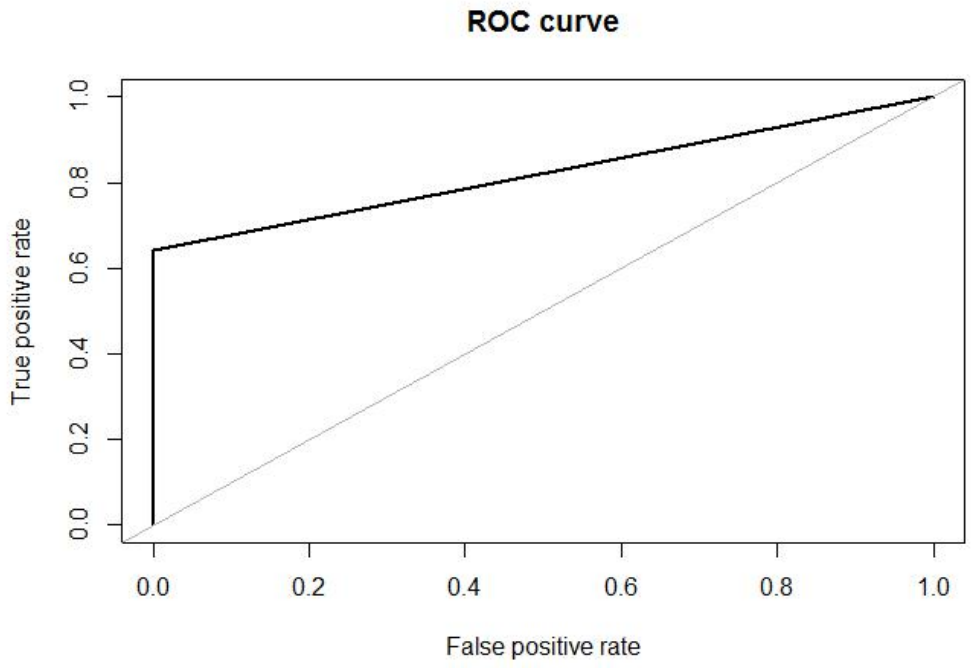
Calculate precision and recall using confusion matrix,also show the ROC curve.

The R code is as follows:

```
library(ROSE)
library(caret)
modelper <- function(x){
  list("Confusion Matrix" = confusionMatrix(test$V29,x)$table,"ROC" =    roc.curve(test$V29,x))
  }
Modelper(LRpred)
```

The result can be shown as follows:

```
## $`Confusion Matrix`
##                  Reference
## Prediction         0         1
##                  0 85278       17
##                  1       53       95
##
## $ROC
## Area under the curve (AUC): 0.821
```



As we can see, we obtain the confusion matrix. And the True Positive(TP)=85278, the False Positive(FP)=17.

The True Negative(TN) is 95 and False Negative(FN)=53.

Precision=TP/(TP+FP)=85278/(85278+17)=99.98%

Recall=TP/(TP+FN)=85278/(85278+53)=99.93%

(4)The precision represents the positive predictive value of the datasets, given the predicted condition is positive. The recall represents the true positive rate or sensitivity, given the condition of the positive dataset.

We notice that most of the non fraud observations were correctly classified but about 2/3 fraudulent observations were incorrectly classified. Our priority should be to classify as much fraudulent observations as possible because classifying non fraud as fraud is better than classifying fraud as non fraud.Therefore, The precision is more important.

(5)Use k-fold approach to check whether we can improve the precision, k equal to ten for our case:

The R code is as follows:

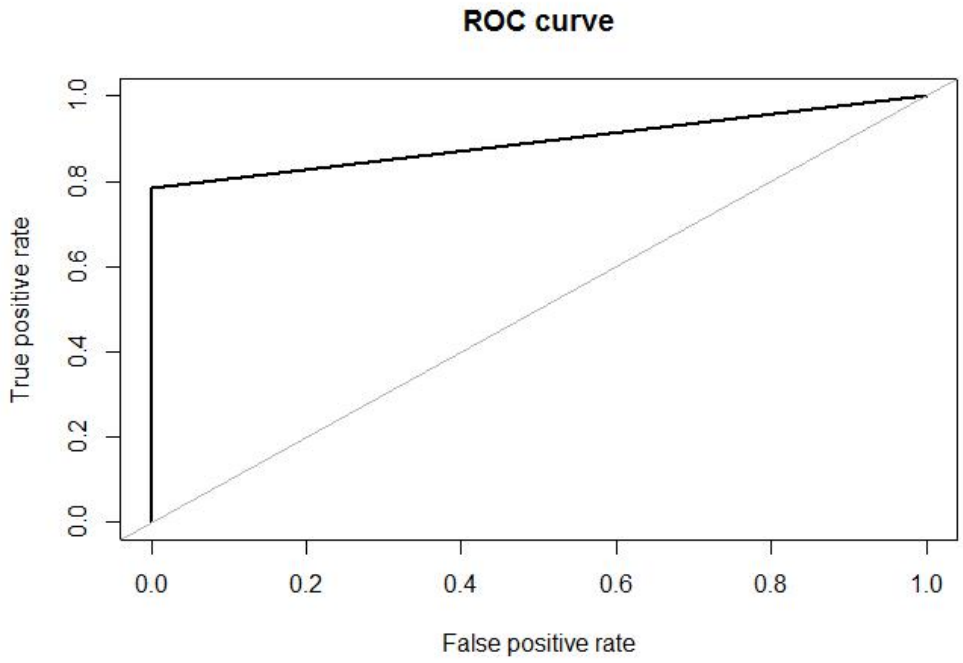
```
setwd("//minerfiles.mst.edu/dfs/users/xdgd8/Desktop/Neural Network/Homework5/hw5/hw5")

Data=CCdata    #copy reduced dataset
set.seed(17)
require(caret)
folds <- createFolds(y=data[,61],k=10)
library(pROC)
max=0
num=0
auc_value<-as.numeric()      #coding for 10-folds method
```

```
for(i in 1:10){
fold_test <- data[folds[[i]],]
fold_train <- data[-folds[[i]],]
fold_pre <- lm(as.numeric(V61)~.,data=fold_train)
fold_predict <- predict(fold_pre,type='response',newdata=fold_test)
auc_value<- append(auc_value,as.numeric(auc(as.numeric(fold_test[,61]),fold_predict)))
}
num<-which.max(auc_value)
print(auc_value)

fold_test <- data[folds[[num]],]
fold_train <- data[-folds[[num]],]
fold_pre <- lm(as.numeric(V61)~.,data=fold_train)
fold_predict <- predict(fold_pre,type='response',newdata=fold_test)
roc_curve <- roc(as.numeric(fold_test[,61]),fold_predict)
plot(roc_curve, print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),
grid.col=c("green", "red"), max.auc.polygon=TRUE,
auc.polygon.col="skyblue", print.thres=TRUE,main="ROC curve for the set with the largest AUC value")

#The result is shown as follows:

## $`Confusion Matrix`

##                   Reference

## Prediction         0         1

##                 0 85287         8

##                 1       32      116

##

## $ROC

## Area under the curve (AUC): 0.892
```

Precision=TP/(TP+FP)=85287/(85287+8)=99.99%

Therefore, It performs better than the previous model. Classifying 116 fraudulent observations correctly but still misclassifies 32 observations.

The ROC curve is show as follows:



(6)Tune the classification threshold and find best model using ROC curves. The threshold is updated as the built-in the parameter of updated weights. We will assign more weight to minority class as compared to majority class. For this purpose we will use the distribution of classes to determine weights. Our model is approximately best, after by changing many iterations and tuning the parameters, the precision is seemly stable for different sampling original dataset.

The R code is shown as follows:

```
model_weights <- ifelse(train$V29== 0,
                        nrow(train)/(2*table(train$Class)[1]),
                        nrow(train)/(2*table(train$Class)[2]))
table(model_weights)
## model_weights
## 0.500864234750276     289.773255813953
```

```
##                    199020                       344
```

```
LRWeighted <- glm(V29~.,data = train,family = binomial,weights = model_weights)
```

```
LRWeightedprob <- predict(LRWeighted,newdata = test,type = "response")
```

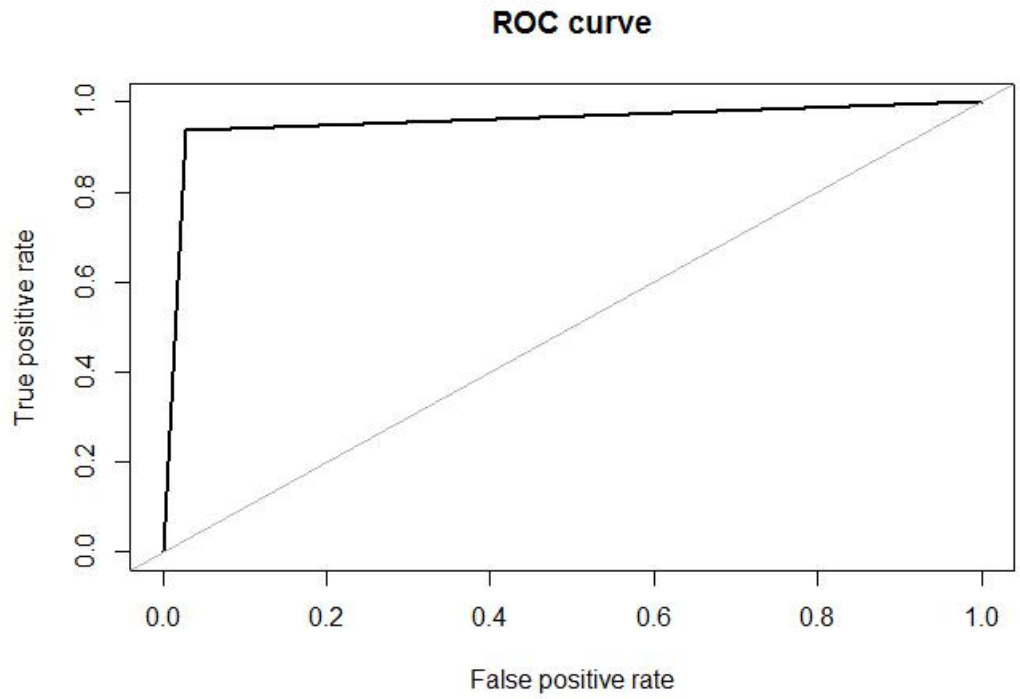
```
LRWeightedpred <- ifelse(LRWeightedprob > 0.5,1,0)
```

```
modelper(LRWeightedpred)
```

```
#The result is shown as follows:
```

```
## $`Confusion Matrix`
##                    Reference
## Prediction         0         1
##                  0 82987    2308
##                  1         9      139
##
## $ROC
## Area under the curve (AUC): 0.956
```

The ROC curve is shown as follows:



Thus we can see that non fraud observations were assigned a weight of 0.5 while fraud observations were assigned weight of 289.77. We get a decent outcome for both the classes but it can be improved.

(7)Use this model on complete dataset

The R code is shown as follows:

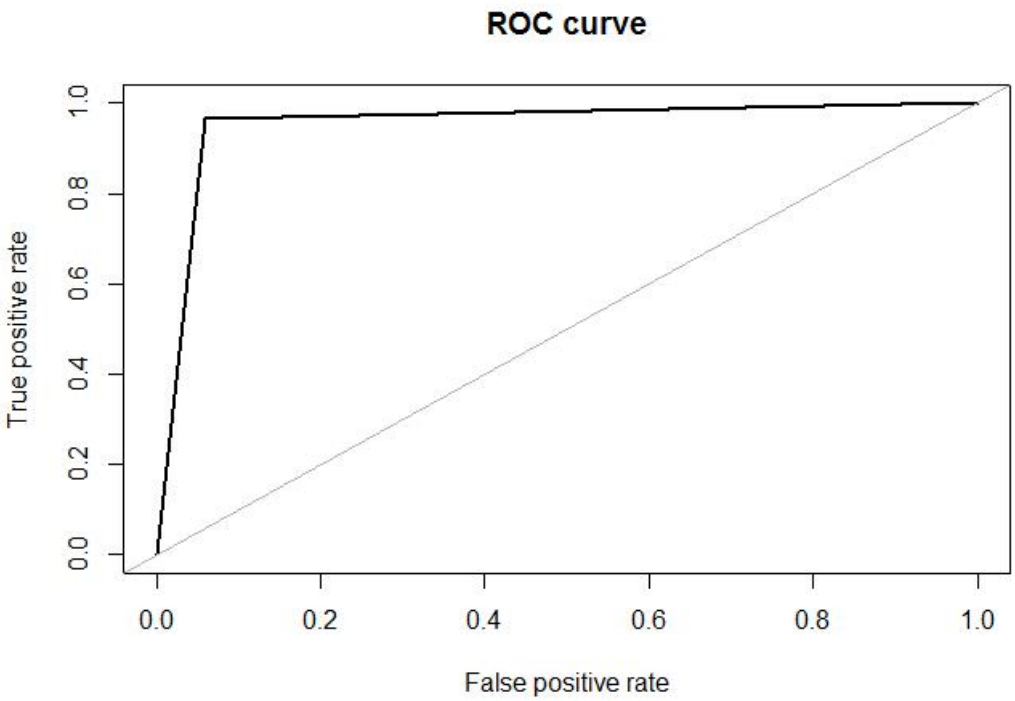
```
#set up the working path of the code

setwd("//minerfiles.mst.edu/dfs/users/xdgd8/Desktop/Neural Network/Homework5/hw5/hw5")
CCdata <- read.csv("creditcard.csv",header=FALSE)    #extract the dataset
library(caTools)
set.seed(999)
index <- sample.split(CCdata$V29,SplitRatio = 70/100)
train <- CCdata[index,]
# I think the validation can be removed here and test directly after training.
test <- CCdata[!index,]
#based V29 to split
train$V29<- as.factor(train$V29)
test$V29<- as.factor(test$V29)
library(ROSE)
library(caret)
model_weights <- ifelse(train$V29== 0,

                                    nrow(train)/(2*table(train$Class)[1]),

                                    nrow(train)/(2*table(train$Class)[2]))

table(model_weights)

LRWeighted <- glm(V29~.,data = train,family = binomial,weights = model_weights)

LRWeightedprob <- predict(LRWeighted,newdata = test,type = "response")

LRWeightedpred <- ifelse(LRWeightedprob > 0.5,1,0)

modelper(LRWeightedpred)

modelper <- function(x){
list("Confusion Matrix" = confusionMatrix(test$V29,x)$table,"ROC" =    roc.curve(test$V29,x))
  }
library(neuralnet)
train$fraud = train$factor == 1
train$nonfraud = train$factor == 0
LRmodel <- glm(Class~.,family = "binomial",data = train)
```

```
network = neuralnet(fraud + nonfraud ~ Sepal.Length + Sepal.Width + Petal.Length +
Petal.Width,train,LRmodel,hidden =
3,inputLayer=28,hiddenLayer=20,OutputLayer=1,iteration=50)
network$result.matrix
head(network$generalized.weights[[1]])
$result.matrix
network$result.matrix
library(neuralnet)
par(mfrow=c(2,2))
gwplot(network,selected.covariate = "Petal.Width")
gwplot(network,selected.covariate = "Petal.Length")
gwplot(network,selected.covariate = "Sepal.Length")
gwplot(network,selected.covariate = "Sepal.Width）
net.predict = compute(network,test[-5])$net.result
net.prediction = c("fraud","nonfraud")[apply(net.predict, 1, which.max)]
predict.table = table(test$V29,net.prediction)
  predict.table
confusionMatrix(predict.table)
## $`Confusion Matrix`
##                   Reference
## Prediction         0         1
##                 0    85357    2410
##                 1       12      148
##
## $ROC
## Area under the curve (AUC): 0.951
```

Precision=TP/(TP+FP)=85357/(85357+2410)=97.25%

From the confusion matrix, we find the AUC is 0.951 and the ROC

expressing the performance is shown as follows:


ROC curve
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0