Credit Card Fraud Detection

Pradeep Babburi

03/01/2017

1. Introduction

Credit card fraud identification through anomaly detection algorithm using multivariate Gaussian distribution. There are couple of reasons why I wanted to employ the anomaly detection algorithm over logistic regression or SVM. One, for the anomaly detection algorithm is well suited for highly skewed data like fraud detection and two that it is least biased towards posterior probabilites of the events. Another advantage is that it is relatively easy to train, infact there is not much training at all in this algorithm (other than finding the optimum threshold probability). However, it has to be noted that for the model to work well, the underlying features need to be independent.

2. Get Dataset

Loading the credit card transactional data into the R environmet using "fread" from data.table library. The data has 28 variables/features which are the principal components of the original data. The Class variable indicates if a transaction is fraud (positive case = 1) or genuine (negative case = 0).

```
X <- fread(input = "creditcard.csv", sep = ",", header = T, showProgress = T)
```

```
##
Read 91.3% of 284807 rows
Read 284807 rows and 31 (of 31) columns from 0.140 GB file in 00:00:03
```

```
X <- data.frame(X)
names(X)
```

```
##  [1] "Time"   "V1"     "V2"     "V3"     "V4"     "V5"     "V6"
##  [8] "V7"     "V8"     "V9"     "V10"    "V11"    "V12"    "V13"
## [15] "V14"    "V15"    "V16"    "V17"    "V18"    "V19"    "V20"
## [22] "V21"    "V22"    "V23"    "V24"    "V25"    "V26"    "V27"
## [29] "V28"    "Amount" "Class"
```

```
skew <- sum(as.numeric(X$Class))/nrow(X)
sprintf('Percentage of fraudulent transactions in the data set %f', skew*100)
```

```
## [1] "Percentage of fraudulent transactions in the data set 0.172749"
```

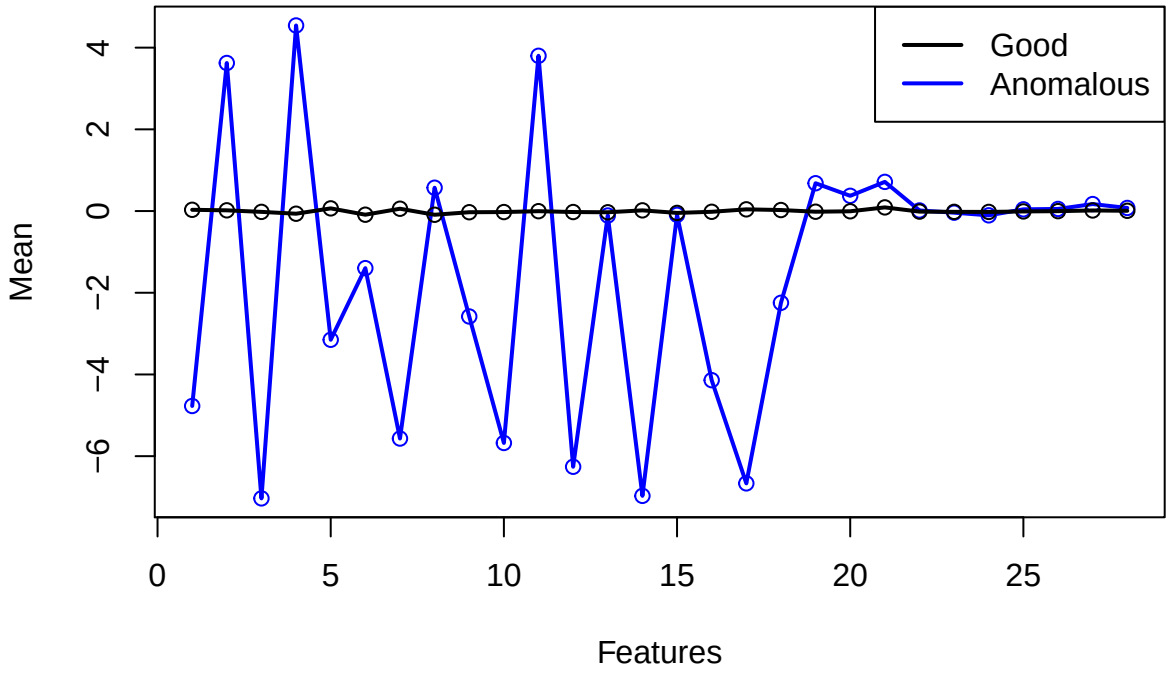
It is evident that the data is highly skewed with less than 0.2% of the transactions being fraud and the rest being legitimate. To do some exploratory analysis let us separate the normal (negative class) and anomalous records (positive class) and find out how the features differ between the two datasets.

```
rownames(X) <- 1:nrow(X)
Xgood <- X[X$Class == 0,]
Xanom <- X[X$Class == 1,]
```

3. Exploratory Analysis

As an initial let's find the mean values of all the features for both positive and negative examples and see how they vary.

```
mugood <- apply(Xgood[sample(rownames(Xgood), size = as.integer(skew *nrow(X)), replace = F), -c(1, 30,
muanom <- apply(Xanom[, -c(1, 30, 31)], 2, mean)
plot(muanom, col = "blue", xlab = "Features", ylab = "Mean")
lines(muanom, col = "blue", lwd = 2)
points(mugood, col = "black")
lines(mugood, col = "black", lwd = 2)
legend("topright", legend = c("Good", "Anomalous"), lty = c(1,1), col = c("black", "blue"), lwd = c(2,2
```

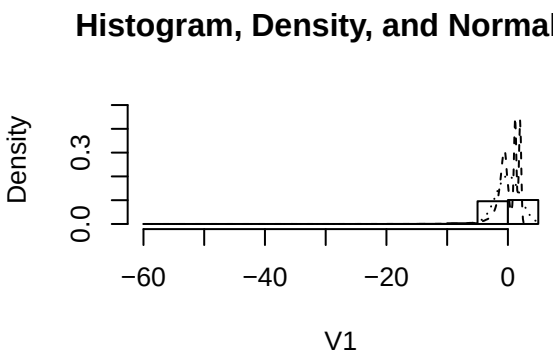


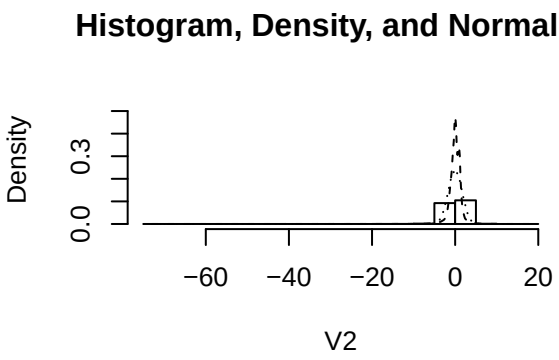
It is obvious that the mean value of some of the features for anomalous examples fall out of range. However, some features especially the later ones (19 through 28) does not vary as much. Hence, for our model it might be safe for us to just ignore these features and focus on those that contribute significantly for a transaction to be identified as fraud. We can also plot the variance of our features in a similar way, however for now I am going to assume that the variance is in its decreasing order from the first feature to the last as these are just the first few orthonormal vectors resulting from PCA of the original data.

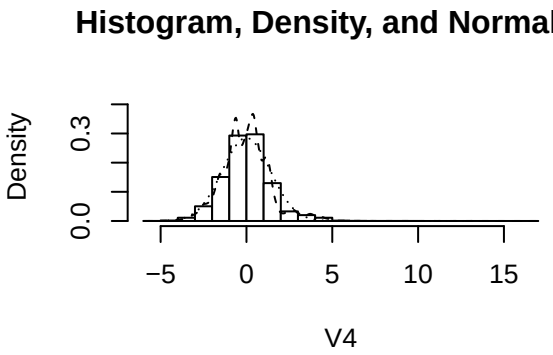
4. Feature Engineering

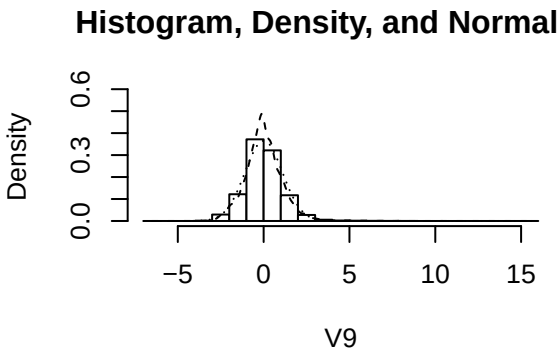
Incidentally, let's ignore the features that aren't of much interest and simplify the dataset for further analysis and training. Also plotting the histogram and density curves to check how the features are distributed, formally they should be close to Gaussian distribution.

```
# drop features that are trivial
todrop <- paste("V", c(8, 13, 15, 19:28), sep = "")
Xgood[, todrop] <- NULL
Xanom[, todrop] <- NULL
# transform the features into more gaussian like
#Xgood[,-c(1,17,18)] <- log(Xgood[,-c(1,17,18)] + 200)
#Xanom[,-c(1,17,18)] <- log(Xanom[,-c(1,17,18)] + 200)
# plot the histogram of features
multi.hist(Xgood[,c(2,3,5,9)], density = T)
```









Next, split the data into training, cross validation and test sets. We will esimate the mean and covariance using the training set, find the best epsilon using the cross validation set and evaluate model performance using the test set. We are going to put 60% of the normal examples into training, 20% into cross validation and the remaining 20% into test set with no anomalous examples in the training set and 50% in each of cross validation and test sets. The reason for this is that we want to estimate the gaussian from only the good transactions and try to find a threshold boundary (epsilon) that separates the anomalous examples using the cross validation set.

```
# split the data 60/20/20 into train/cv/test set with 0/50/50 anomalous examples
g <- rownames(Xgood); a <- rownames(Xanom);
lg <- length(g); la <- length(a);
# training set
Xtrain <- data.matrix(Xgood[g[1:(lg*0.6)],-c(1,17)])
# cross validation set
Xcv <- data.matrix(rbind(Xgood[g[(lg*0.6+1):(lg*0.8)],-c(1,17)], Xanom[a[1:(la*0.5)],-c(1,17)]))
# test set
Xtest <- data.matrix(rbind(Xgood[g[(lg*0.8+1):lg],-c(1,17)], Xanom[a[(la*0.5+1):la],-c(1,17)]))
# shuffle the data
Xcv <- Xcv[sample(nrow(Xcv), nrow(Xcv), replace = F),]
Xtest <- Xtest[sample(nrow(Xtest), nrow(Xtest), replace = F),]
# clear variables that are not required
rm(list = c("a", "g", "la", "lg"))
```

5. Model Formulation

Estimating the mean and covariance from the training set.

```
class = 16                              # column position of class variable
mu <- apply(Xtrain[,-class], 2, mean)      # mean of all features
```

```
sigma <- cov(Xtrain[,-class])                # covariance matrix of all features
#sigma <- apply(Xtrain[,-16], 2, var)
#sigma <- diag(sigma)                       # assuming all the features are uncorrelated
```

Calculating the pdf of all examples in the cross validation set using the mean and covariance from the training set. The function "selectThreshold" takes the ground truth positive and negative examples and the calculated probabilities to find the optimum threshold that best separates the classes. The function iterates through many values in "pcv" calculating the F1 score from precision and recall for each value thus finally selecting an optimum pcv that has the highest F1 score.

```
# find the pdf on cross validation set
pcv <- dmvnorm(Xcv[,-class], mu, sigma)
min(pcv)
```

```
## [1] 0
```

```
range(pcv[pcv>0])
```

```
## [1] 1.531604e-322  1.486060e-07
```

It turns out that the range of estimated probabilities are very wide and could be difficult for us to find the optimum threshold value with in a resonable number of iterations. To solve this let's apply a log function and narrow the range. Also, replacing zeros in the original probabilities with the next higher value to avoid numerical instability.

```
pcv[pcv==0] <- min(pcv[pcv>0])
pcv <- log(pcv)
# find the best threshold for maximum F1 score
epsilon <- selectThreshold(Xcv[,class], pcv)
sprintf('The threshold value is found to be epsilon = %e', epsilon)
```

```
## [1] "The threshold value is found to be epsilon = -2.195268e+02"
```

```
pred_cv <- as.numeric(pcv < epsilon)
confusionMatrix(table(pred_cv, Xcv[,class]),
                positive = levels(factor(Xcv[,class]))[2],
                mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##
## pred_cv     0     1
##       0 56783    49
##       1    80   197
##
##                Accuracy : 0.9977
##                  95% CI : (0.9973, 0.9981)
##     No Information Rate : 0.9957
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7522
##  Mcnemar's Test P-Value : 0.008258
##
##               Precision : 0.711191
##                  Recall : 0.800813
##                      F1 : 0.753346
##              Prevalence : 0.004308
```

```
##          Detection Rate : 0.003450
##    Detection Prevalence : 0.004850
##       Balanced Accuracy : 0.899703
##
##        'Positive' Class : 1
##
```

6. Model Evaluation

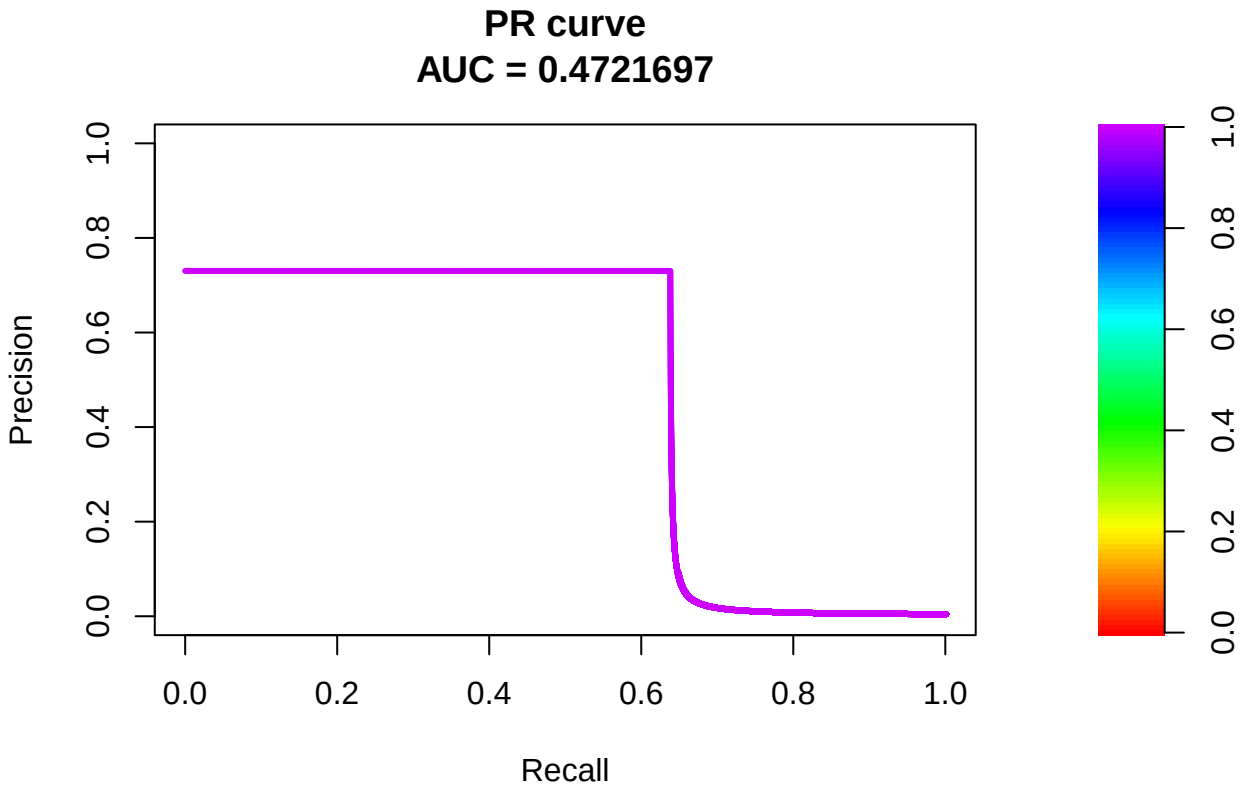
Let's test the model performance using the test set by calculating the pdfs followed by the log function the same way as we did for cross validation set. Consequently, predicting the positive class whenever the estimated probability is less than the threshold value and negative otherwise.

```
# evaluate the model on test set
ptest <- dmvnorm(Xtest[,-class], mu, sigma)
ptest[ptest==0] <- min(ptest[ptest>0])
ptest <- log(ptest)
pred_test <- as.numeric(ptest < epsilon)
confusionMatrix(table(pred_test, Xtest[,class]),
                positive = levels(factor(Xtest[,class]))[2],
                mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##
## pred_test     0     1
##         0 56805    89
##         1    58   157
##
##                Accuracy : 0.9974
##                  95% CI : (0.997, 0.9978)
##     No Information Rate : 0.9957
##     P-Value [Acc > NIR] : 5.557e-12
##
##                   Kappa : 0.6798
##  Mcnemar's Test P-Value : 0.01335
##
##               Precision : 0.730233
##                  Recall : 0.638211
##                      F1 : 0.681128
##              Prevalence : 0.004308
##          Detection Rate : 0.002749
##    Detection Prevalence : 0.003765
##       Balanced Accuracy : 0.818596
##
##        'Positive' Class : 1
##
```

Plotting the PR curve

```
# plot precision-recall curve
fg <- pred_test[Xtest[,class]==1]
bg <- pred_test[Xtest[,class]==0]
pr <- pr.curve(scores.class0 = fg, scores.class1 = bg, curve = T)
plot(pr)
```



As we can see the cross validation set got a precision of 71% and recall of about 80%, while the test set has the precision and recall at about 73% and 64% respectively.

7. Model Analysis

Let's now find out why the model is classifying lot of false positives and false negatives. Since we classified an example as positive class when the calculated probability is less than epsilon, clearly there must be lot of negative examples that has probability less than epsilon (false positives) and positive examples that has probability greater than epsilon (false negatives). Let's focus on the false positive cases and see why the calculated probabilities of these legitimate transactions are below the threshold value.

```
min(ptest)
```

```
## [1] -732.588
```

```
sum(names(ptest[ptest==min(ptest)]) %in% rownames(X[X$Class==0,]))
```

```
## [1] 10
```

Surprisingly, there are 10 records with the least probability in the test set that are not anomalous but are classified as such given the epsilon value of -219. This means that irrespective of the selected threshold value, the model could never classify these examples as negative. By definition of our algorithm, by calculating the independence assumption of the features, it tend to be the case that the model estimates a low probability when the features take on some extreme value for an anomalous example and a higher probability when they do not, typically for a normal example.

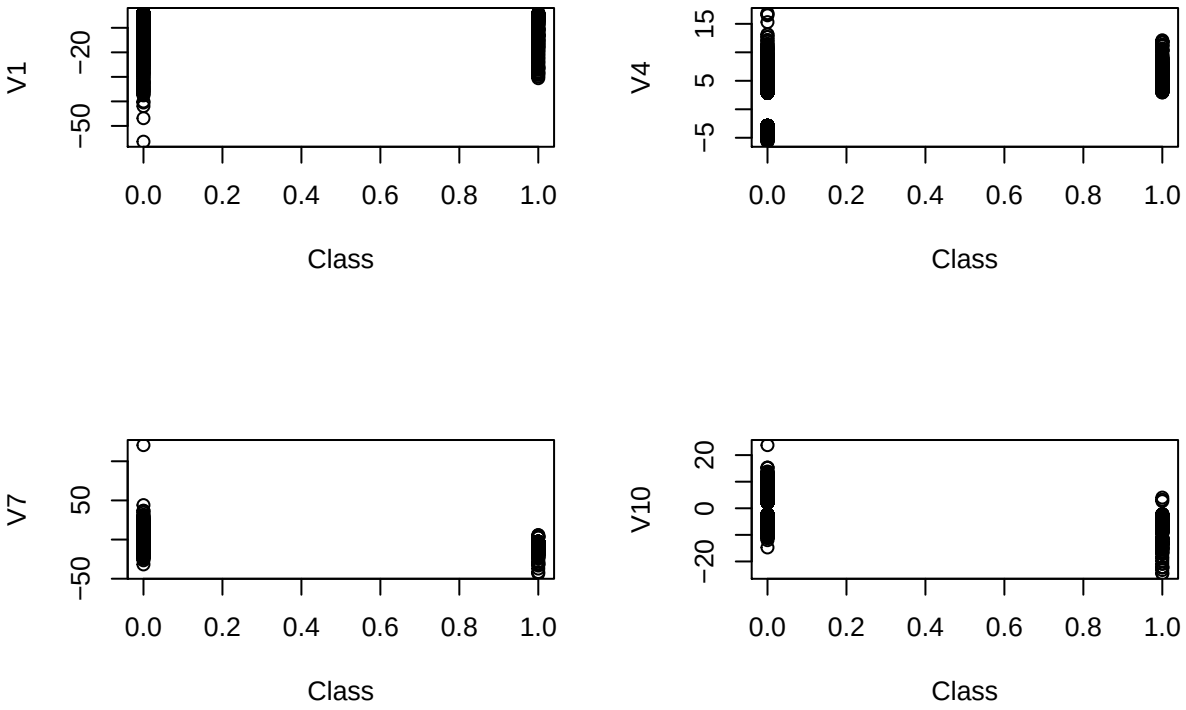
In the next section, we will see how the features vary by the Class variable to get a better understanding of how they are distributed.

Since we are interested in the extreme values, our goal is to find records that have taken feature values beyond at least 2 standard deviations from the mean. More formally, we are going to find the confidence intervals

(98% would give us little more than 2 standard deviations) and feature values that fall outside of the interval. Let's also plot these extreme values for few features and see how they span for each class.

```
X[, todrop] <- NULL
# find confidence intervals of all the features
ci <- apply(X[,-c(1, 17, 18)], 2, function(a) { mean(a) + c(-1,1)*qnorm(0.98)*sd(a) })

# find examples that fall outside the ci
extremes <- list()
tmp <- names(X)[grepl("V", names(X))]
for (i in tmp) {
  extremes[[i]] <- rownames(X[X[, i] < ci[1, i] | X[, i] > ci[2, i],])
}
par(mfrow=c(2,2))
# plot features
with(X[extremes$V1,], plot(Class, V1))
with(X[extremes$V4,], plot(Class, V4))
with(X[extremes$V7,], plot(Class, V7))
with(X[extremes$V10,], plot(Class, V10))
```



Contrary to what we ideally want, it turns out that there are lot of negative examples whose feature values are beyond 2 standard deviations from the mean which explains the cause for predicting lot of false positives. In addition, the plot also shows that the range of feature values for positive examples are either close or with in the range of negative examples rendering the features inadequate to classify the positive and negative examples better. One method to solve this problem is to choose features or even engineer new features from the original ones that tend to take unusually large or small values in the event of an anomaly.