QUANTITATIVE METHODS IN RETAIL FINANCE 2019 COURSEWORK 2

Alexander Pinches CID:01201653

February 2019

1 Introduction

We know by definition the kernel density estimator is as follows.

$$\hat{f}_p(\mathbf{x}; h) = \frac{1}{nh^m} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x_i}}{h}\right)$$

Substituting in our kernel function and rearranging we get.

$$\hat{f}_p(\mathbf{x}; h) = \frac{(2\pi)^{-\frac{m}{2}}}{nh^m} \sum_{i=1}^{n} \exp\left(\frac{-(\mathbf{x} - \mathbf{x_i}) \cdot (\mathbf{x} - \mathbf{x_i})}{2h^2}\right)$$

Taking the log of the kernel density estimator and adding and subtracting the constant $\varphi(\mathbf{x}; h)$ and rearranging we get.

$$\iff \log \hat{f}_p(\mathbf{x}; h) = c + \varphi(\mathbf{x}; h) - +\varphi(\mathbf{x}; h) + \log\left[\sum_{i=1}^{n} \exp\left(\frac{-(\mathbf{x} - \mathbf{x_i}) \cdot (\mathbf{x} - \mathbf{x_i})}{2h^2}\right)\right]$$

With the constant $c = \log\left(\frac{(2\pi)^{-m/2}}{nh^m}\right)$. We then put the negative $\varphi(\mathbf{x}; h)$ inside the log and using the properties of the exponential put it inside the exponential inside the sum to get the required result.

$$\iff \log \hat{f}_p(\mathbf{x}; h) = c + \varphi(\mathbf{x}; h) + \log\left[\sum_{i=1}^{n} \exp\left(\frac{-(\mathbf{x} - \mathbf{x_i}) \cdot (\mathbf{x} - \mathbf{x_i})}{2h^2} - \varphi(\mathbf{x}; h)\right)\right]$$

2 Proof of 3

$$\varphi(\mathbf{x}; h) \leq \log \hat{f}_p(\mathbf{x}; h) - c \leq \log n, \forall \mathbf{x}, h$$

Looking first at the left inequality.

$$\varphi(\mathbf{x}; h) \leq \log \hat{f}_p(\mathbf{x}; h) - c$$

Substituting in for $\log \hat{f}_p(\mathbf{x}; h)$ and rearranging we see we need to prove the following inequality.

$$\iff 0 \leq \log \left[\sum_{i=1}^{n} \exp \left(\frac{-(\mathbf{x} - \mathbf{x_i}) \cdot (\mathbf{x} - \mathbf{x_i})}{2h^2} - \varphi(\mathbf{x}; h) \right) \right]$$

We know from the properties of the log this is true for a real number if and only if the following holds.

$$\iff 1 \leq \sum_{i=1}^{n} \exp \left(\frac{-(\mathbf{x} - \mathbf{x_i}) \cdot (\mathbf{x} - \mathbf{x_i})}{2h^2} - \varphi(\mathbf{x}; h) \right)$$

We also know that the exponential function is strictly positive for real values. Also noting the definition of $\varphi(\mathbf{x}; h)$ for some $i \in \{1, 2...n\}$ say t

$$\exp \left(\frac{-(\mathbf{x} - \mathbf{x_t}) \cdot (\mathbf{x} - \mathbf{x_t})}{2h^2} - \varphi(\mathbf{x}; h) \right) = \exp(0) = 1$$

Therefore the sum must be greater than or equal to 1 by the properties of the exponential.

Looking at the second inequality.

$$\log \hat{f}_p(\mathbf{x}; h) - c \leq \log n$$

Similarly to before we sub in for the log kernel density estimator but we also put the $\varphi(\mathbf{x}; h)$ inside the log and simplify to get.

$$\iff \log \left[\sum_{i=1}^{n} \exp \left(\frac{-(\mathbf{x} - \mathbf{x_i}) \cdot (\mathbf{x} - \mathbf{x_i})}{2h^2} \right) \right] \leq \log n$$

As we know the exponential function is monotonically increasing we can take the exponential of each side to get.

$$\iff \sum_{i=1}^{n} \exp \left(\frac{-(\mathbf{x} - \mathbf{x_i}) \cdot (\mathbf{x} - \mathbf{x_i})}{2h^2} \right) \leq n$$

$$\frac{-(\mathbf{x} - \mathbf{x_i}) \cdot (\mathbf{x} - \mathbf{x_i})}{2h^2} \leq 0, \forall \mathbf{x}, h \iff \exp \left(\frac{-(\mathbf{x} - \mathbf{x_i}) \cdot (\mathbf{x} - \mathbf{x_i})}{2h^2} \right) \leq 1, \forall \mathbf{x}, h$$

therefore the sum must be less than n.

3 Implementing algorithm and its PR curve

First we load in the data and the PRROC package to use after creating the model.

```
library("PRROC") # import libraries
# load in data
load("C:/Users/Alex/OneDrive/Documents/Imperial␣Projects/Quant␣Finance/testD2.RData")
load("C:/Users/Alex/OneDrive/Documents/Imperial␣Projects/Quant␣Finance/trainD2.RData")
```

We impliment the algorithm in R and create predictions on the test and training data without the class column.

```
KDE_score <- function(x,h,train){ # create function to calculate score using given algorithm
  x <- as.matrix(x) #'set to matrix type
  train <- as.matrix(train) # set to matrix type
  n <- nrow(train) # calculate size of training dataset
  s <- nrow(x) # calculate size of test dataset
  score <- matrix(NA, nrow = s, ncol = 1) # create matrix to store scores
  for (t in 1:s){ # for each point in the test set
    phis <- matrix(NA, nrow = n, ncol = 1) # create matrix to store phis
    for (i in 1:n){
      phis[i] <- -(x[t,] - train[i,]) %*% (x[t,] - train[i,]) / (2 * h^2) # calculate phis
    }
    phi <- max(phis) # calculate max
    sum <- 0 # initialise sum
    for (i in 1:n){
      sum <- sum + exp(phis[i] - phi) # calculate sum
    }
    score[t] <- phi + log(sum) # store score
  }



  return(score) # return scores
}

preds <- KDE_score(testD2[,-31],0.1,trainD2[,-31]) # calculate scores


pr <- pr.curve(scores.class0 = preds[testD2$Class==1],
               scores.class1 = preds[testD2$Class==0], curve = T) # create pr curve
plot(pr) # plot it
```

We see that the PR curve in 1 is very poor with an AUC of just 0.042 as calculated by functions in the PRROC package.Looking at the curve this likely because the KDE can detect obvious cases of fraud but struggles with cases that are less obvious which is the case for most cases in this dataset causing precision to drop rapidly as recall increases. As precision is the number of true frauds predicted divied by the number of frauds predicted and recall is the number of frauds correctly predicted divided by the number of frauds.

4 Highest recall for a given alarm rate

For an alarm rate of 0.5% we use the script below and the results we plotted earlier to calculate the maximum recall we can achieve with this alarm rate.

```
p0 <- sum(testD2$Class==1)/nrow(testD2) # calculate p0
alarm_rate <- p0 * pr$curve[,2]/pr$curve[,1] # calculate alarm rate
# print max recall for given alarm rate of 0.05%
sprintf("Highest␣recall␣for␣the␣given␣alarm␣rate␣%s",pr$curve[sum(alarm_rate<0.005),2])
```

The maximum recall for the given alarm rate is 0.035. A very low recall. This indicates this model isn't a very good model for predicting fraud. This means if we don't want to annoy customers frequently with fraud alerts we will have to accept a lot of cases of fraud much more than the ANN did.

5 Comparison with ANN

We see the best ANN produced significantly better results having a PRAUC of 0.814 significantly higher than 0.042 and for the alarm rate of 0.5% a maximum recall of 0.401 much higher than 0.035. This greater performance is likely firstly because the neural network does not assume some underlying distribution where as the kernel density estimator here we've assumed a Gaussian distribution which may not be the case this gives the neural network more flexability. Also especially with neural networks with more nodes it can pick up on smaller details in the data where as in the KDE they are often lost. Neural networks also work better with data like this where the data is strongly unbalanced as fraud is very rare. However with the KDE it is easier to understand what exactly the algorithm is doing and explain how it works where as with a neural network its a black box and there's no clear way to see exactly why a model works. Also in the KDE all the predictors are assumed to give the same predictive ability, they are equally weighted. A ANN can as it is a supervised learning technique distinguish between useful predictors and weight them accordingly by changing the weights of nodes where as the KDE assumes that all the predictors have predictive ability of the same amounts (all equally weighted). Thus relies on their being 2 distinct clusters in the space where as in reality with most cases of fraud it isn't so distinct. This is likely why the KDE will determine obvious cases of fraud but as shown in its PR curve struggles with less obvious cases as we increase recall precision drops rapidly.
This makes the neural network the far superior approach with better performance and much faster to train. The only disadvantages are its hard to explain why a model thinks a transaction is fraudulent because its hard to interpret the model. So explaining to someone why the model works is a challenge. Also the neural network has more hyper-parameters to pick and test and also there's a risk that you'll find a local minimum when the model is training rather than the global minima. KDE with a different kernel may provide a better result than the one here but its unlikely it'll increase the performance to the level of the ANN.

6 PR Curve Plot

Below is the PR curve plotted from the KDE scores.

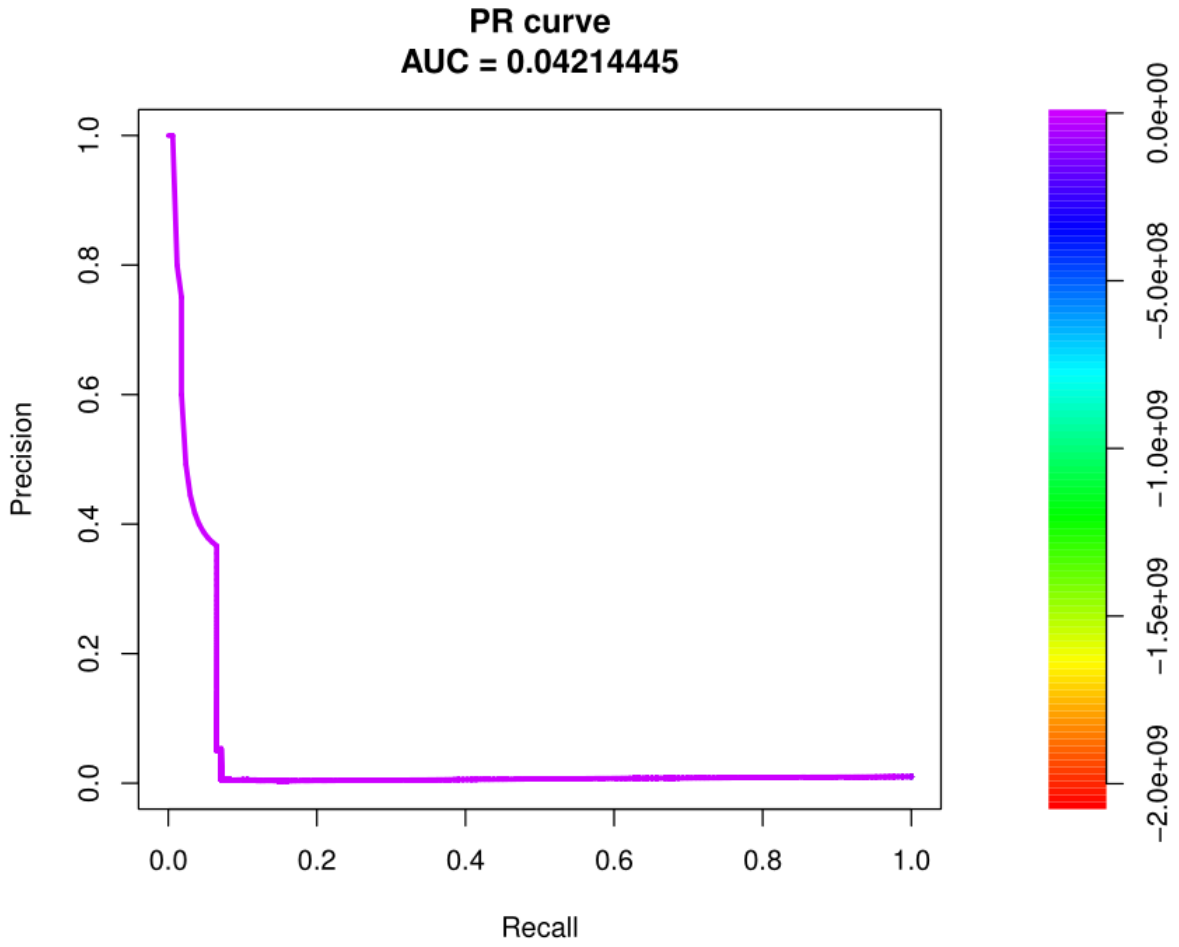

Figure 1: PR Curve for the KDE