Credit Card Fraud Detection

Leroy S. Buliro

1/29/2020

Executive Summary

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. Given the data, the goal is to identify fraudulent credit card transactions.

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

Source : https://www.kaggle.com/mlg-ulb/creditcardfraud

Frame the problem

To identify a fraudulant credit card transaction from this dataset. Since target variable are 0(non-fraud) and 1(fraud), this is a binary classification problem. Supervised machine learning technique such as Logistic regresssion and Support vector machine could be good fit for this problem

Metric for evaluation

AUC : a classic evaluation measurement for classifiers that plot Sensitivity against Specificity

AUCPR : a evaluation measurement for classifiers for an imbalanced data that plot Precision against Recall

Exploratory Data Analysis

Dimension

```
## [1] 284807     31
```

Data Dictionary

Time : Number of seconds elapsed between this transaction and the first transaction in the dataset
V1 - V28 : result of a PCA Dimensionality reduction to protect user identities and sensitive features(v1-v28)
Amount : Transaction amount Class : 1 for fraudulent transactions, 0 otherwise

Imbalanced target The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. This could lead to misunderstanding when interpret the result of accuracy of the classification

```
## # A tibble: 2 x 2
##   Class  Count
##   <int>  <int>
## 1     0 284315
## 2     1    492
```

Fraud Total Amount

```
## [1] 39767309
```

Fraud amount propotion

```
## [1] 0.001473461
```

Missing Values There are no missing value in the data set

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

Header of dataset

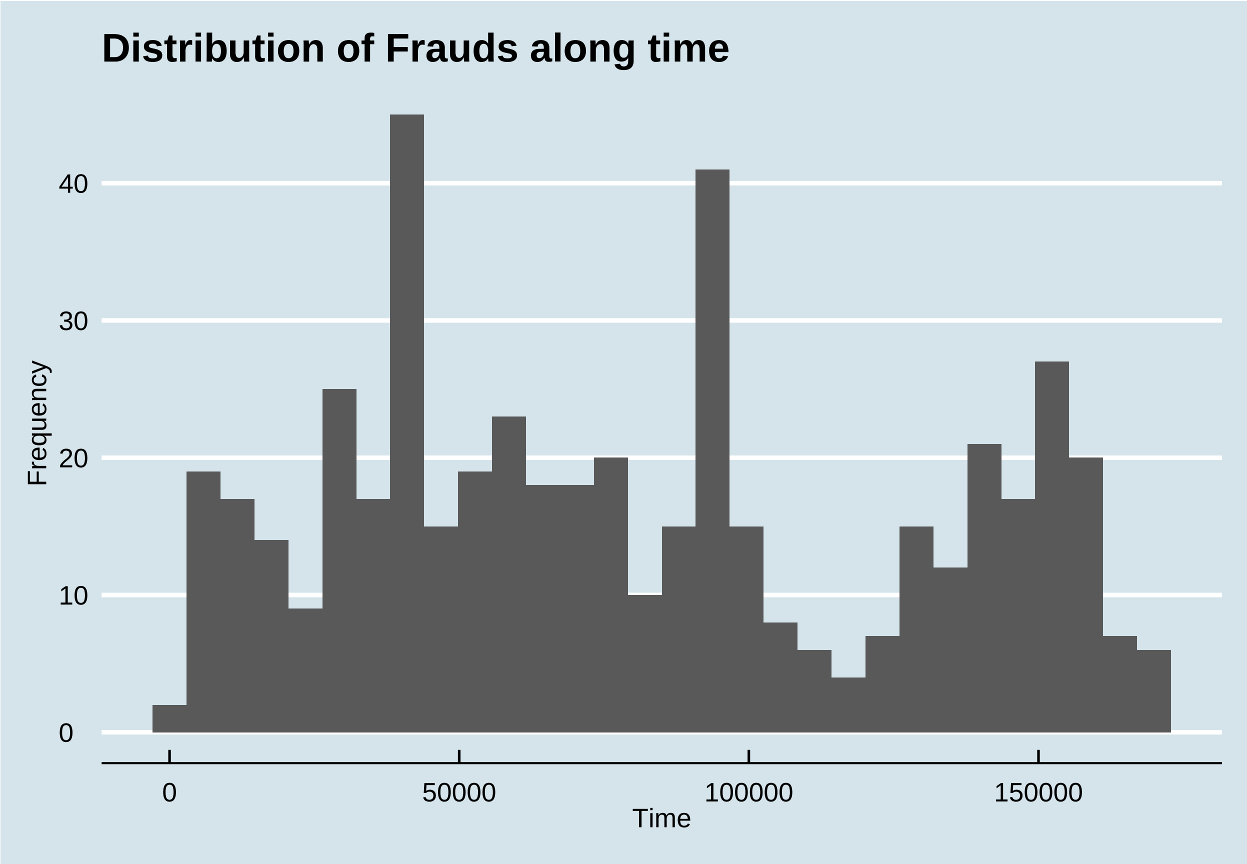
```
##   Time         V1          V2        V3         V4          V5          V6
## 1    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
## 5    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
## 6    2 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755
##            V7          V8         V9         V10        V11         V12
## 1  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
## 2 -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
## 3  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
## 4  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
## 6  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620  0.35989384
##          V13        V14        V15        V16         V17         V18
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124  0.02579058
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466 -0.18336127
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938 -0.12135931
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279  1.96577500
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324 -0.03819479
## 6 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282  0.06865315
##           V19         V20          V21          V22         V23         V24
## 1  0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391  0.06692807
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802 -0.33984648
## 3 -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226 -0.68928096
## 4 -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052 -1.17557533
## 5  0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808  0.14126698
## 6 -0.03319379  0.08496767 -0.208253515 -0.559824796 -0.02639767 -0.37142658
##          V25        V26          V27         V28 Amount Class
## 1  0.1285394 -0.1891148  0.133558377 -0.02105305 149.62     0
## 2  0.1671704  0.1258945 -0.008983099  0.01472417   2.69     0
## 3 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66     0
```

```
## 4  0.6473760 -0.2219288  0.062722849  0.06145763 123.50     0
## 5 -0.2060096  0.5022922  0.219422230  0.21515315  69.99     0
## 6 -0.2327938  0.1059148  0.253844225  0.08108026   3.67     0
```

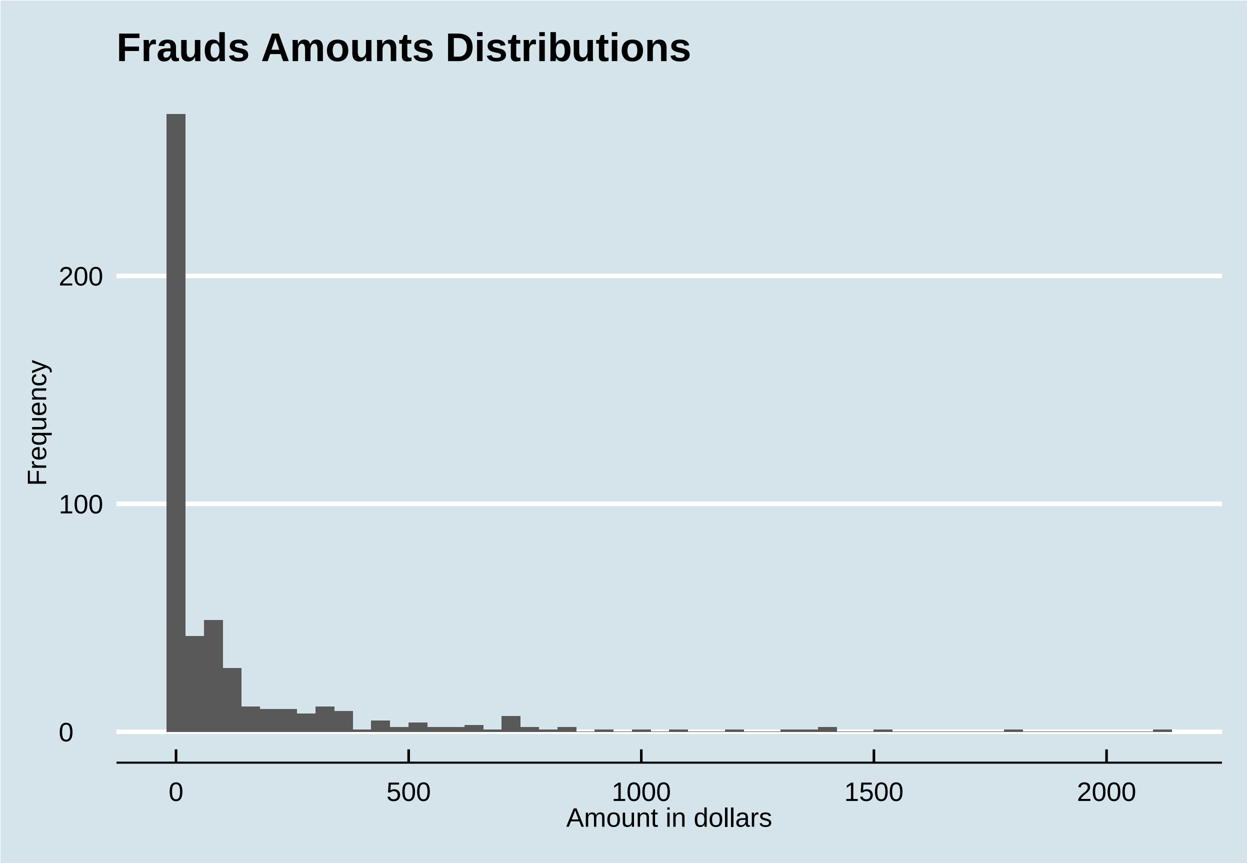
Distribution of Data

Distribution of Frauds along time

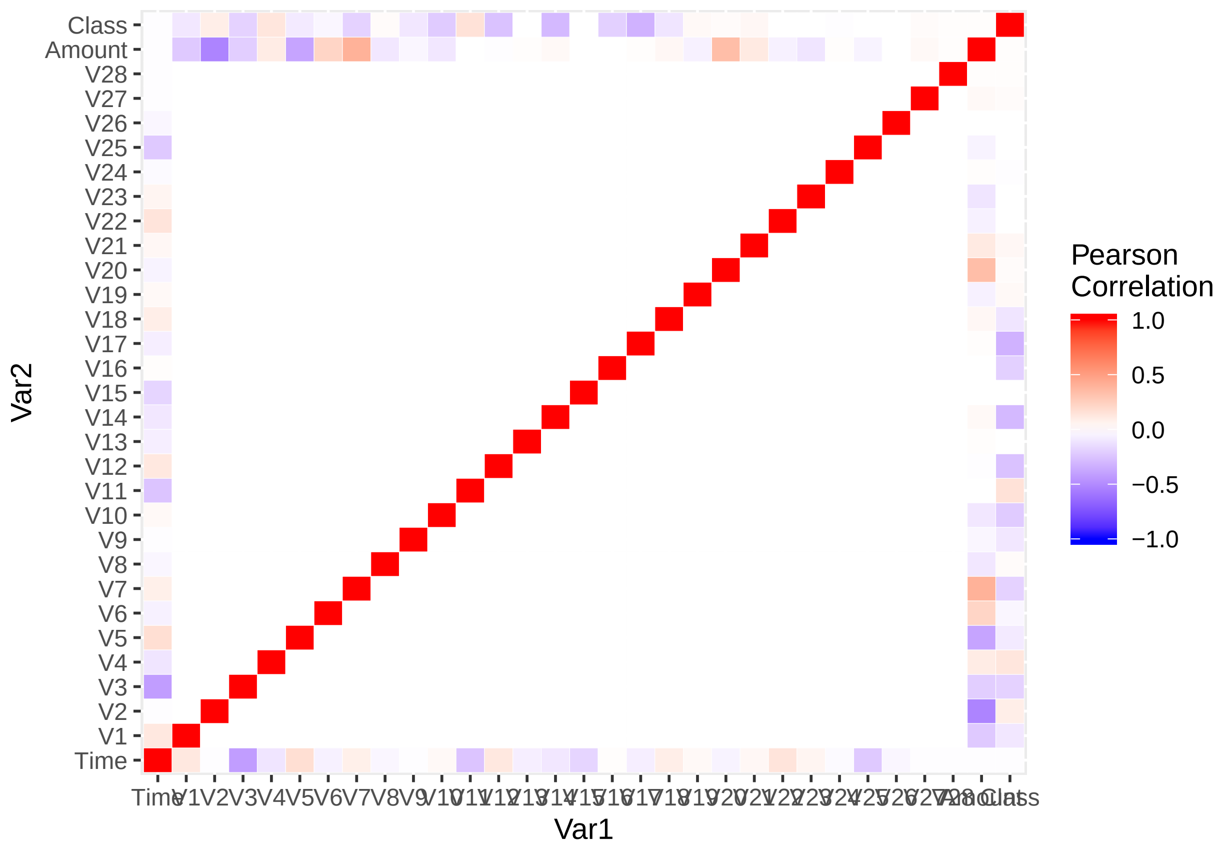
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Distribution of Frauds Amount in dollars



Study Correlations



Data Preparation

Convert target variable class into factor (categorical)

```
creditcard$Class <- as.factor(creditcard$Class)
```

Train- test split Split data into train, test and cross validation set

Model Building

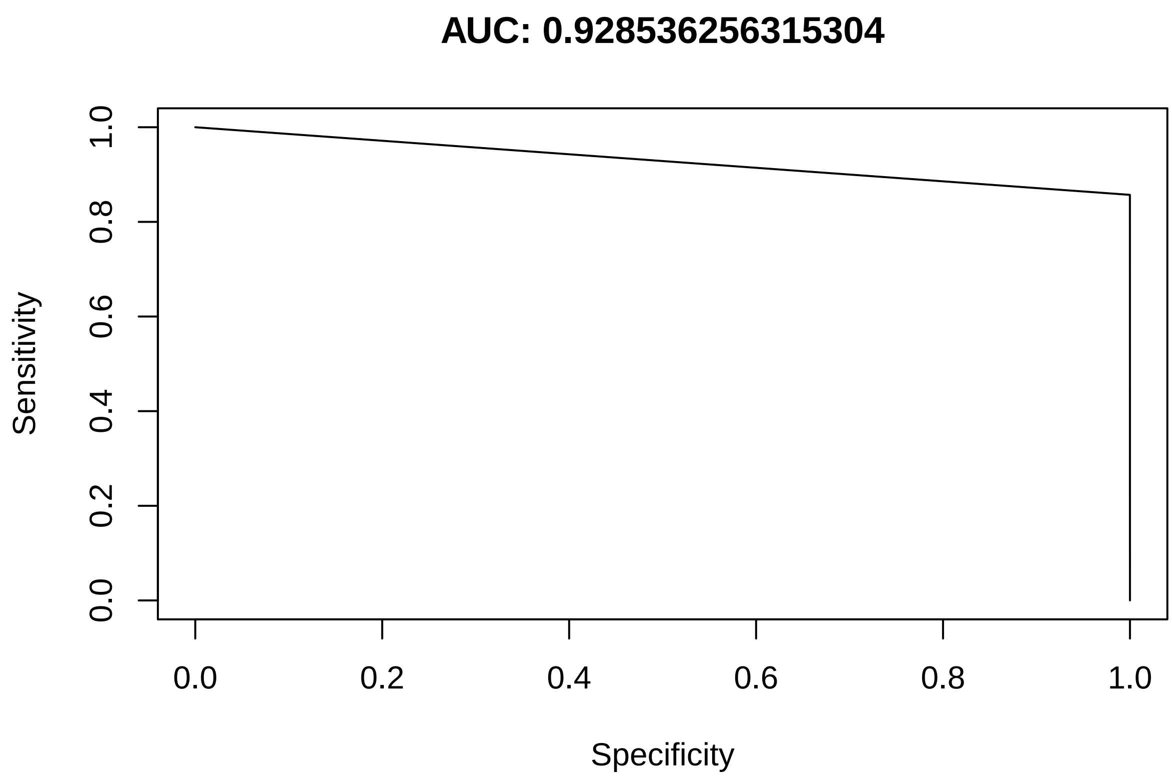
Base model - Random forrest Classifier

Random forrest perform quite well on data set with many features. Let's use it as a baselie model. The result are an AUC of 0.88 and AUCPR of 0.75.

```
rf_model <- randomForest(Class ~ ., data = train, ntree = 500)
```

```
# Get the feature importance

feature_imp_rf <- data.frame(importance(rf_model))

# Make predictions based on this model
```

```
predictions <- predict(rf_model, newdata=test)

# Compute the AUC and AUPCR

pred <- prediction(
  as.numeric(as.character(predictions)),
  as.numeric(as.character(test$Class))
)

auc_val_rf <- performance(pred, "auc")

auc_plot_rf <- performance(pred, 'sens', 'spec')

aucpr_plot_rf <- performance(pred, "prec", "rec", curve = T,  dg.compute = T)

aucpr_val_rf <- pr.curve(scores.class0 = predictions[test$Class == 1], scores.class1 = predictions[test$

# make the relative plot

plot(auc_plot_rf, main=paste("AUC:", auc_val_rf@y.values[[1]]))
```




```
# Adding the respective metrics to the results dataset


results <- data.frame(
```

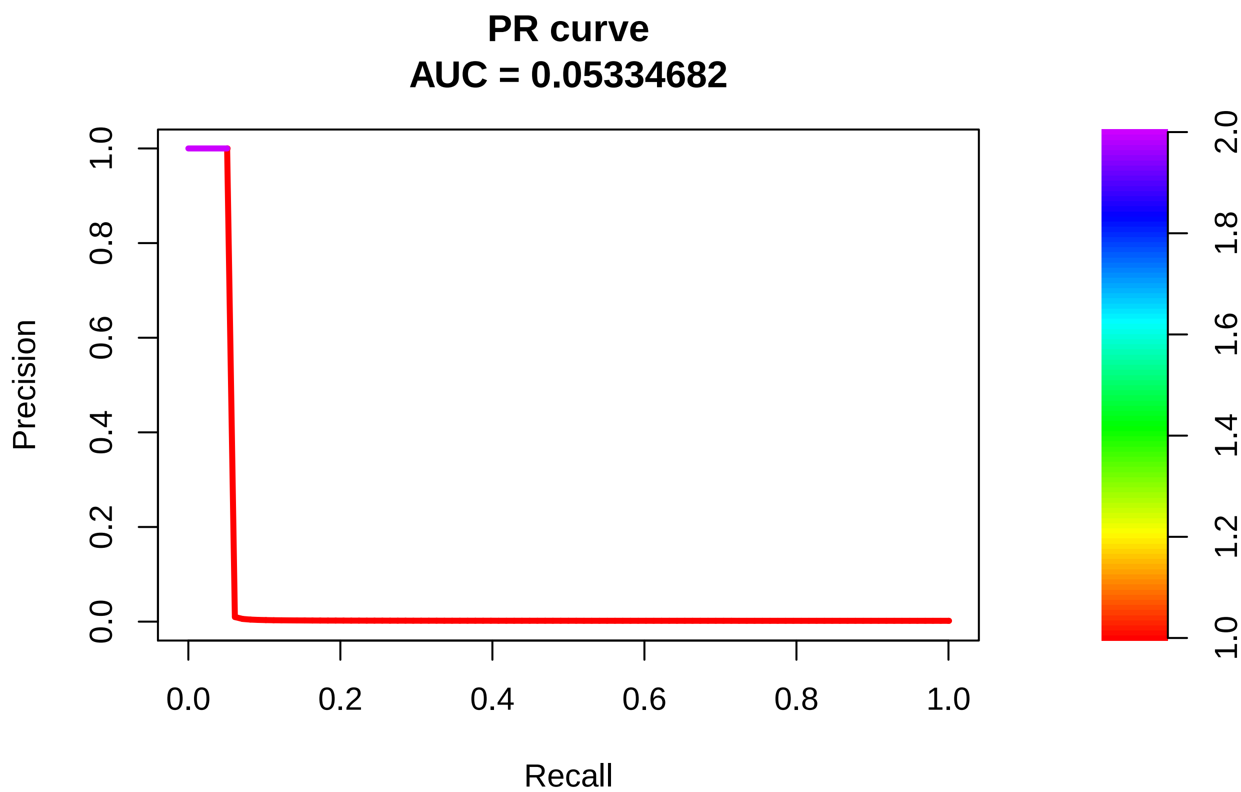
```r
  Model = "Random Forest",
  AUC = auc_val_rf@y.values[[1]],
  AUCPR = aucpr_val_rf$auc.integral)

results
```

```
##           Model       AUC     AUCPR
## 1 Random Forest 0.9285363 0.8195824
```

KNN - K-Nearest Neighbors

The result are an AUC of 0.55 and AUCPR of 0.1. Which is quite obvious since random forrest should perform better.

```r
# Build a KNN Model with Class as Target and all other
# variables as predictors. k is set to 5

knn_model <- knn(train[,-30], test[,-30], train$Class, k=5, prob = TRUE)

# Compute the AUC and AUCPR for the KNN Model

pred <- prediction(
  as.numeric(as.character(knn_model)),
  as.numeric(as.character(test$Class))
)

auc_val_knn <- performance(pred, "auc")

auc_plot_knn <- performance(pred, 'sens', 'spec')
aucpr_plot_knn <- performance(pred, "prec", "rec")

aucpr_val_knn <- pr.curve(
  scores.class0 = knn_model[test$Class == 1],
  scores.class1 = knn_model[test$Class == 0],
  curve = T,
  dg.compute = T
)

# Make the relative plot

plot(aucpr_val_knn)
```




```r
# Adding the respective metrics to the results dataset

results <- results %>% add_row(
  Model = "K-Nearest Neighbors k=5",
  AUC = auc_val_knn@y.values[[1]],
  AUCPR = aucpr_val_knn$auc.integral
)
results
```

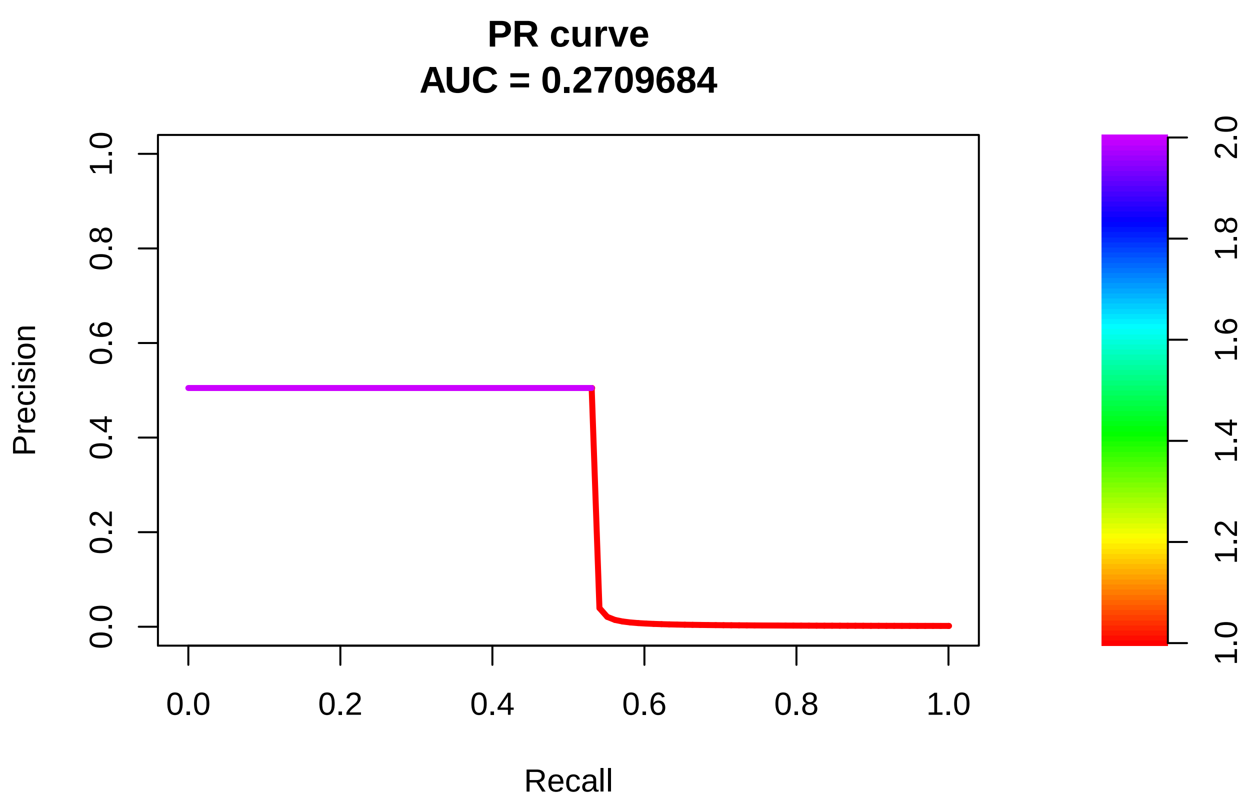
```
##                     Model       AUC      AUCPR
## 1           Random Forest 0.9285363 0.81958237
## 2 K-Nearest Neighbors k=5 0.5255102 0.05334682
```

SVM - Support Vector Machine

The result are an AUC of 0.8 and AUCPR of 0.3. See that for imbalanced data SVM perform quite terrible.

```r
# Set seed 1234 for reproducibility
set.seed(1234)

# Build a SVM Model with Class as Target and all other
# variables as predictors. The kernel is set to sigmoid

svm_model <- svm(Class ~ ., data = train, kernel='sigmoid')
```

```
# Make predictions based on this model

predictions <- predict(svm_model, newdata=test)

# Compute AUC and AUCPR

pred <- prediction(
  as.numeric(as.character(predictions)),                                  as.numeric(as.character(test$Cl
)

auc_val_svm <- performance(pred, "auc")

auc_plot_svm <- performance(pred, 'sens', 'spec')
aucpr_plot_svm <- performance(pred, "prec", "rec")

aucpr_val_svm <- pr.curve(
  scores.class0 = predictions[test$Class == 1],
  scores.class1 = predictions[test$Class == 0],
  curve = T,
  dg.compute = T
)

# Make the relative plot

plot(aucpr_val_svm)
```



```
# Adding the respective metrics to the results dataset

results <- results %>% add_row(
  Model = "SVM - Support Vector Machine",
  AUC = auc_val_svm@y.values[[1]],
  AUCPR = aucpr_val_svm$auc.integral)
results
```

```
##                          Model       AUC      AUCPR
## 1                Random Forest 0.9285363 0.81958237
## 2      K-Nearest Neighbors k=5 0.5255102 0.05334682
## 3 SVM - Support Vector Machine 0.7648577 0.27096841
```

Results

This is the summary results for all the models builted, trained and validated.

```
# Shows the results

results %>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
             position = "center",
             font_size = 10,
             full_width = FALSE)
```

Model	AUC	AUCPR
Random Forest	0.9285363	0.8195824
K-Nearest Neighbors k=5	0.5255102	0.0533468
SVM - Support Vector Machine	0.7648577	0.2709684

Conclusion

See that, ensemble model, random forrest performs the best among other models. It is clear why random forrest gain a lot of popularity among data science community.