QUANTITATIVE METHODS IN RETAIL FINANCE 2019 COURSEWORK 1

Alexander Pinches

30 January 2019

Initialise

First we load the required libraries and the data set itself. For convenience we set a seed for the random number generator to allow the results to be replicated subsequent runs.

```
#required libraries
library("dplyr")
library("nnet")
library("PRROC")
#load data
load("CCfraud.RData")
#set seed to keep same results for each run
set.seed(512)
```

Data analysis

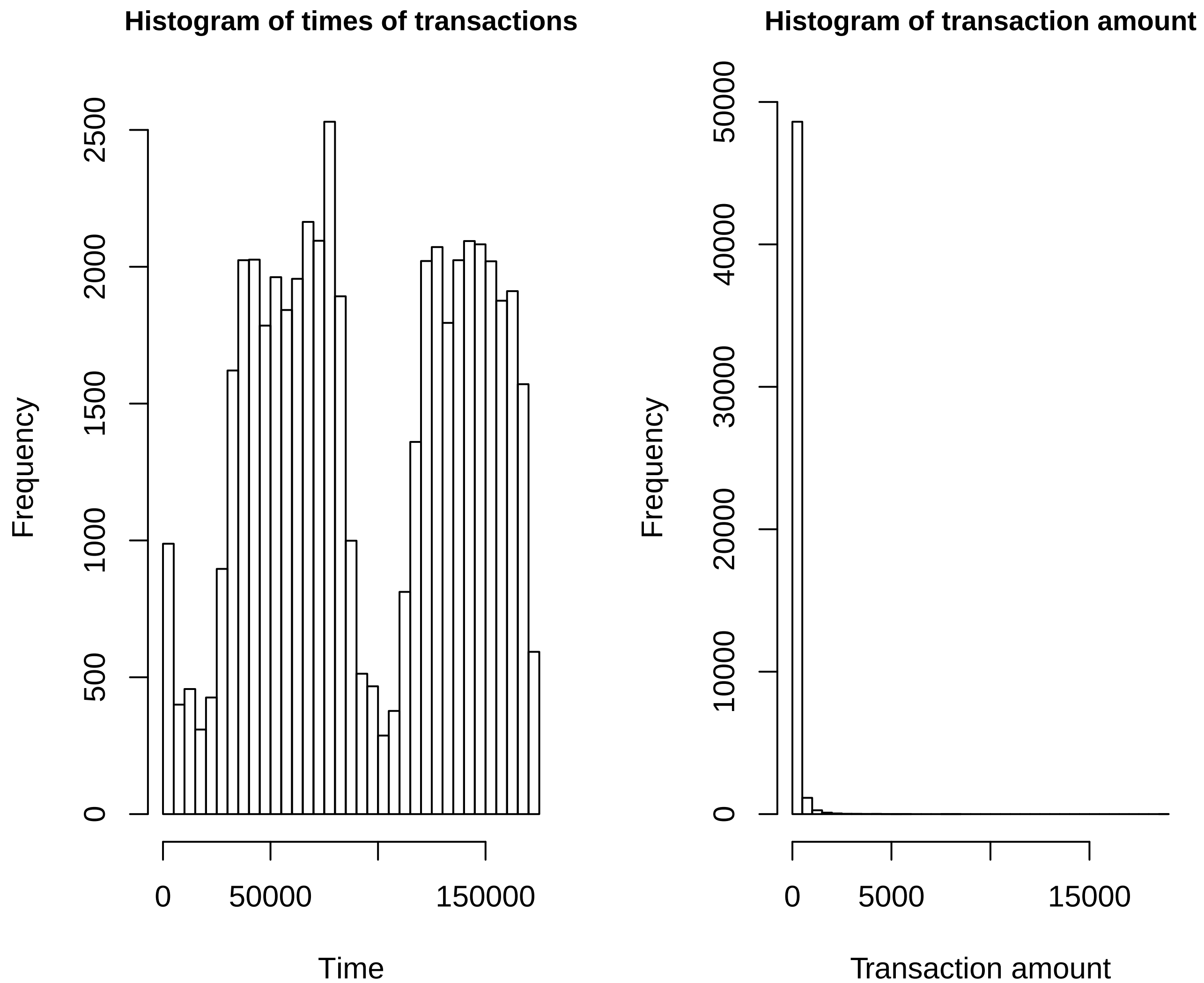
To explore the data set we can use `summary()` to see summary statistics of `Time`,`Amount` and `Class` such as their mean and range.

```
#summarise time amount class
summary(D2[,c(1,30,31)])
```

```
##       Time            Amount             Class
##  Min.   :     2   Min.   :    0.00   Min.   :0.000000
##  1st Qu.: 54242   1st Qu.:    5.49   1st Qu.:0.000000
##  Median : 84428   Median :   21.94   Median :0.000000
##  Mean   : 94452   Mean   :   88.28   Mean   :0.009792
##  3rd Qu.:139052   3rd Qu.:   78.36   3rd Qu.:0.000000
##  Max.   :172792   Max.   :18910.00   Max.   :1.000000
```

We see `Amount` has a low average but a high range so is heavily skewed with the mean outside the interquartile range. `Time` appears not to be skewed heavily as we would expect given that the range of times spans about 2 days. We would expect a similar number of transactions from day to day. `Class` is almost all non-fraudulent transactions as its mean is very close to zero. If we plot histograms of `Amount` and `Time` we can see how they are distributed. We need not plot a histogram of `Class` as it is binary.

```
#plot histograms of time and amount
par(mfrow=c(1,2), cex.main=0.9)
hist(D2[,1],main = "Histogram of times of transactions",
     xlab = "Time", breaks=50)
hist(D2[,30],main = "Histogram of transaction amount",
     xlab = "Transaction amount", breaks=50)
```



As suggested above we see that transaction `Amount` is heavily skewed with mostly small transactions and a few very large ones. We see with `Time` two large peaks late in each day this is likely due to people finishing work and then going shopping. \ If we show the number of fraudulent transactions indicated by a one against non-fraudulent twos we see most of the transactions are legitimate with a very small proportion fraudulent so the data set is imbalanced.

```
# calculate numbers of each class
D2 %>% group_by(Class) %>% tally()
```

```
## # A tibble: 2 x 2
##   Class     n
##   <int> <int>
## 1     0 49755
## 2     1   492
```

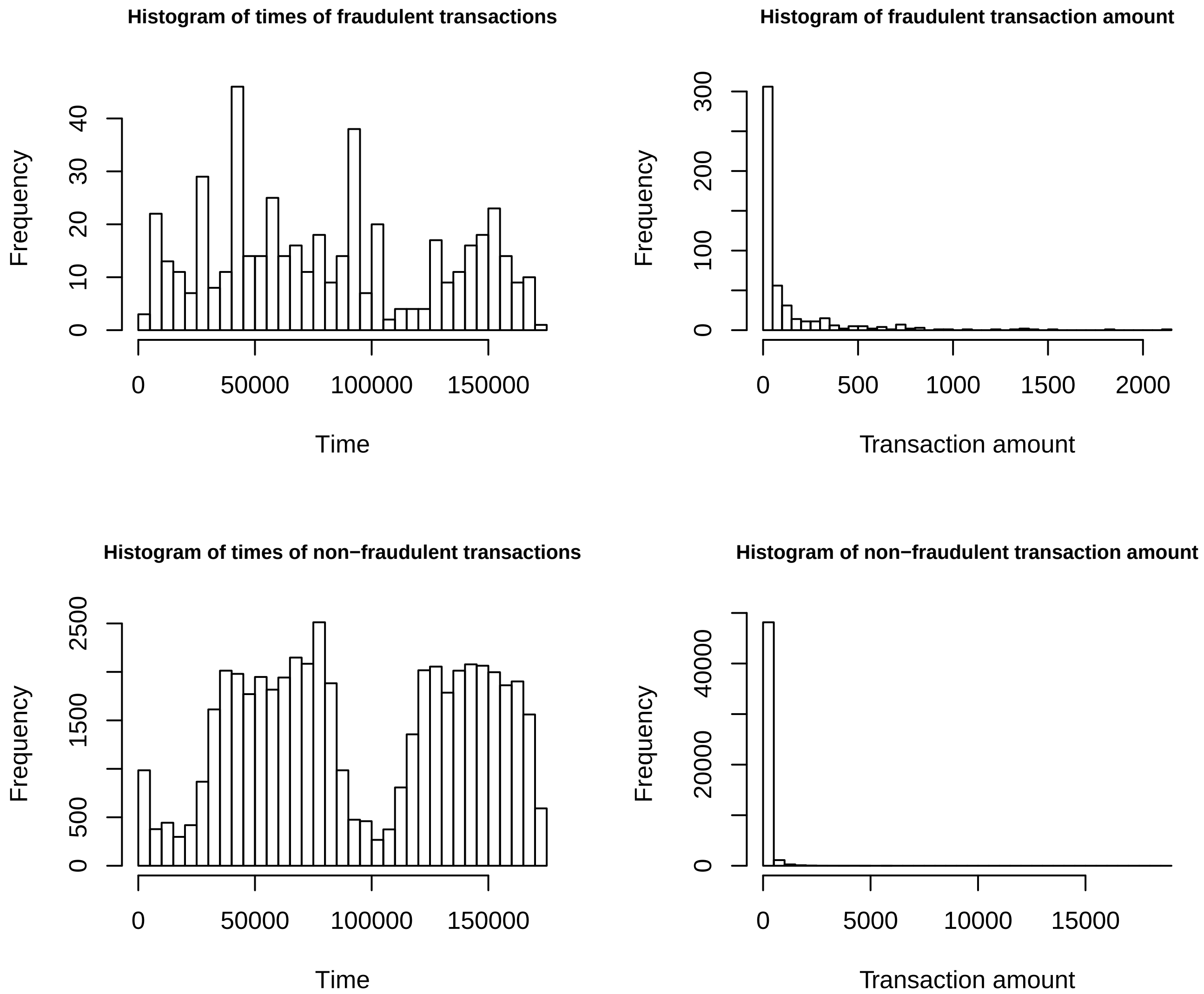
```
# print proportions of each class
sprintf("Proportion of fraudulent transactions:%s",sum(D2$Class==1)/nrow(D2))
```

```
## [1] "Proportion of fraudulent transactions:0.00979162935100603"
```

If we plot histograms of `Amount` and `Time` we can see whether they are distributed differently and thus they have some predictive value in detecting fraud. We see that fraudulent transactions tend to be less skewed in `Amount` tending to be smaller and we see that fraudulent transactions occured more in the first day than the second and are more spread out through each day. So they could help us determine a fraudulent transaction.

```
#plot histograms of amount and time given class
par(mfrow=c(2,2), cex.main=0.8)
```

```
hist(D2[which(D2$Class == 1),1],main = "Histogram of times of fraudulent transactions",
     xlab = "Time",breaks = 50)
hist(D2[which(D2$Class == 1),30],main = "Histogram of fraudulent transaction amount",
     xlab = "Transaction amount",breaks = 50)
hist(D2[which(D2$Class == 0),1],main = "Histogram of times of non-fraudulent transactions",
     xlab = "Time",breaks = 50)
hist(D2[which(D2$Class == 0),30],main = "Histogram of non-fraudulent transaction amount",
     xlab = "Transaction amount",breaks = 50)
```



Creating training and testing data

We randomly split the data set $\frac{2}{3}$ into the training data set and the rest into the test data set and we check that the proportions of fraudulent transactions in each is similar. We see that the training dataset has about twice the amount of fraudulent transactions and is twice the size of the test set so we see that they therefore have similar proportions of fraudulent transactions.

```
#randomly select a training and test sample indices
trainSample=sample(1:nrow(D2),nrow(D2)*2/3,replace = F)
testSample=setdiff(1:nrow(D2),trainSample)

#create training and test data sets
trainD2=D2[trainSample,]
```

```
testD2=D2[testSample,]

#check proportion of fradulant transactions in each
trainD2 %>% group_by(Class) %>% tally()
```

```
## # A tibble: 2 x 2
##   Class     n
##   <int> <int>
## 1     0 33175
## 2     1   323
```

```
testD2 %>% group_by(Class) %>% tally()
```

```
## # A tibble: 2 x 2
##   Class     n
##   <int> <int>
## 1     0 16580
## 2     1   169
```

We then save the training and test datasets and normalise them so we can train our models faster and also improve their performance.

```
#save training and test datasets
save(trainD2,file = "trainD2.RData")
save(testD2,file = "testD2.RData")

#create a normalising function
normalise <- function(x) (x-min(x))/(max(x)-min(x))

#scale the datasets
scaled_train <- as.data.frame(apply(trainD2, 2, normalise))
scaled_test <- as.data.frame(apply(testD2, 2, normalise))
```

Training the ANN

After scaling our data we can test different neural network sizes choosing the best one based on which has the highest PRAUC. As when creating the model we assign random weights to the nodes first we could find a local minima instead of a global minima to increase our chances of finding this we train each neural network size 5 times. We save the model with the highest PRAUC and summarise the PRAUCs of every model we train below.

```
#create place to store PRAUCs and name columns
pr <- matrix(NA, nrow=5,ncol=5)
colnames(pr) <- c("5","10","15","20","25")
for (n in 1:5){# test 5 hidden layer sizes
  for (i in 1:5){# train each ANN 5 times to ensure we find a global not local maximum AUC
    model <- nnet(Class~.,data = scaled_train,size=n*5,maxit=1000,linout=T, trace=F)# create modek

    preds <- predict(model, newdata = scaled_test)# create predictions

    pr[i,n] <- pr.curve(scores.class0 = preds[scaled_test$Class==1],
                scores.class1 = preds[scaled_test$Class==0], curve = F)$auc.integral# store PRAUC's

    if (pr[i,n]==max(pr,na.rm=T)){# save model with best AUC
      best_model <- model
    }

  }
}
summary(pr)# summarise results
```

```
##         5                 10                15                20
```

```
##  Min.   :0.07401   Min.   :0.02028   Min.   :0.07003   Min.   :0.08577
##  1st Qu.:0.10165   1st Qu.:0.72516   1st Qu.:0.17986   1st Qu.:0.09594
##  Median :0.13328   Median :0.73483   Median :0.19900   Median :0.22118
##  Mean   :0.34784   Mean   :0.60330   Mean   :0.30728   Mean   :0.26998
##  3rd Qu.:0.63278   3rd Qu.:0.74572   3rd Qu.:0.38032   3rd Qu.:0.40348
##  Max.   :0.79744   Max.   :0.79050   Max.   :0.70721   Max.   :0.54352
##        25
##  Min.   :0.009151
##  1st Qu.:0.104458
##  Median :0.494872
##  Mean   :0.443142
##  3rd Qu.:0.793230
##  Max.   :0.814001
```

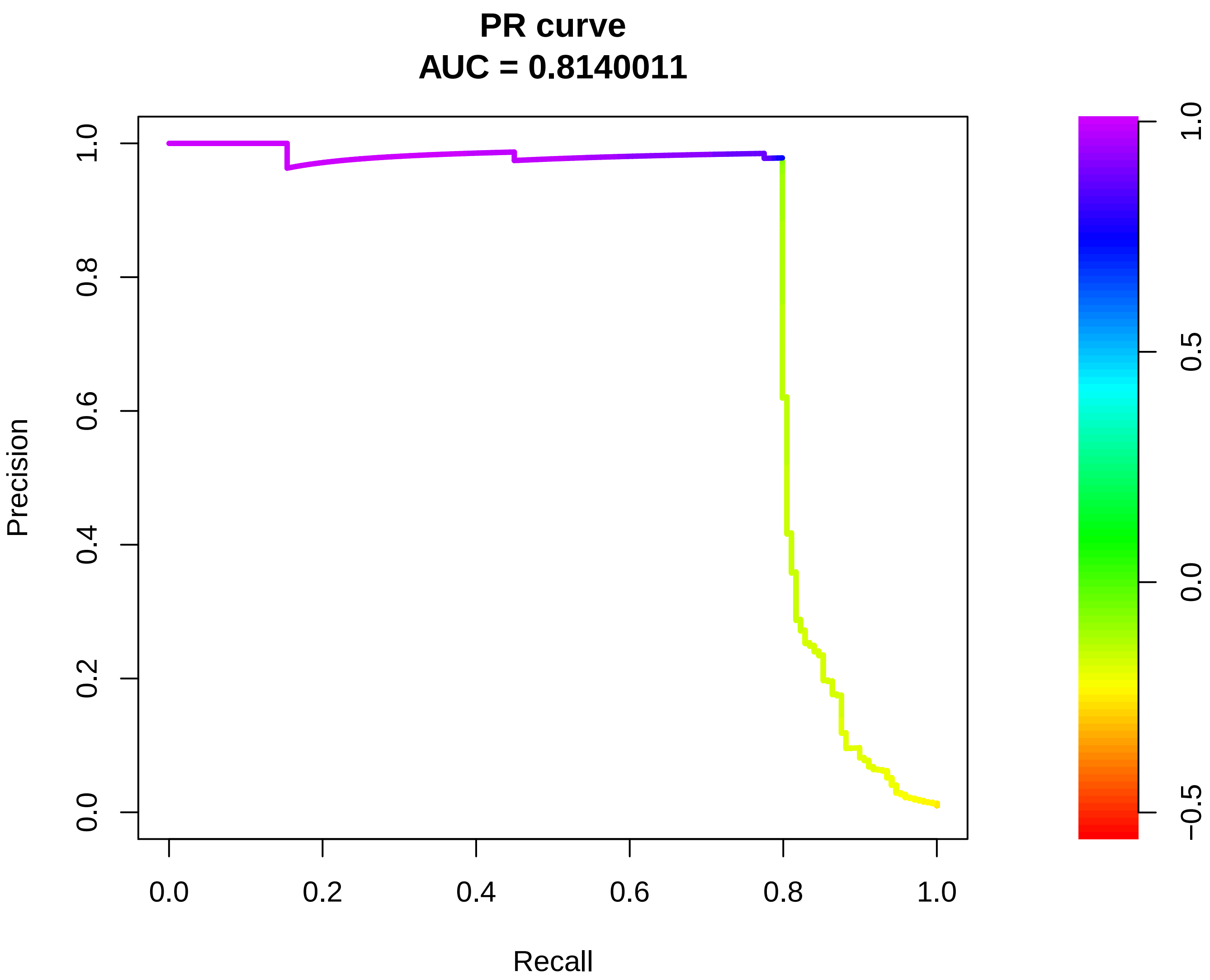
We see that on average 10 nodes performes best however the best performing model had 25 nodes. This is likely due to 25 node neural networks finding local minima or overfitting the training data. The 25 node model that performs best likely performs best because by having more nodes it allows the model to have a greater predictive ability at the risk of overfitting as it can pick up on more predictive features of the dataset. Overfitting when it identifies features which arent predictive of fraud.

PR curve

With the model with the highest PRAUC we plot its PR curve below.

```
preds <- predict(best_model, newdata = scaled_test)# create predictions using best model

best_pr <- pr.curve(scores.class0 = preds[scaled_test$Class==1],
                    scores.class1 = preds[scaled_test$Class==0], curve = T)# create pr curve
plot(best_pr)# plot it
```



We note that this is a good PR curve as the line is close to the top right hand corner as we want a high precision and recall and thus this curve has a high AUC.

Alarm rate

If we wanted to find the highest possible recall we can achieve for an alarm rate below 0.05%. We calculate the alarm rate at each point of the PR curve and find the point at which it passes 0.05% and the recall before this point is the highest recall we can achieve with this constraint. We use the recall and precision values calculated by the `pr.curve()` function and calculate p_0 directly from the test data set.

```
p0 <- sum(scaled_test$Class==1)/nrow(scaled_test)# calculate p0
alarm_rate <- p0 * best_pr$curve[,2]/best_pr$curve[,1]# calculate alarm rate
# print max recall for given alarm rate of 0.05%
sprintf("Highest recall for the given alarm rate %s",best_pr$curve[sum(alarm_rate<0.005),2])
```

```
## [1] "Highest recall for the given alarm rate 0.400584795321637"
```