Fraud Detection Project

Andy Zhao

2023-04-04

Overview

The objective of this project is to build some models to assess their qualities in detecting credit card transaction fraud. The dataset is downloaded from this websitehttps://www.kaggle.com/datasets/mlg-ulb/creditcardfraud (https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud).

The Time, Amount,and V1-28 columns are predictors. The Class is the response variable, 1 represents fraud, 0 represents legal, and there are 492 frauds out of 284,807 transactions. The frauds account for 0.172% of all transactions. The dataset's extremely unbalanced. Thus AUC score would be unreliable. So AUPRC would be measured and compared between those values.

A good Area Under Precision Recall Curve should be higher than 0.8. After the training and evaluating of 9 popular models, **lightGBM** model wins with the highest AUPRC value at **0.8245189**.

Exploratory Data Analysis

First, read the dataset, check its dimensions, structures, and confirm that features V1, V2, … V28 are the principal components obtained with PCA. Time isn't as relevant, so will be removed. Amount should be scaled. Correlation matrix plot and heatmap show that V17, V14 and V12 might be the most significant variables.

Data Exploration

```
#clear environment
rm(list=ls())

#make reproducible
set.seed(1)

#set working directories
setwd("/users/zhaolong/downloads")

#read the dataset
cc<- read.csv("creditcard.csv")
```

```
#data exploratory
dim(cc)
```

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

```
str(cc)
```

```
## 'data.frame':    284807 obs. of  31 variables:
##  $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
##  $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
##  $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
##  $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
##  $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
##  $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
##  $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
##  $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
##  $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
##  $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
##  $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
##  $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
##  $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
##  $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
##  $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
##  $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
##  $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
##  $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
##  $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
##  $ V19   : num  0.404 -0.146 -2.262 -1.233 0.803 ...
##  $ V20   : num  0.2514 -0.0691 0.525 -0.208 0.4085 ...
##  $ V21   : num  -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...
##  $ V22   : num  0.27784 -0.63867 0.77168 0.00527 0.79828 ...
##  $ V23   : num  -0.11 0.101 0.909 -0.19 -0.137 ...
##  $ V24   : num  0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...
##  $ V25   : num  0.129 0.167 -0.328 0.647 -0.206 ...
##  $ V26   : num  -0.189 0.126 -0.139 -0.222 0.502 ...
##  $ V27   : num  0.13356 -0.00898 -0.05535 0.06272 0.21942 ...
##  $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
##  $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
##  $ Class : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
#summary(cc)
head(cc,3)
```

	T...	V1	V2	V3	V4	V5	V6	V7
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	0	-1.359807	-0.07278117	2.5363467	1.3781552	-0.33832077	0.46238778	0.23959855
2	0	1.191857	0.26615071	0.1664801	0.4481541	0.06001765	-0.08236081	-0.07880298
3	1	-1.358354	-1.34016307	1.7732093	0.3797796	-0.50319813	1.80049938	0.79146096

3 rows | 1-10 of 32 columns

```
table(cc$Class)
```

```
##
##      0      1
## 284315    492
```

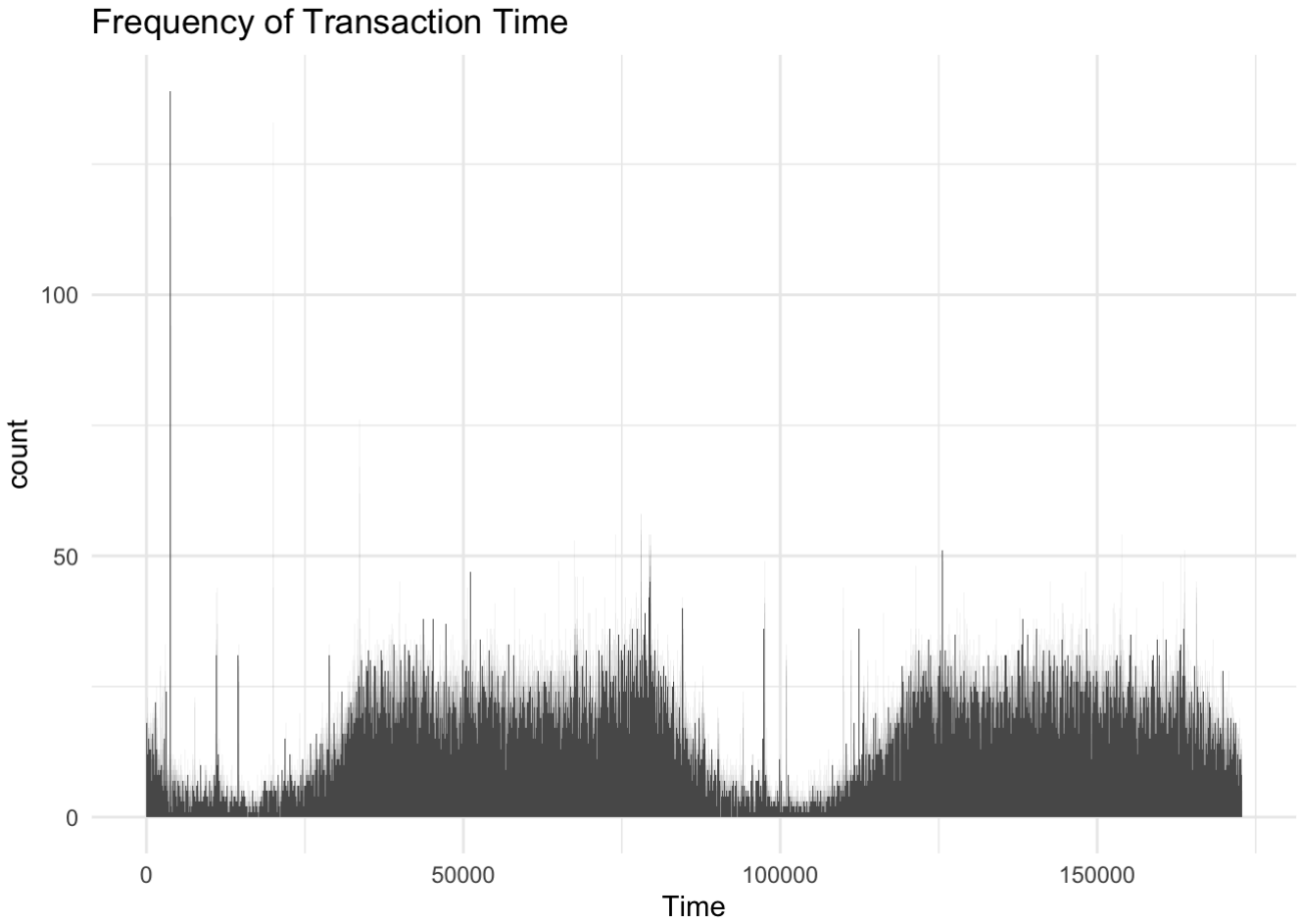
```
#find missing values
sapply(cc, function(x) sum(is.na(x)))
```

```
##   Time     V1     V2     V3     V4     V5     V6     V7     V8     V9    V10
##      0      0      0      0      0      0      0      0      0      0      0
##    V11    V12    V13    V14    V15    V16    V17    V18    V19    V20    V21
##      0      0      0      0      0      0      0      0      0      0      0
##    V22    V23    V24    V25    V26    V27    V28 Amount  Class
##      0      0      0      0      0      0      0      0      0
```

Data Visualization

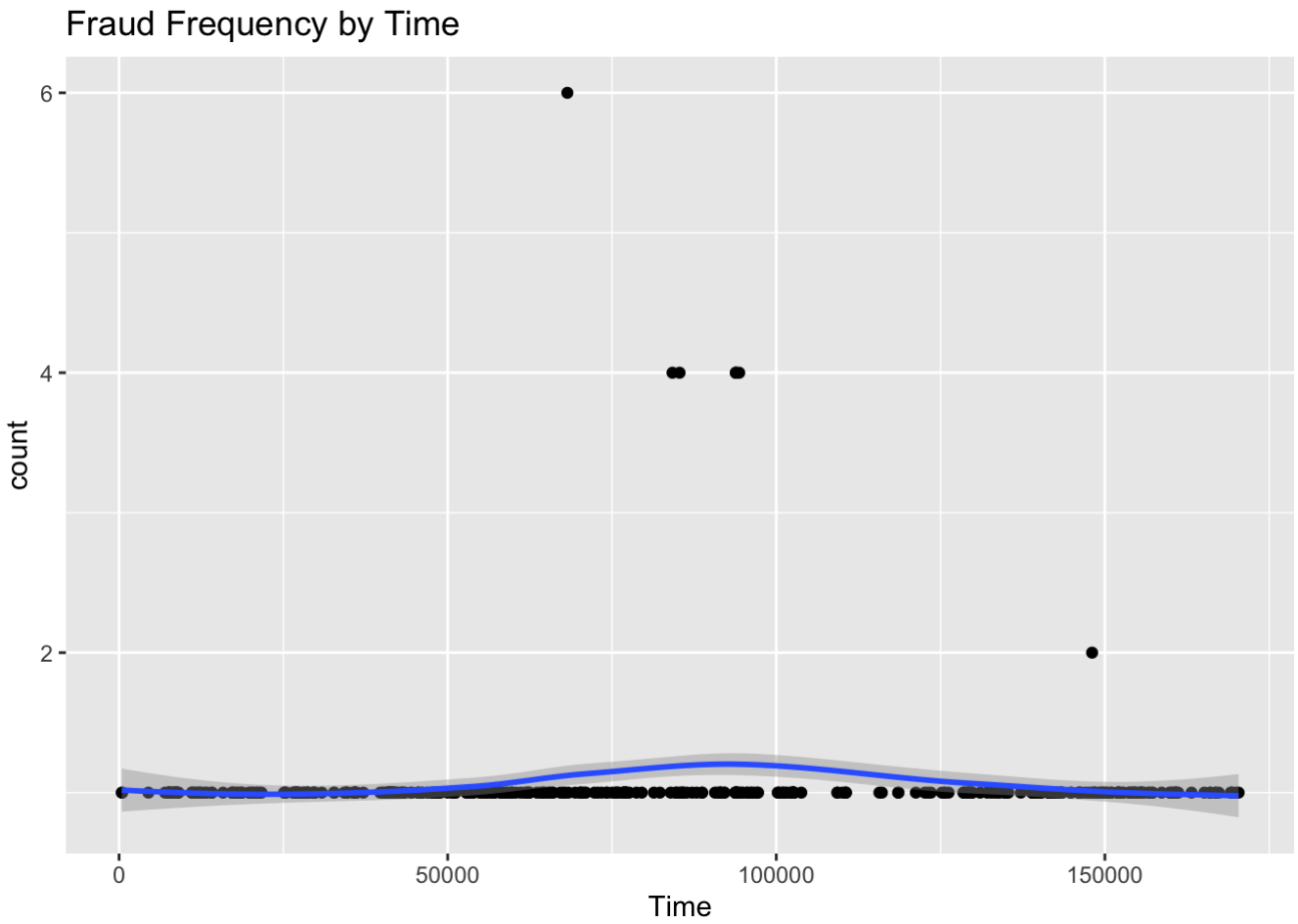
```
#Proportions between Legal and Frauds Transactions
cc %>%
  ggplot(aes(Class)) +
  geom_bar() +
  scale_x_discrete() +
  labs(title = "Proportions between Legal and Frauds Transactions")
```


```
#Frequency of Transaction Time
cc %>%
  ggplot(aes(Time)) +
  theme_minimal()  +
  geom_histogram(binwidth = 10)  +
  labs(title = "Frequency of Transaction Time")
```



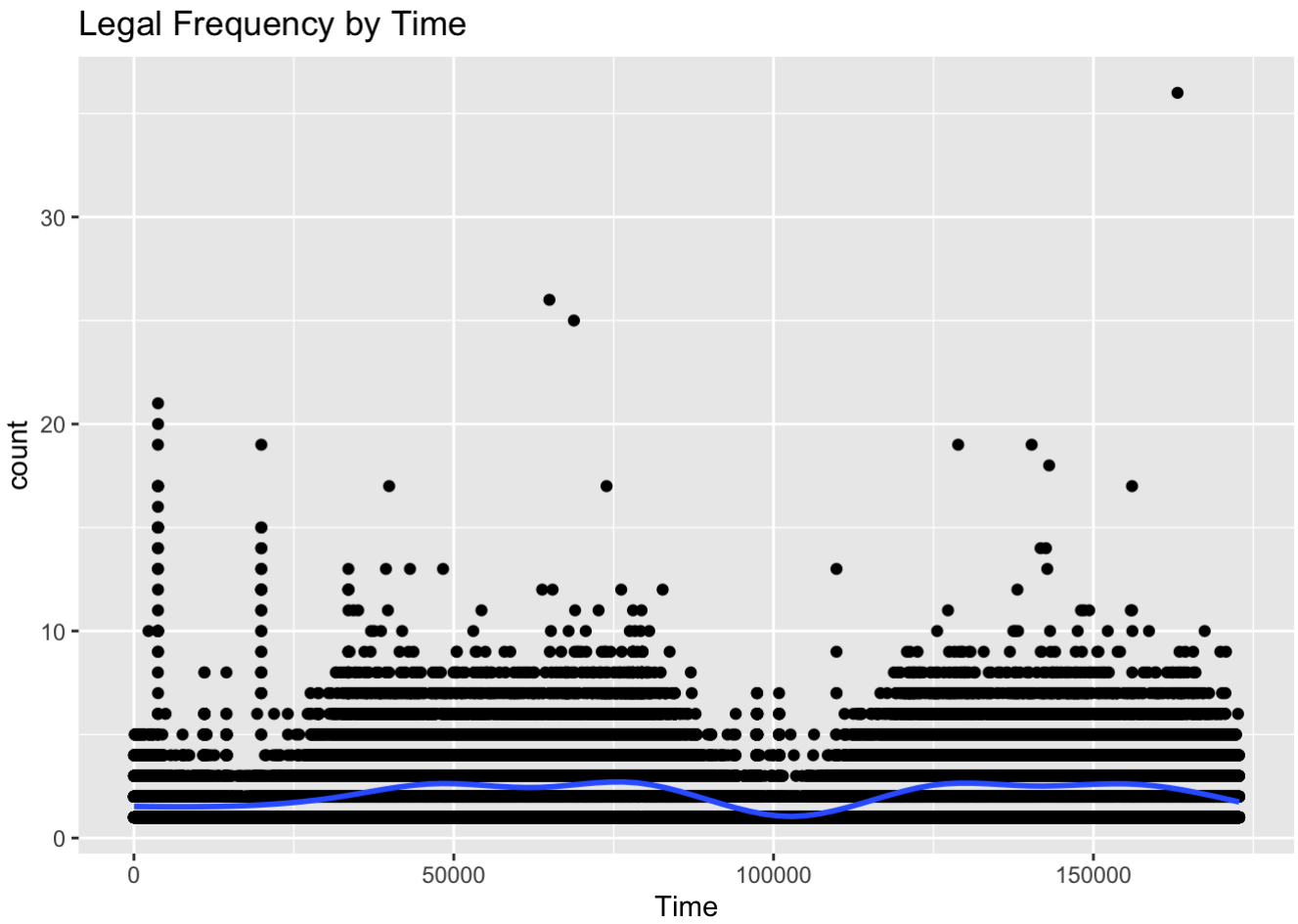

```
#Fraud Frequency by Time
cc[cc$Class==1,] %>% group_by(Time) %>%
  summarize(count=n()) %>%
  ggplot(aes(Time, count)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Fraud Frequency by Time")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

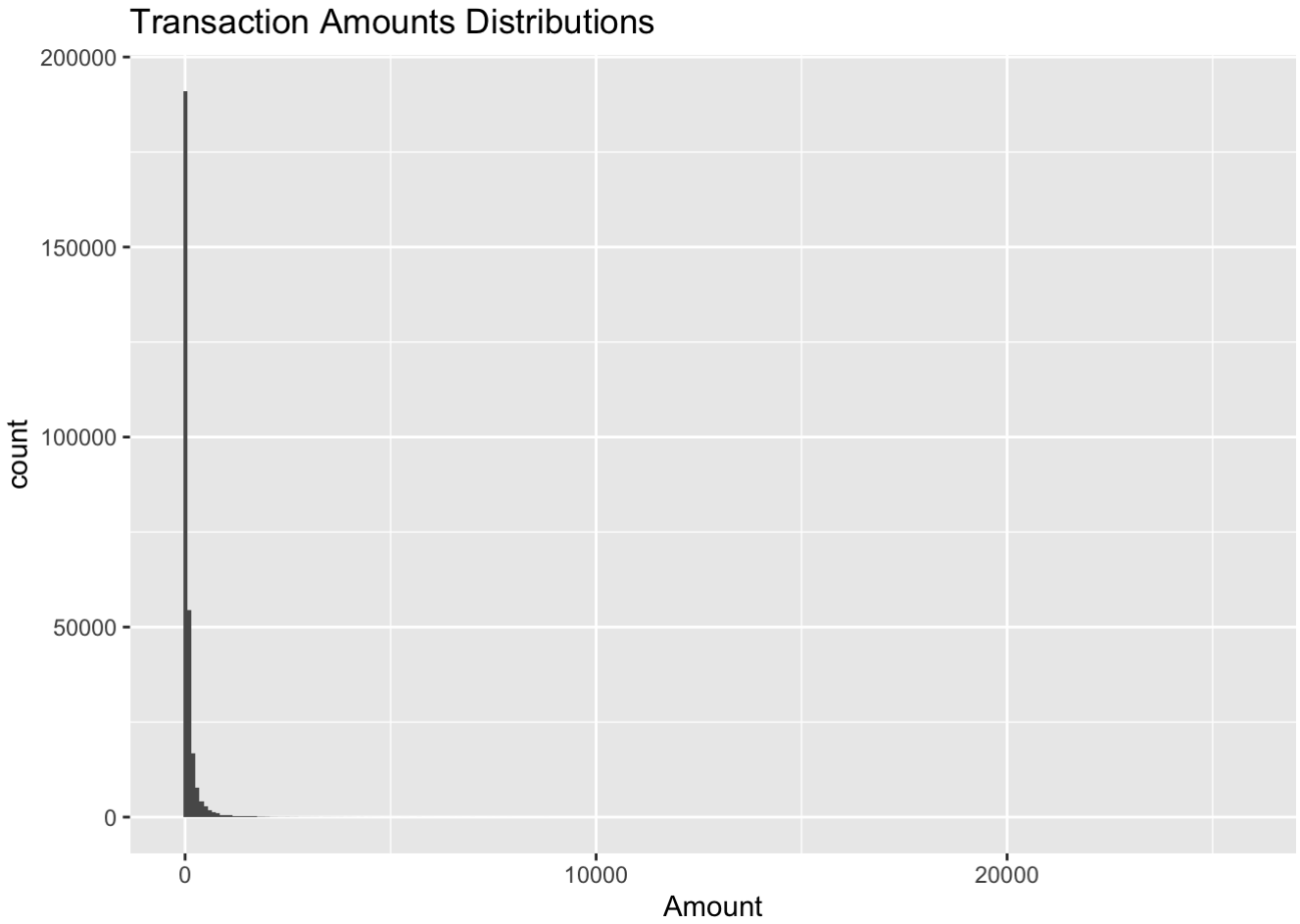



```
#Legal Frequency by Time
cc[cc$Class==0,] %>% group_by(Time) %>%
  summarize(count=n()) %>%
  ggplot(aes(Time, count)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Legal Frequency by Time")
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```




```
#Frauds Amounts Distributions
cc%>%
  ggplot(aes(Amount)) +
  geom_histogram(binwidth = 100) +
  labs(title = "Transaction Amounts Distributions")
```




```
#Top 10 Transaction Amounts
cc%>%
  group_by(Amount) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  head(n=10) %>% kable() %>% kable_styling(latex_options = "HOLD_position",bootstrap_
options = c("responsive"),
                                           position = "center",
                                           full_width = FALSE) %>% column_spec(2,colo
r = "white" , background ="green")
```

Amount	count
1.00	13688
1.98	6044
0.89	4872
9.99	4747
15.00	3280
0.76	2998
10.00	2950
1.29	2892

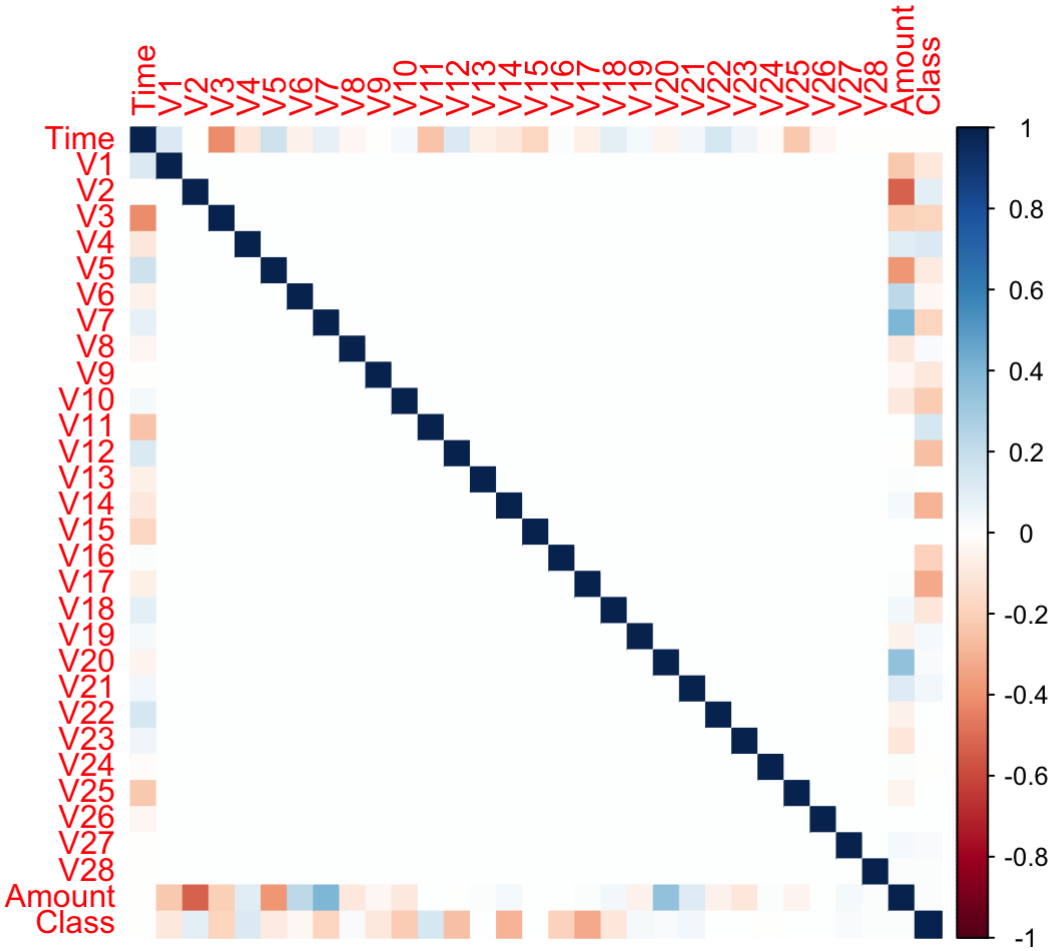
Amount	count
1.79	2623
0.99	2304

```
#Bottom 10 Transaction Amounts
cc%>%
  group_by(Amount) %>%
  summarise(count = n()) %>%
  arrange(count) %>%
  head(n=10) %>% kable() %>% kable_styling(latex_options = "HOLD_position",bootstrap_
options = c("responsive"),
                                     position = "center",
                                     full_width = FALSE) %>% column_spec(2,color =
"white" , background ="red")
```

Amount	count
0.18	1
0.29	1
0.34	1
21.88	1
28.47	1
32.28	1
32.58	1
33.18	1
33.64	1
33.84	1

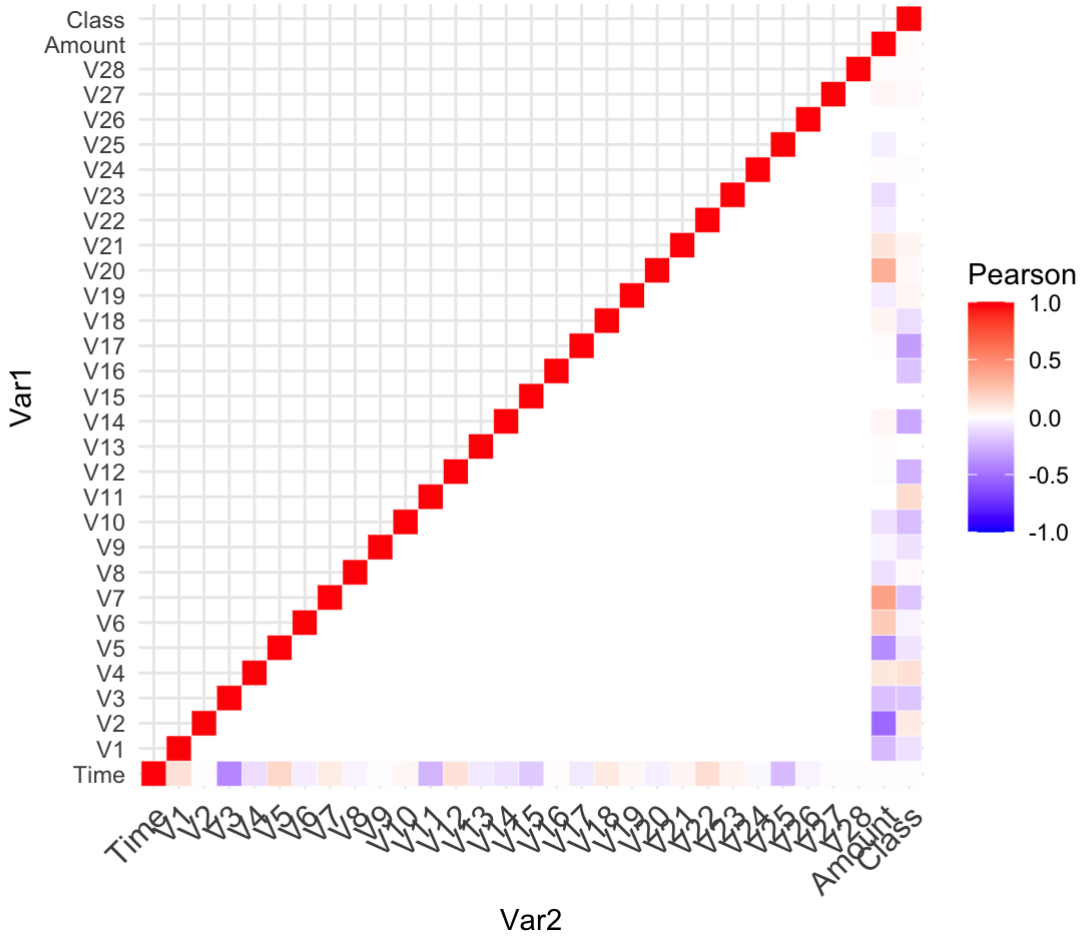
Correlation Matrix Plot

```
#scale Amount column
cc$Amount <- scale(cc$Amount, center = TRUE, scale = TRUE)

#prepare Class for correlation
cc$Class <- as.numeric(cc$Class)

#plot the correlation matrix
cormat<-round(cor(cc),2)
corrplot(cormat, method = "color")
```



Heatmap Plot

```
# Get upper triangle of the correlation matrix
get_upper_tri <- function(cormat){
  cormat[lower.tri(cormat)]<- NA
  return(cormat)
}
upper_tri <- get_upper_tri(cormat)

# Melt the correlation matrix
library(reshape2)
melted_cormat <- melt(upper_tri, na.rm = TRUE)

# Plot the heatmap
ggplot(data = melted_cormat, aes(Var2, Var1, fill = value))+
  geom_tile(color = "white")+
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1,1), space = "Lab",
                       name="Pearson") +
  theme_minimal()+
  theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                   size = 12, hjust = 1))+
  coord_fixed()
```




```
#data preparation and splitting
cc1<-cc[,-1]
index<-sample(1:nrow(cc1),as.integer(0.75*nrow(cc1)))
train <- cc1[index,]
test <- cc1[-index,]
```

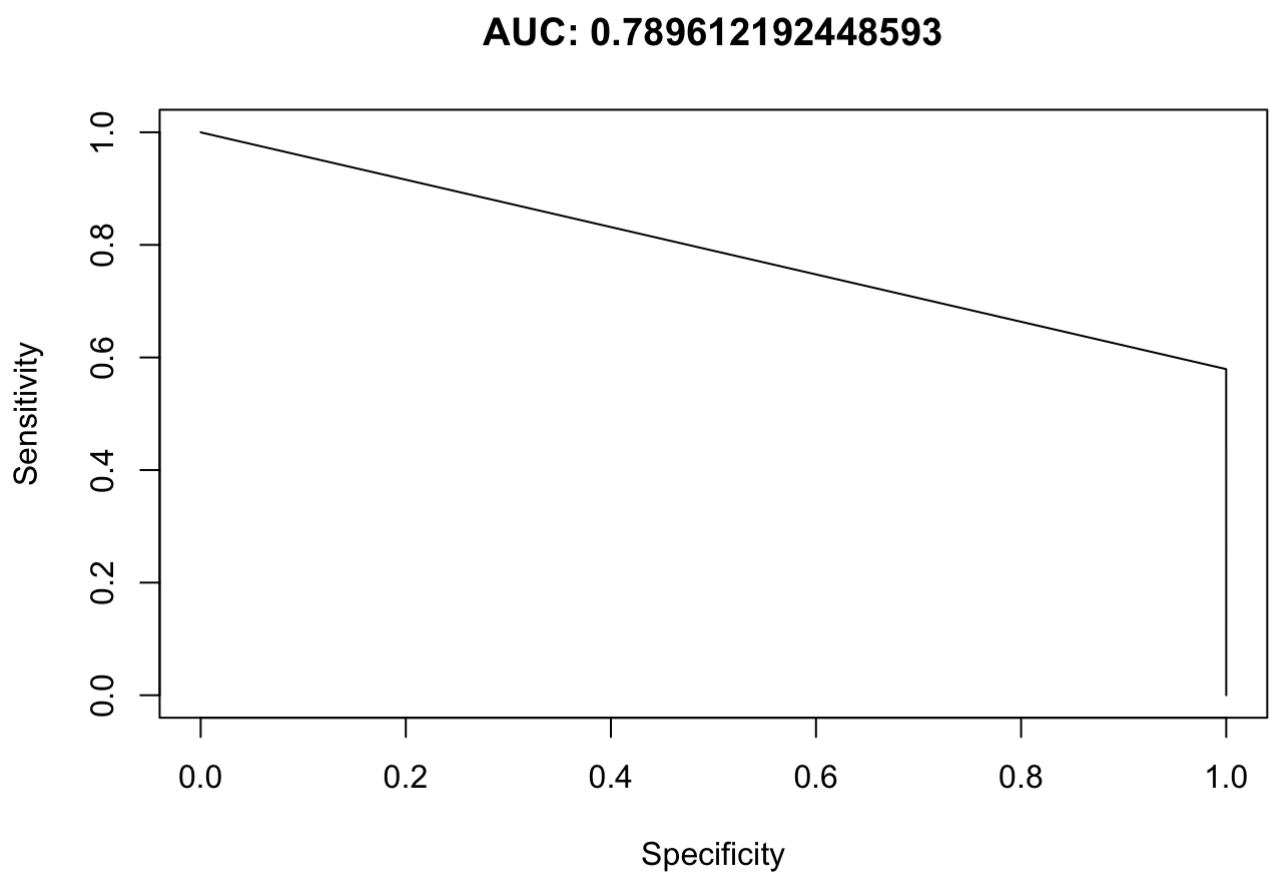
Methods - Modeling Analysis and Comparison

After use the train dataset to train the 9 models, apply the models on test dataset to predict respectively. Then measure AUC and AUPRC values, and make plots to compare their accuracies.

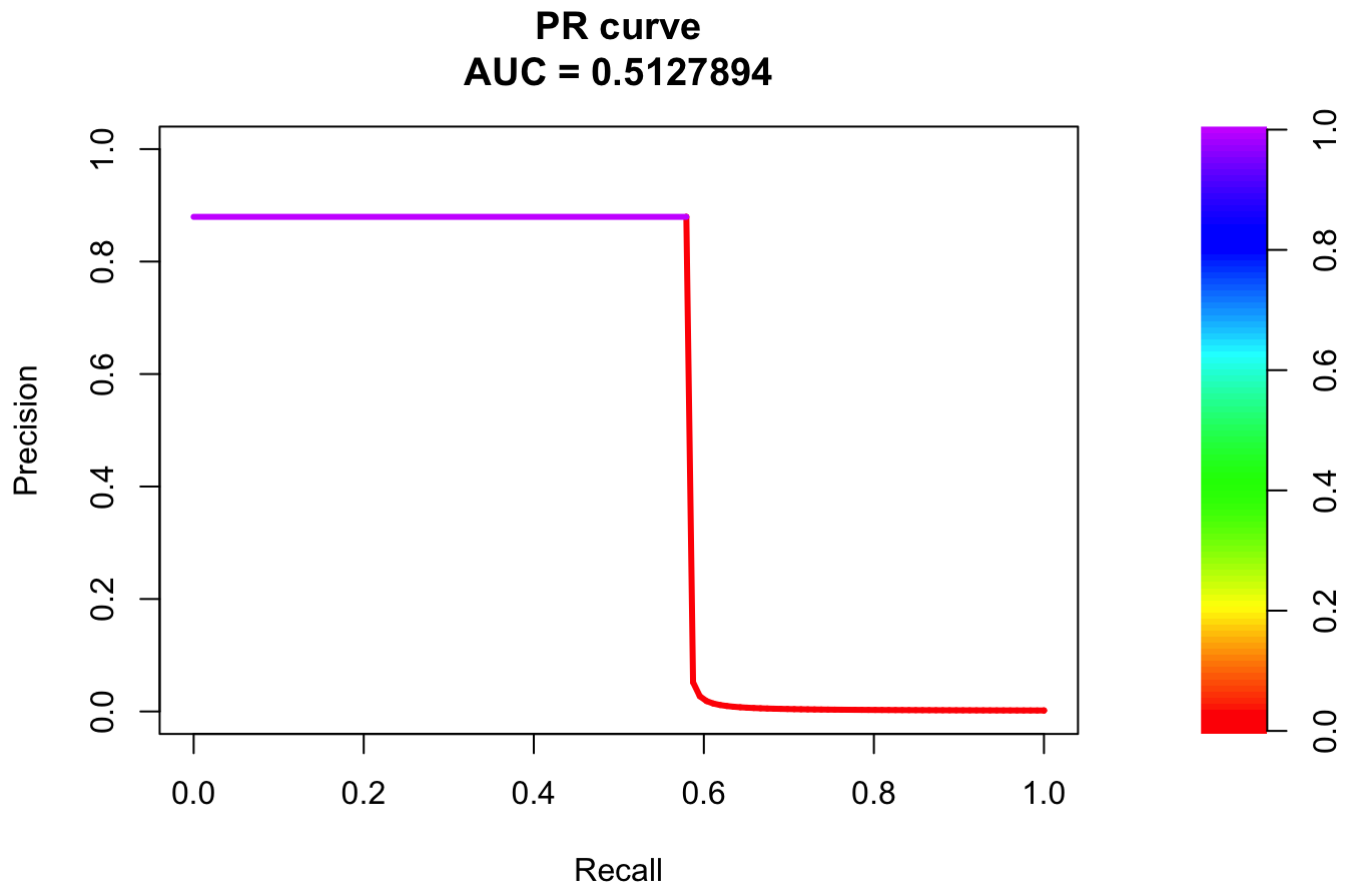
Logistic Regression

The logistic regression is the most basic model to predict a binary response variable. The running time is fast, though the result isn't as accurate.

```
##----------------------------logistic regression----------------------------
m1<-glm(Class~., train,family="binomial")

#apply the model on test dataset to predict
p0<-predict(m1, test, type = "response")

p1 <- ifelse(p0>0.5, 1, 0)


pred1<-prediction(p1,test$Class)


auc1 <- performance(pred1, "auc")
AUPRC1 <- pr.curve(
  scores.class0 = p1[test$Class == 1],
  scores.class1 = p1[test$Class == 0],
  curve = T,
)
# have auc and AUPRC plots
plot(performance(pred1, 'sens', 'spec'), main=paste("AUC:", auc1@y.values))
```




```
plot(AUPRC1)
```



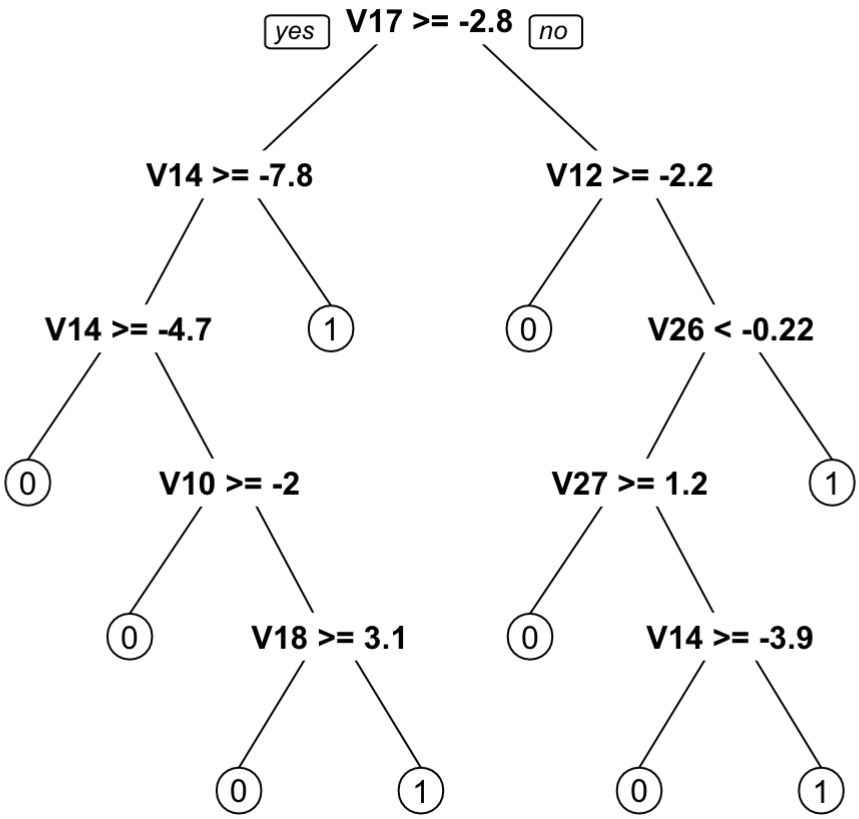

```
# add values to the result dataframe
results <- data.frame(Model = "Logistic Regression", AUC = auc1@y.values[[1]],
                      AUPRC = AUPRC1$auc.integral)
# show results
results %>%
  kable() %>%
  kable_styling(bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE) %>%
  column_spec(2,color = "white" , background ="blue") %>%
  column_spec(3,color = "white" , background ="red")
```

Model	AUC	AUPRC
Logistic Regression	0.7896122	0.5127894

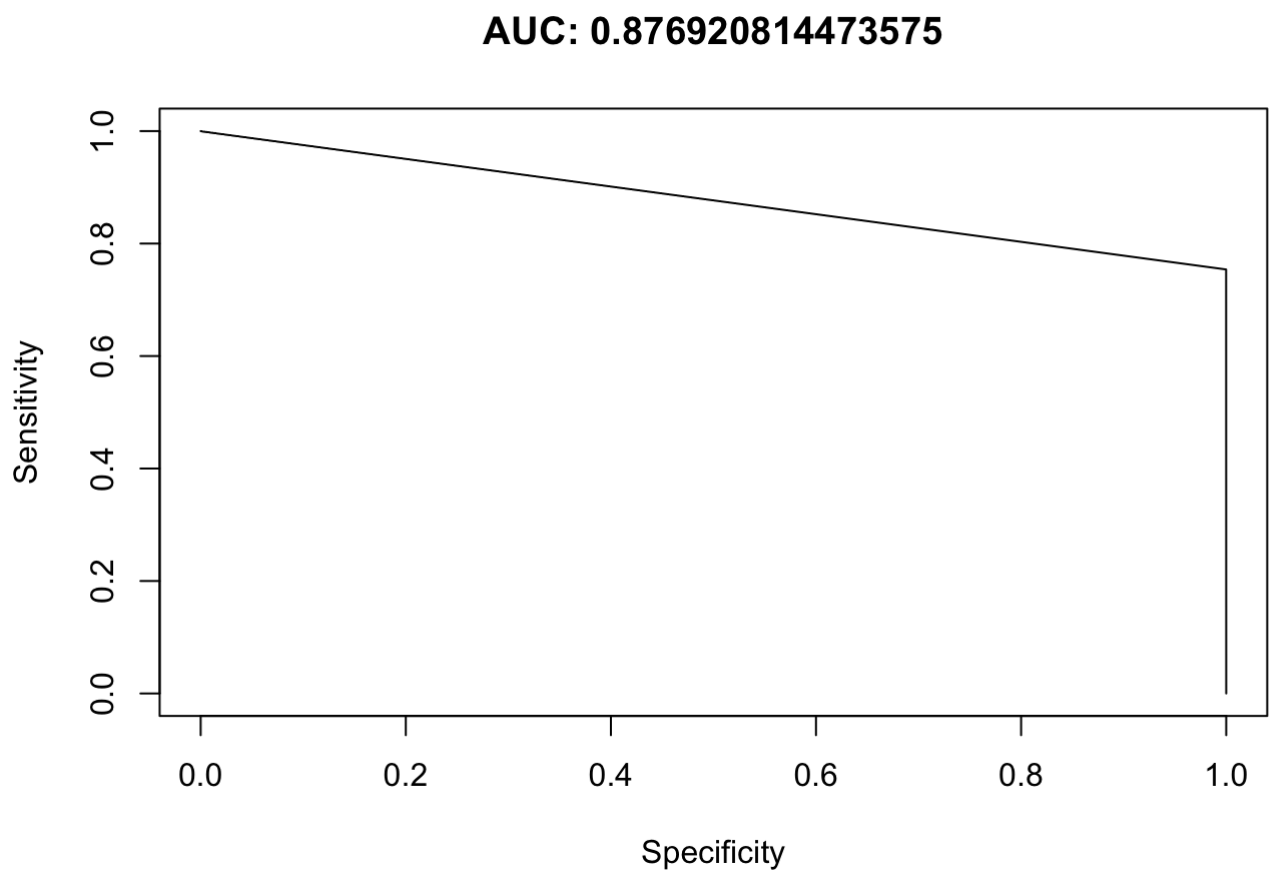
Decision Tree

Decision tree model preforms slightly better than the previous one. Also the model also confirms with the previous assumption that V17,V14, and V12 are the most relevant variables.

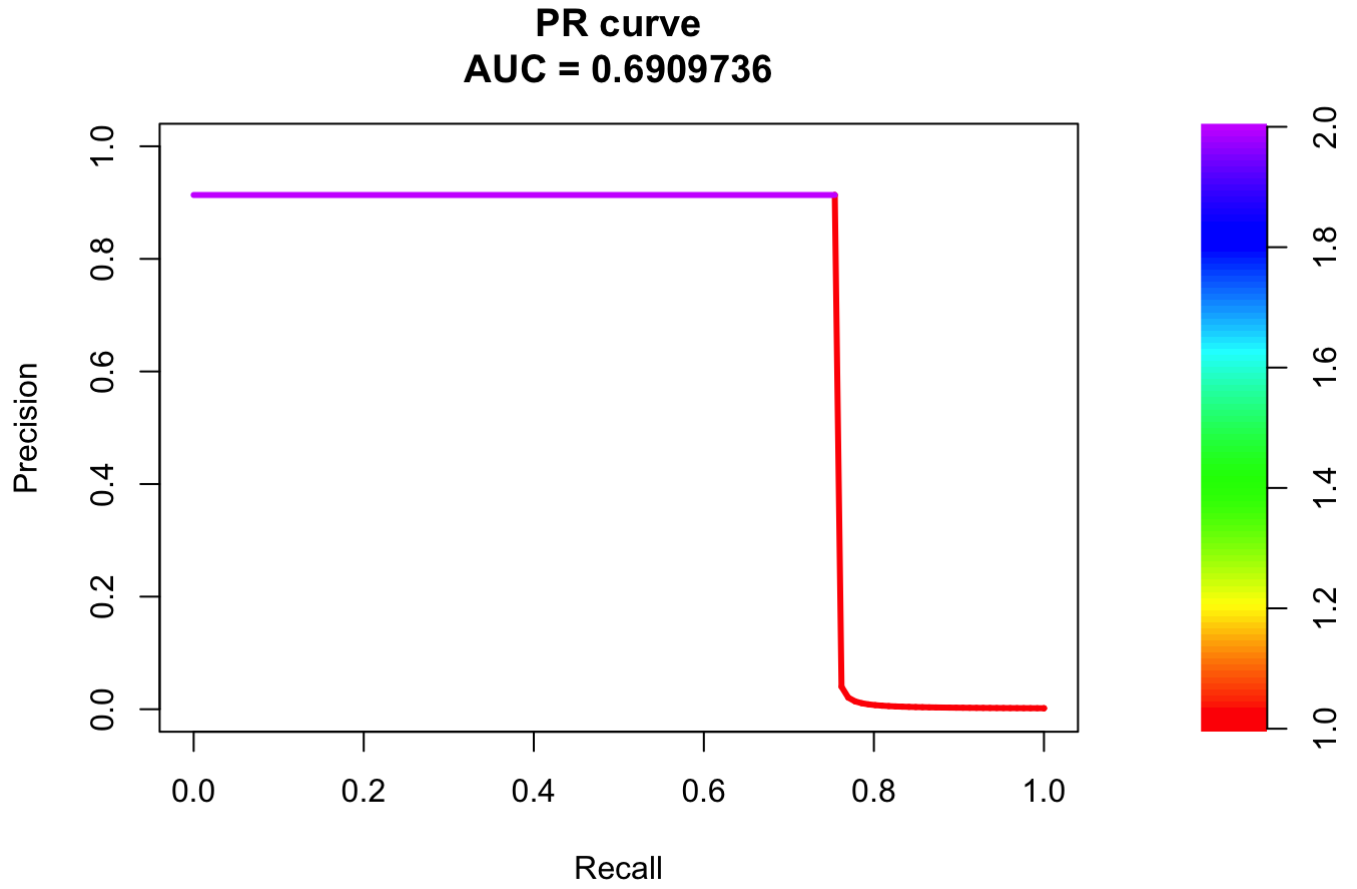
```
##----------------------------decision tree----------------------------
m2<-rpart(Class ~ ., data = train, method = "class", minbucket = 10)

#check the most important variables
prp(m2)
```




```
#apply the model on test dataset to predict
p2<-predict(m2, test, type = "class")

p2<-as.numeric(p2)
pred2<-prediction(p2,test$Class)

auc2 <- performance(pred2, "auc")
AUPRC2 <- pr.curve(
  scores.class0 = p2[test$Class == 1],
  scores.class1 = p2[test$Class == 0],
  curve = T,
)
# have auc and AUPRC plots
plot(performance(pred2, 'sens', 'spec'), main=paste("AUC:", auc2@y.values))
```


AUC: 0.876920814473575
Sensitivity
Specificity
0.0
0.2
0.4
0.6
0.8
1.0

```
plot(AUPRC2)
```


PR curve
AUC = 0.6909736
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
1.0
1.2
1.4
1.6
1.8
2.0

```
# add values to the result dataframe
results <- results %>% add_row(
  Model = "Decision Tree",
  AUC = auc2@y.values[[1]],
  AUPRC = AUPRC2$auc.integral
)

# show results
results %>%
  kable() %>%
  kable_styling(latex_options = "HOLD_position",bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE) %>%
  column_spec(2,color = "white" , background ="blue") %>%
  column_spec(3,color = "white" , background ="red")
```

Model	AUC	AUPRC
Logistic Regression	0.7896122	0.5127894
Decision Tree	0.8769208	0.6909736

Random Forest

Random forest has better results than decision tree model. Though, its running time is probably many times of that. If the ntree value is very big, e.g. 1000, my device will run out of memory. If it's a value of 300, it'd take hours. So for the sake of efficiency, the value is set as 30.

The importance dataframe shows that V17,V14, and V12 are the most relevant variables one more time.

```
##--------------------------random forest----------------------------
m3<-randomForest(Class ~ ., data = train, ntree = 30)
```

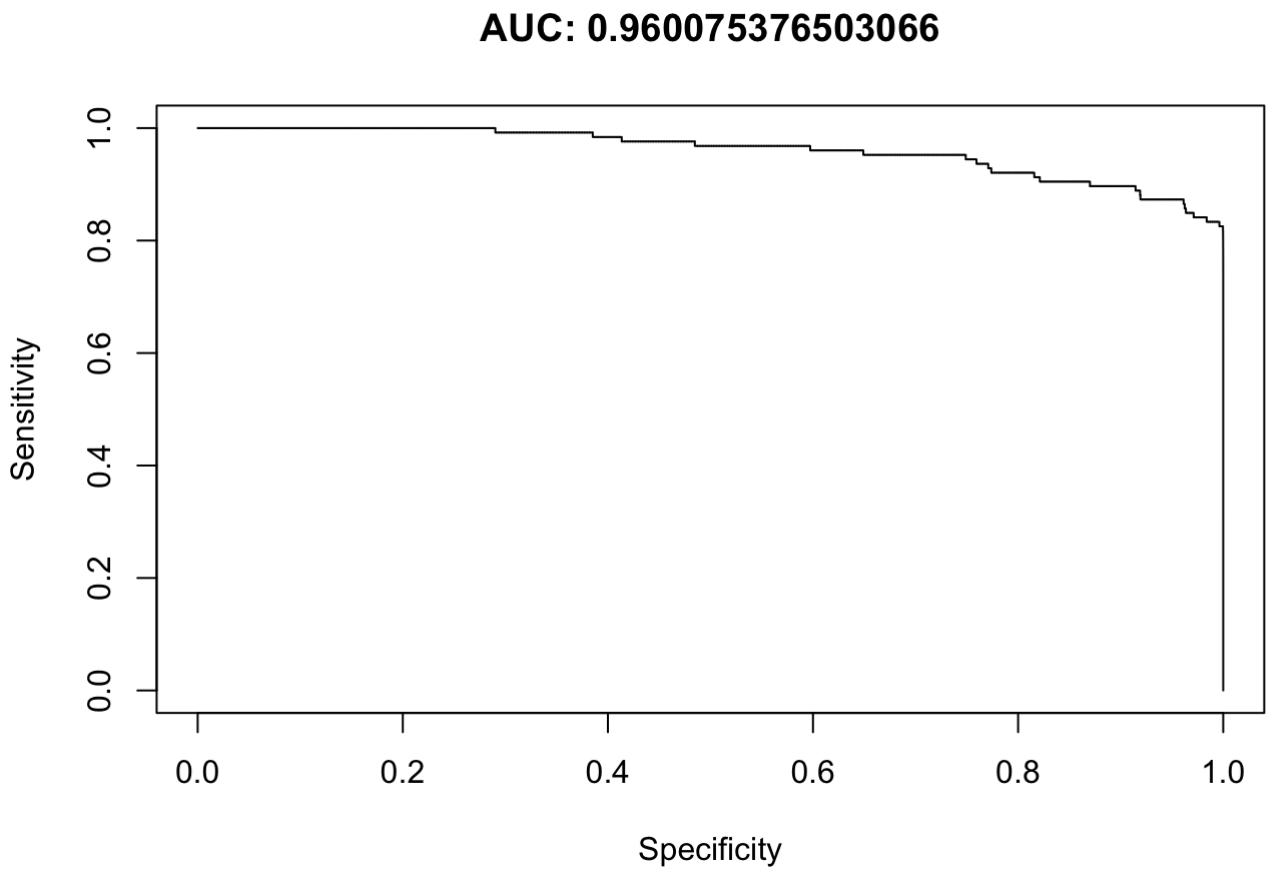
```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values.  Are you sure you want to do regression?
```

```
#check the most significant varaibles
data.frame(importance(m3)) %>% arrange(desc(IncNodePurity)) %>%
  kable() %>%
  kable_styling(latex_options = "HOLD_position",bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE)
```

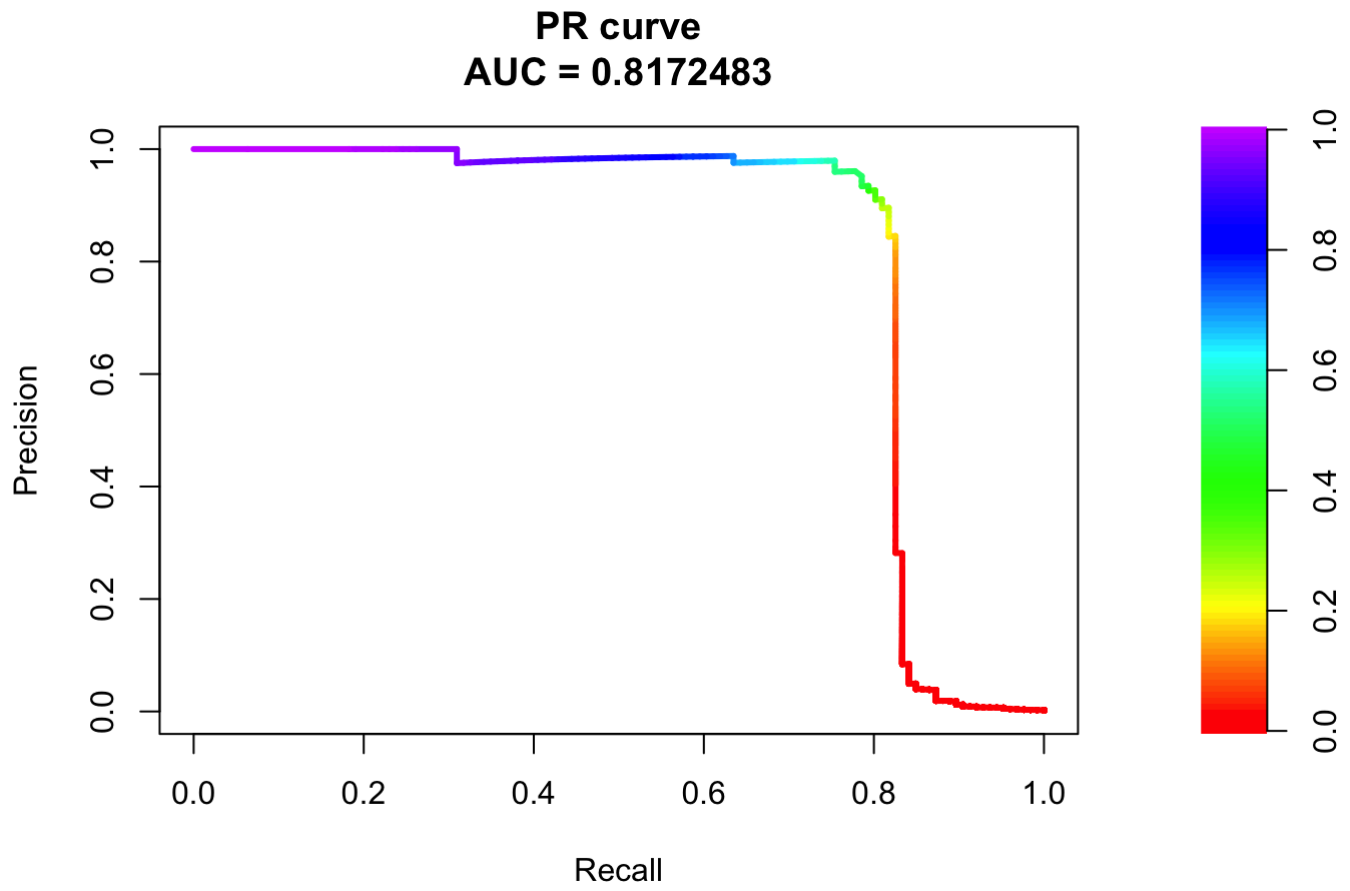
	IncNodePurity
V17	74.715433
V12	61.550632
V14	40.581778
V11	27.334452

	IncNodePurity
V10	23.266537
V16	15.459815
V9	14.723112
V4	9.937463
V18	9.269729
V7	7.827459
V26	7.527030
V21	5.647040
Amount	4.757682
V2	4.636433
V15	4.588658
V6	4.498892
V13	4.080087
V1	3.730087
V22	3.655868
V27	3.635844
V19	3.565862
V8	3.281452
V28	3.185359
V3	3.083444
V20	2.914206
V24	2.778630
V5	2.671633
V25	2.446588
V23	2.090222

```
#apply the model on test dataset to predict
p3<-predict(m3,test)

p3<-as.numeric(p3)
pred3<-prediction(p3,test$Class)
auc3 <- performance(pred3, "auc")
AUPRC3 <- pr.curve(
  scores.class0 = p3[test$Class == 1],
  scores.class1 = p3[test$Class == 0],
  curve = T,
)

# have auc and AUPRC plots
plot(performance(pred3, 'sens', 'spec'), main=paste("AUC:", auc3@y.values))
```




```
plot(AUPRC3)
```



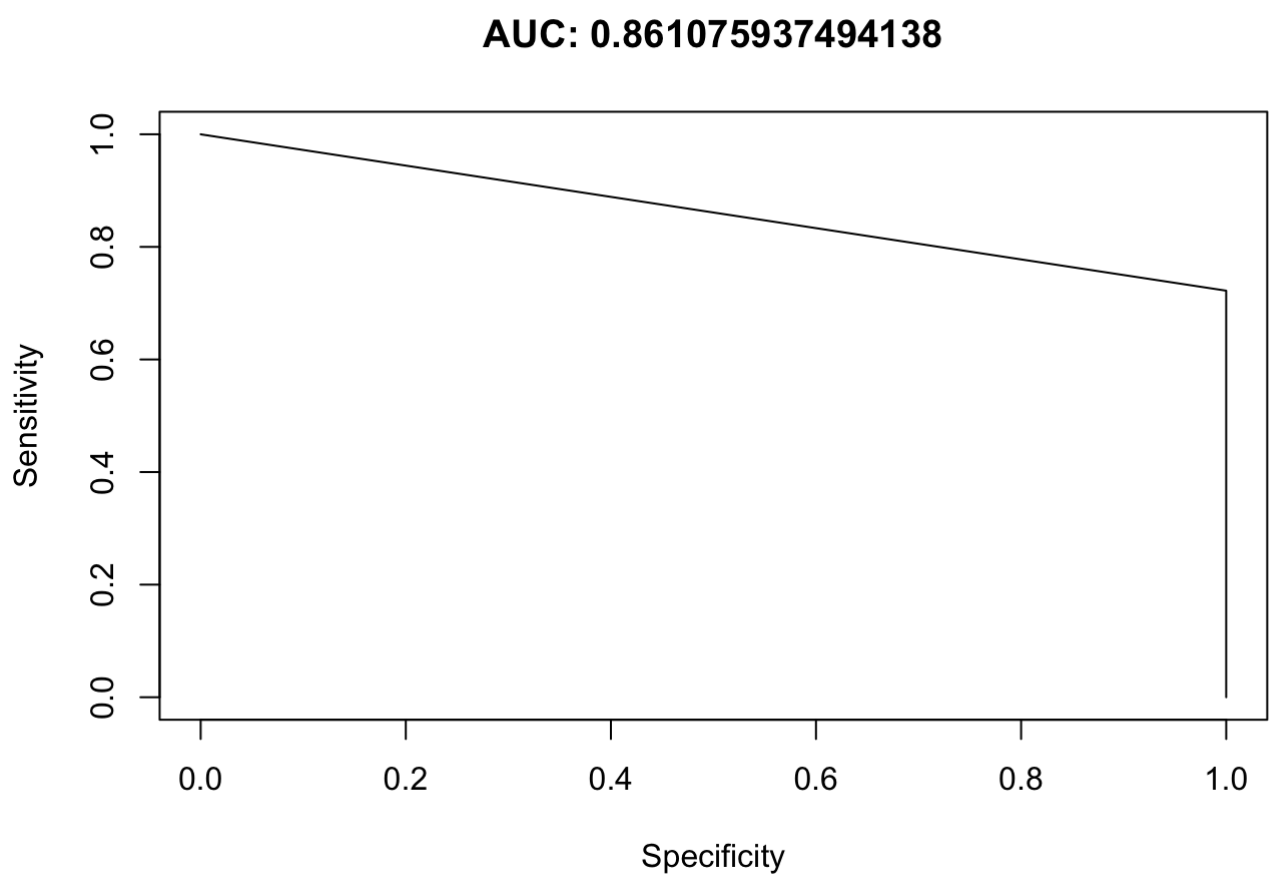

```
# add values to the result dataframe
results <- results %>% add_row(
  Model = "Random Forest",
  AUC = auc3@y.values[[1]],
  AUPRC = AUPRC3$auc.integral
)

# show results
results %>%
  kable() %>%
  kable_styling(latex_options = "HOLD_position",bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE) %>%
  column_spec(2,color = "white" , background ="blue") %>%
  column_spec(3,color = "white" , background ="red")
```

Model	AUC	AUPRC
Logistic Regression	0.7896122	0.5127894
Decision Tree	0.8769208	0.6909736
Random Forest	0.9600754	0.8172483

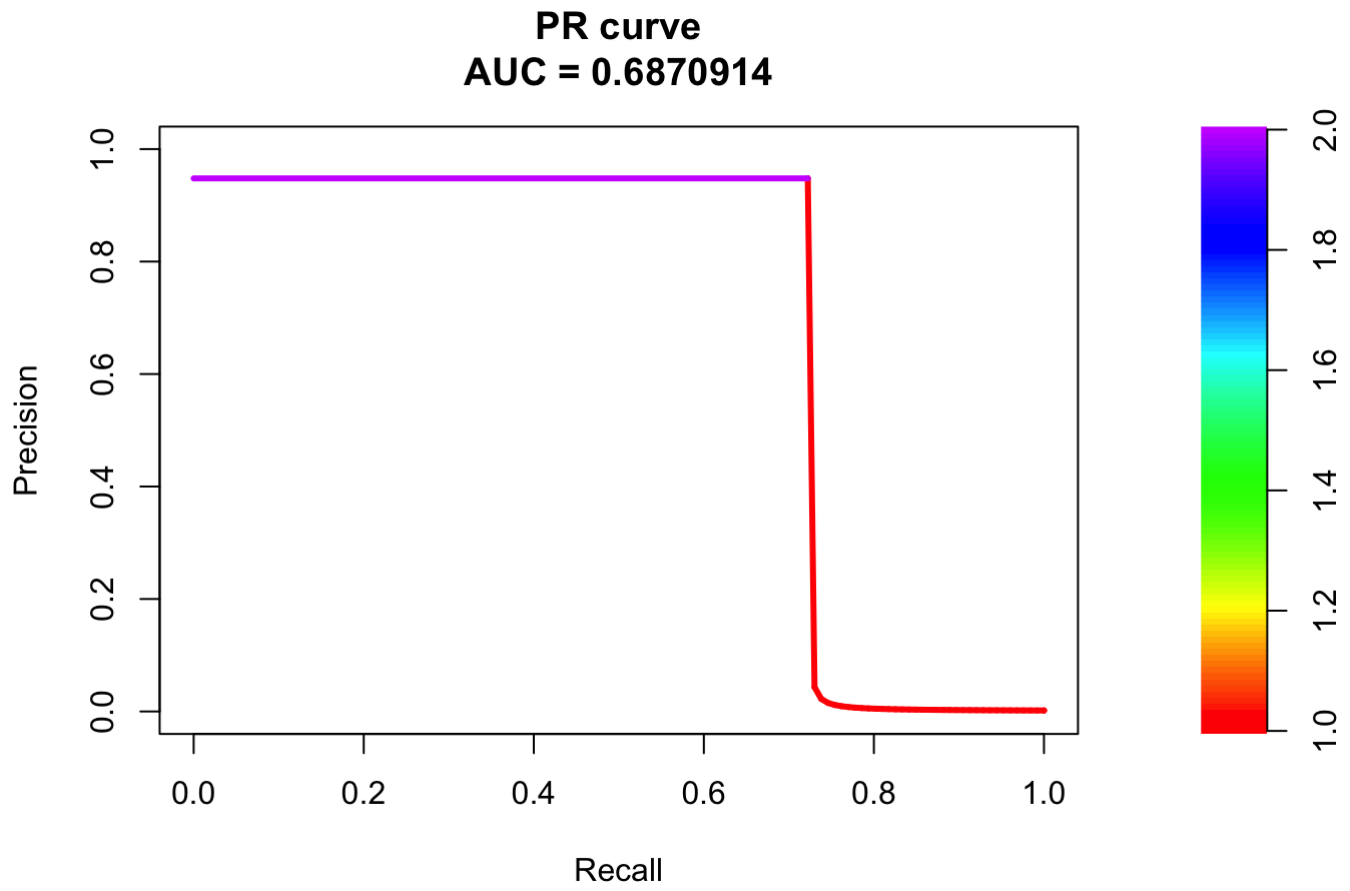
KNN Model

KNN model also takes a lot of time to run, especially with higher number of folds. Though it's overall accuracy isn't as good as Random Forest model, and even slightly worse than Decision Tree model.

```
##---------------------------knn model---------------------------

m4 <- knn(train[,-30], test[,-30], train$Class, k=4, prob = TRUE)

p4<-as.numeric(m4)

# compute the AUC and AUPRC values
pred4<-prediction(p4,test$Class)

auc4 <- performance(pred4, "auc")
AUPRC4 <- pr.curve(
  scores.class0 = p4[test$Class == 1],
  scores.class1 = p4[test$Class == 0],
  curve = T,
)

# have auc and AUPRC plots
plot(performance(pred4, 'sens', 'spec'), main=paste("AUC:", auc4@y.values))
```




```
plot(AUPRC4)
```




```
# add values to the result dataframe
results <- results %>% add_row(
  Model = "KNN Model",
  AUC = auc4@y.values[[1]],
  AUPRC = AUPRC4$auc.integral
)

# show results
results %>%
  kable() %>%
  kable_styling(latex_options = "HOLD_position",bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE) %>%
  column_spec(2,color = "white" , background ="blue") %>%
  column_spec(3,color = "white" , background ="red")
```

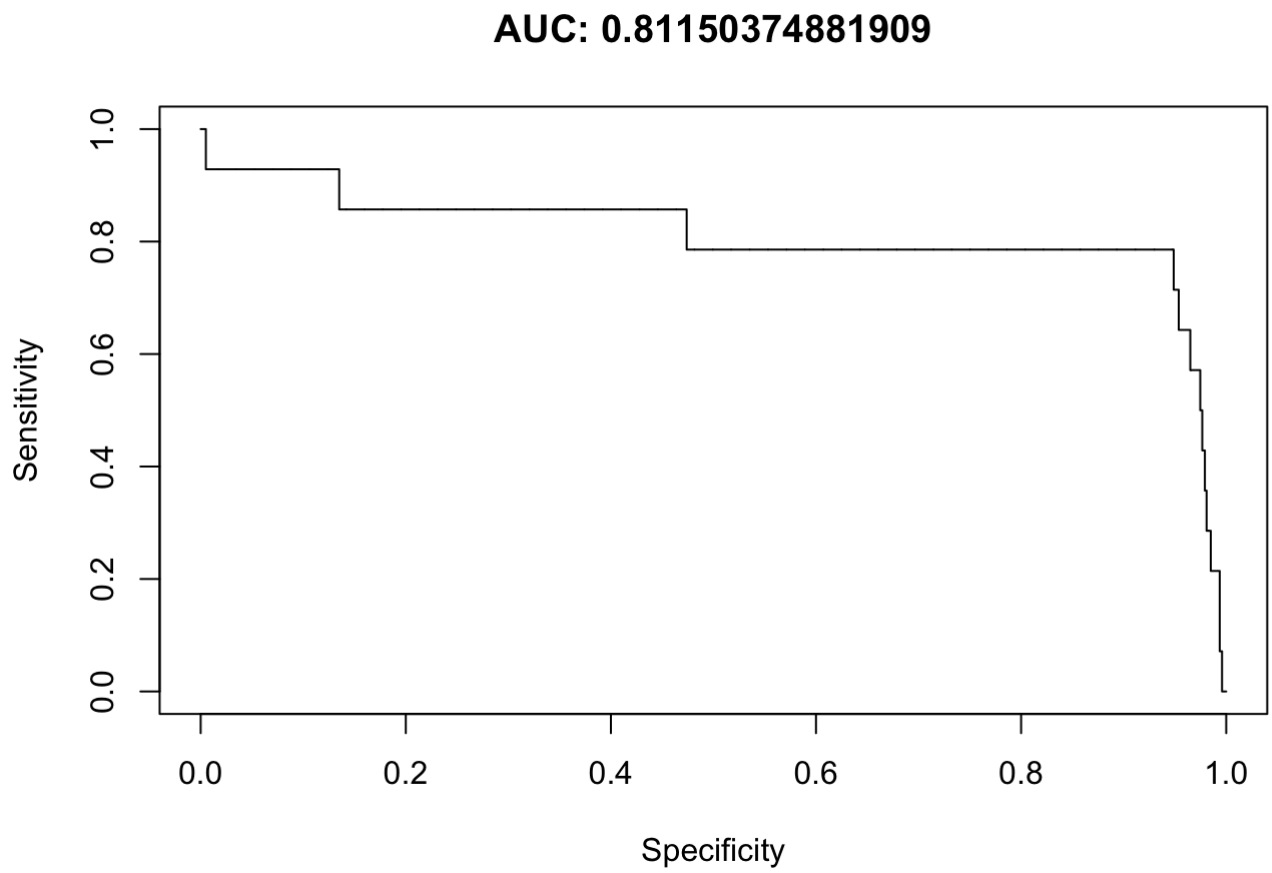
Model	AUC	AUPRC
Logistic Regression	0.7896122	0.5127894
Decision Tree	0.8769208	0.6909736
Random Forest	0.9600754	0.8172483
KNN Model	0.8610759	0.6870914

SVM Model

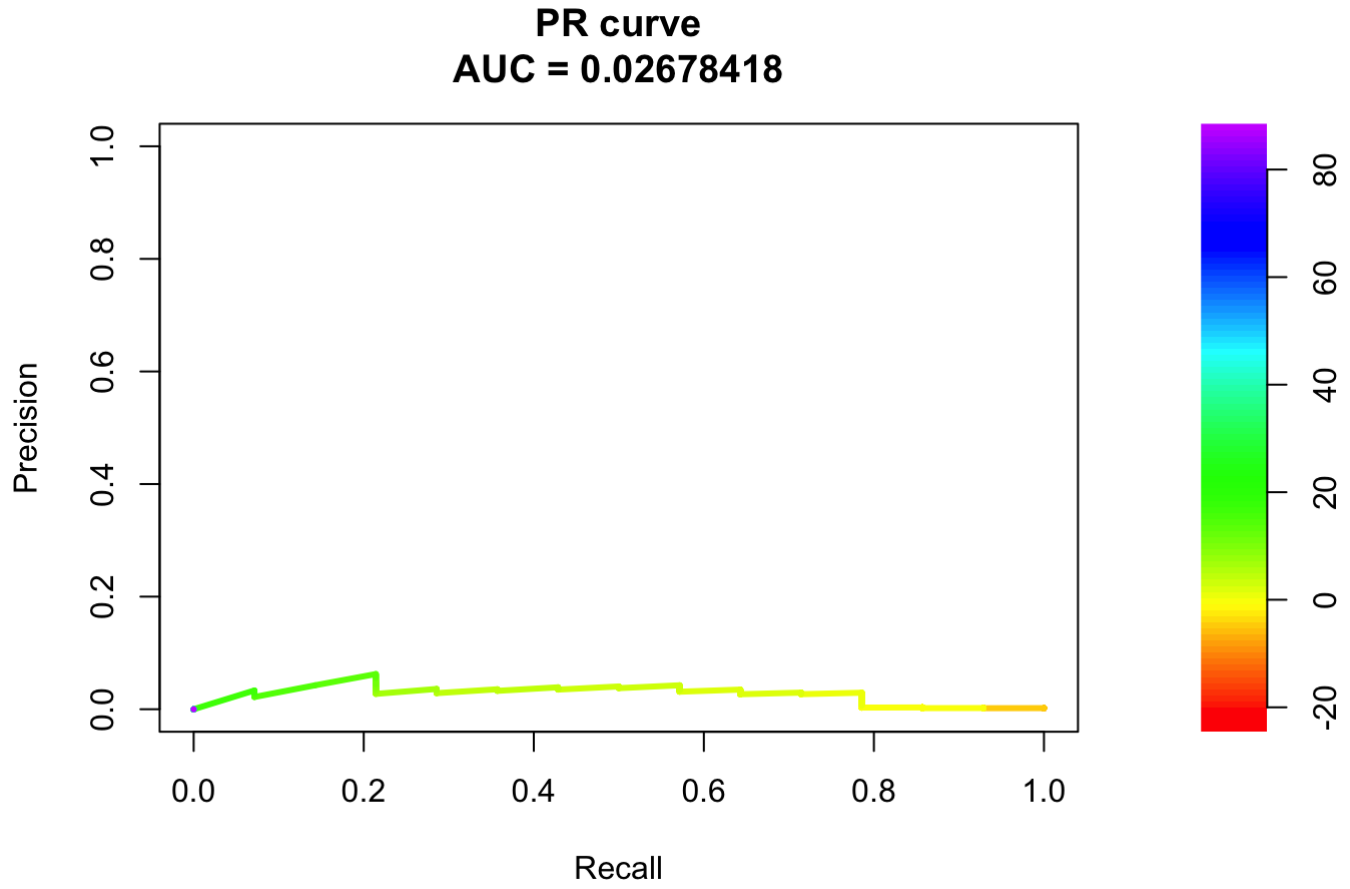
SVM model's running time is the longest, though its accuracy is the lowest. If ksvm function is applied with different thresholds, the result might be better.

Since my computer couldn't deal with the computation power required by training the svm model on the train dataset. Thus only 10% of the original dataset is used to train and test.

```
##---------------------------svm model---------------------------

#split 10% of cc1 into cc5 dataset
split5<-sample.split(cc1$Class, SplitRatio=0.1)
cc5<-subset(cc1, split5 == T)

#have the train and test data for svm model
index5<-sample(1:nrow(cc5),as.integer(0.75*nrow(cc5)))
train5 <- cc5[index5,]
test5 <- cc5[-index5,]

m5<-svm(Class ~ ., data = train5, kernel='sigmoid')

#apply the model on test dataset to predict
p5<-predict(m5,test5)

p5<-as.numeric(p5)
pred5<-prediction(p5,test5$Class)
# compute the AUC and AUPRC values
auc5 <- performance(pred5, "auc")
AUPRC5 <- pr.curve(
  scores.class0 = p5[test5$Class == 1],
  scores.class1 = p5[test5$Class == 0],
  curve = T,
)

# have auc and AUPRC plots
plot(performance(pred5, 'sens', 'spec'), main=paste("AUC:", auc5@y.values))
```


AUC: 0.81150374881909
Sensitivity
Specificity
0.0
0.2
0.4
0.6
0.8
1.0

```
plot(AUPRC5)
```


PR curve
AUC = 0.02678418
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
-20
0
20
40
60
80

```
# add values to the result dataframe
results <- results %>% add_row(
  Model = "SVM Model",
  AUC = auc5@y.values[[1]],
  AUPRC = AUPRC5$auc.integral
)

# show results
results %>%
  kable() %>%
  kable_styling(latex_options = "HOLD_position",bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE) %>%
  column_spec(2,color = "white" , background ="blue") %>%
  column_spec(3,color = "white" , background ="red")
```

Model	AUC	AUPRC
Logistic Regression	0.7896122	0.5127894
Decision Tree	0.8769208	0.6909736
Random Forest	0.9600754	0.8172483
KNN Model	0.8610759	0.6870914
SVM Model	0.8115037	0.0267842

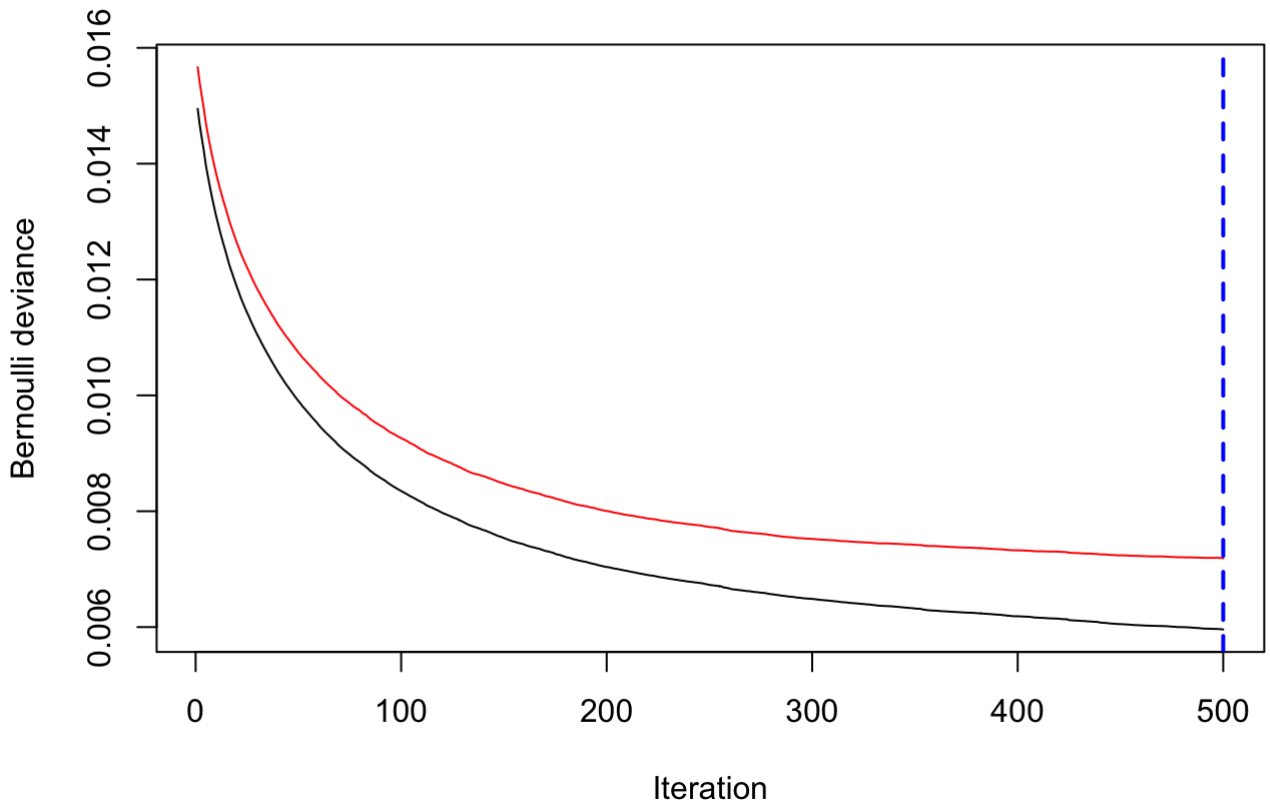
GMB Model

GMB Model is way faster than SVM, Random Forest, and KNN models with a really good overall accuracy result. No wonder gradient boosted machines are extremely popular, boosted by it's high efficiency and accuracy.

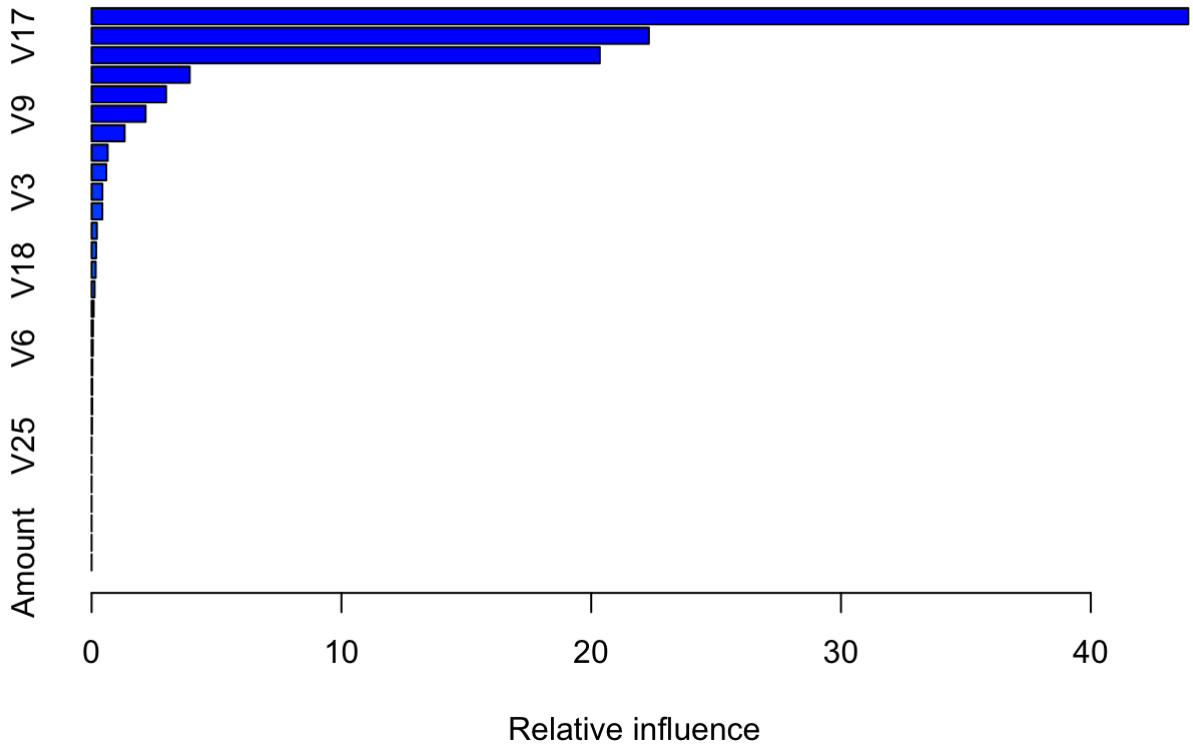
Random forests build an ensemble of deep independent trees, but GBMs build that of shallow and weak successive trees with each tree learning and improving on the previous.

Also, V12,V17, and V14 are three most significant features.

```
##----------------------------gbm model----------------------------

m6<- gbm(as.character(Class) ~ .,
                 distribution = "bernoulli",
                 data = rbind(train, test),
                 n.trees = 500,
                 interaction.depth = 3,
                 n.minobsinnode = 100,
                 shrinkage = 0.01,
                 train.fraction = 0.75,
)

# Determine best iteration based on test data
best.iter = gbm.perf(m6, method = "test")
```



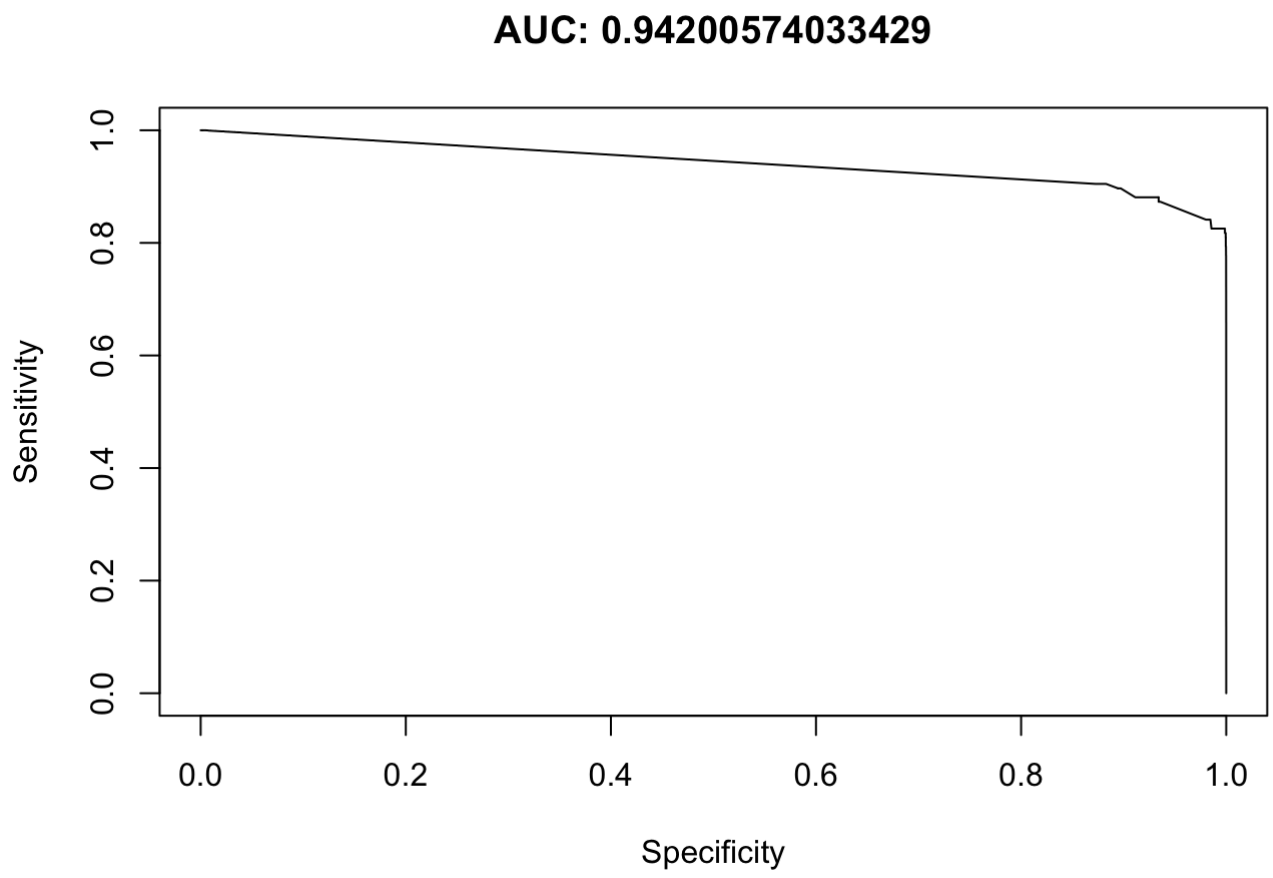

```
# Get feature importance
summary(m6, n.trees = best.iter)
```



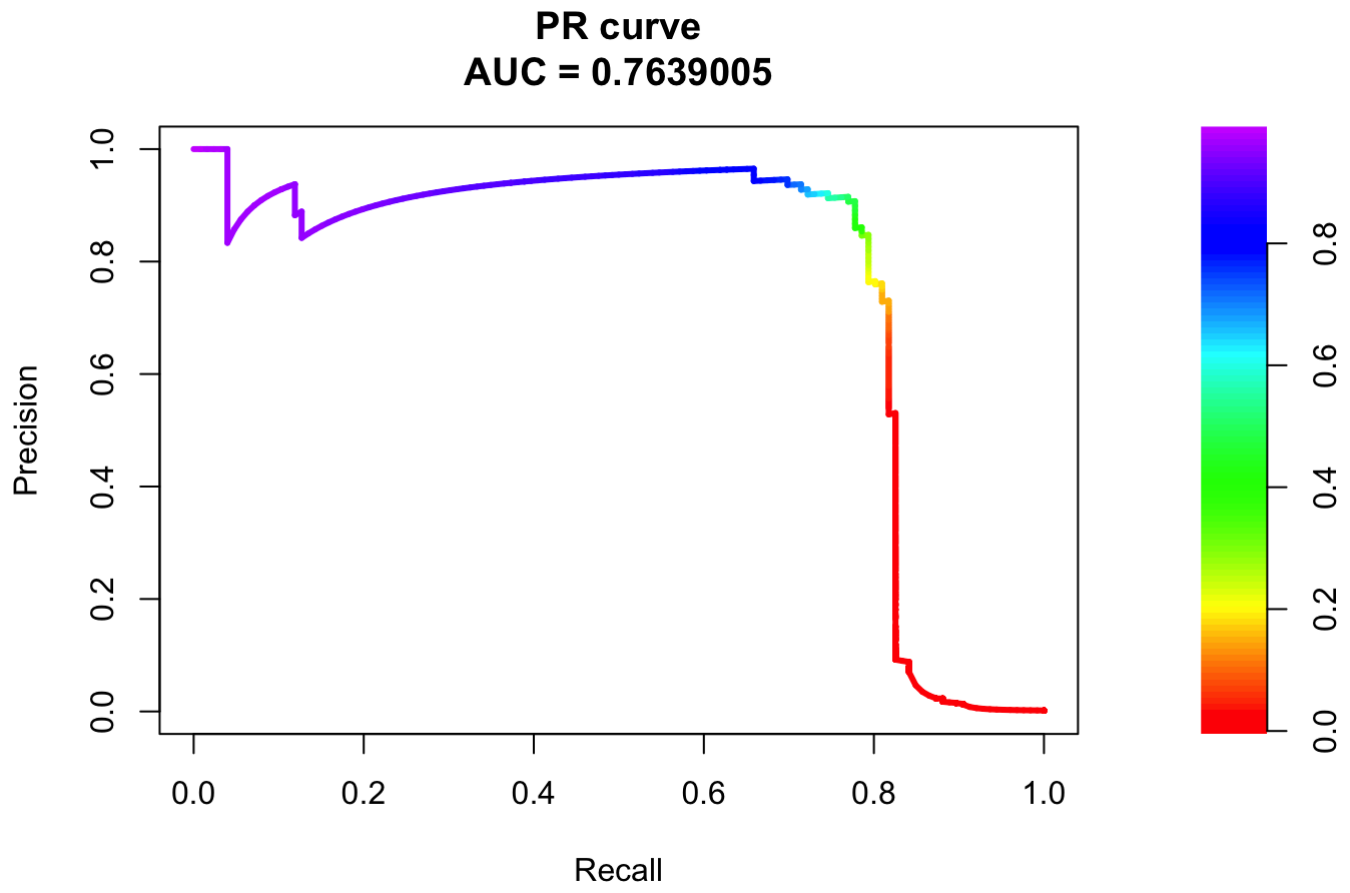
	var <chr>	rel.inf <dbl>
V12	V12	43.900760189
V17	V17	22.309441770
V14	V14	20.342831907
V20	V20	3.927375794
V10	V10	2.980975797
V9	V9	2.159938541
V7	V7	1.321638700
V11	V11	0.643943156
V4	V4	0.585250787
V3	V3	0.430542357

1-10 of 29 rows Previous **1** 2 3 Next

```
# Make predictions based on this model
p6 = predict.gbm(
  m6,
  newdata = test,
  n.trees = best.iter,
  type="response"
)

p6<-as.numeric(p6)
pred6<-prediction(p6,test$Class)

# compute the AUC and AUPRC values

auc6 <- performance(pred6, "auc")
AUPRC6 <- pr.curve(
  scores.class0 = p6[test$Class == 1],
  scores.class1 = p6[test$Class == 0],
  curve = T,
)

# have auc and AUPRC plots
plot(performance(pred6, 'sens', 'spec'), main=paste("AUC:", auc6@y.values))
```




```
plot(AUPRC6)
```




```
# add values to the result dataframe
results <- results %>% add_row(
  Model = "GBM Model",
  AUC = auc6@y.values[[1]],
  AUPRC = AUPRC6$auc.integral
)

# show results
results %>%
  kable() %>%
  kable_styling(latex_options = "HOLD_position",bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE) %>%
  column_spec(2,color = "white" , background ="blue") %>%
  column_spec(3,color = "white" , background ="red")
```

Model	AUC	AUPRC
Logistic Regression	0.7896122	0.5127894
Decision Tree	0.8769208	0.6909736
Random Forest	0.9600754	0.8172483
KNN Model	0.8610759	0.6870914
SVM Model	0.8115037	0.0267842

Model	AUC	AUPRC
GBM Model	0.9420057	0.7639005

LightGBM Model

LightGBM model performs even better that GBM model in efficiency and accuracy. Though, it splits the tree leaf-wise which can lead to overfitting as it produces much complex trees. Nevertheless, we get the highest measure of AUPRC value of 0.8245189 by far.

Also, V14 and V10 are two most significant features.

```
##-------------------------lightgbm-------------------------

# make the training and testing data, and train the model
m7train <- lgb.Dataset(
  as.matrix(train[, colnames(train) != "Class"]),
  label = as.numeric(as.character(train$Class))
)

m7test <- lgb.Dataset(
  as.matrix(test[, colnames(test) != "Class"]),
  label = test$Class
)

m7p = list(
  objective = "binary",
  metric = "binary_error"
)

m7 <- lgb.train(
  params = m7p,
  data = m7train,
  valids = list(test = m7test),
  learning_rate = 0.01,
  nrounds = 500,
  early_stopping_rounds = 40,
  eval_freq = 20
)
```

```
## Warning in lgb.train(params = m7p, data = m7train, valids = list(test =
## m7test), : lgb.train: Found the following passed through '...': learning_rate.
## These will be used, but in future releases of lightgbm, this warning will
## become an error. Add these to 'params' instead. See ?lgb.train for
## documentation on how to call this function.
```

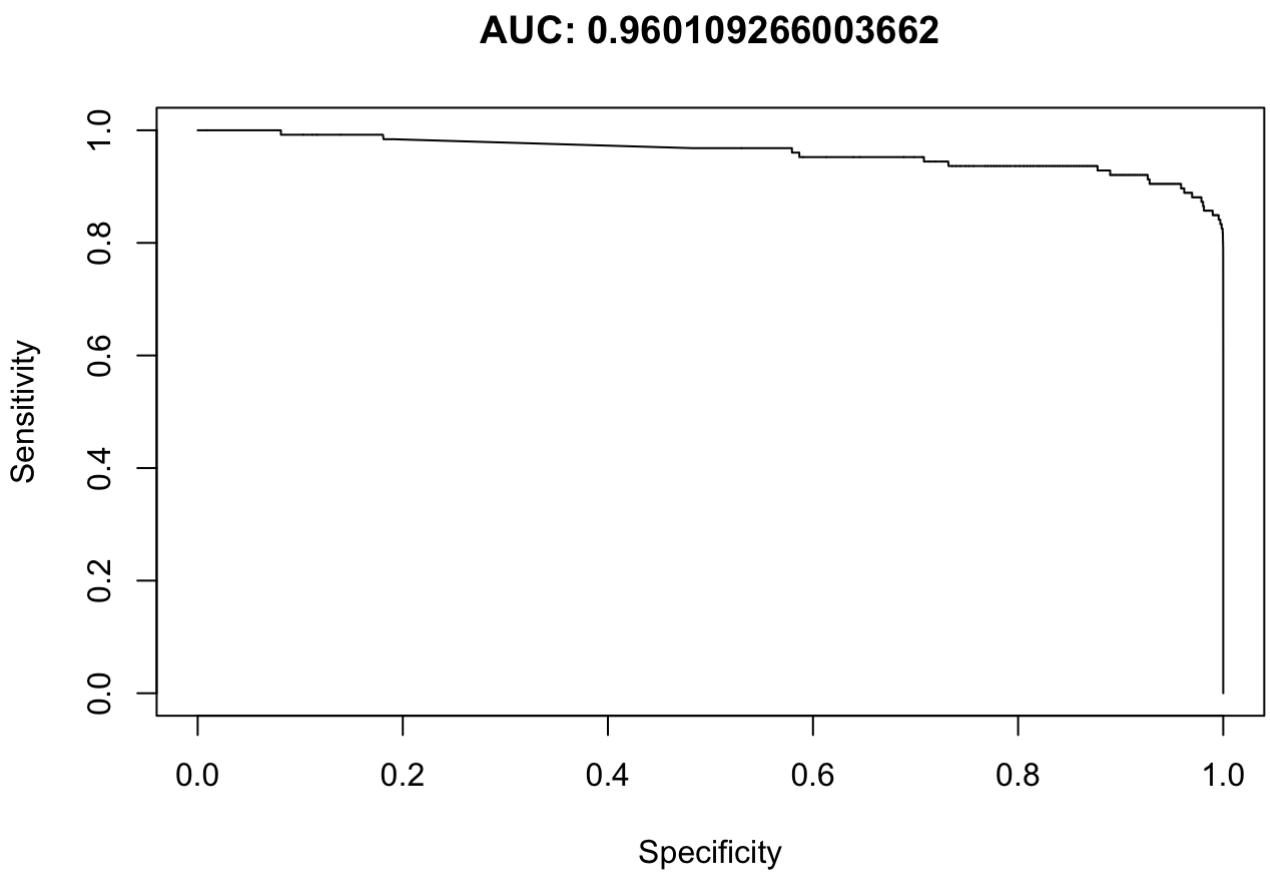
```
## [LightGBM] [Info] Number of positive: 366, number of negative: 213239
## [LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testi
ng was 0.059586 seconds.
## You can set `force_col_wise=true` to remove the overhead.
## [LightGBM] [Info] Total Bins 7395
## [LightGBM] [Info] Number of data points in the train set: 213605, number of used f
eatures: 29
## [LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001713 -> initscore=-6.367536
## [LightGBM] [Info] Start training from score -6.367536
## [1] "[1]:  test's binary_error:0.00176961"
## [1] "[21]:  test's binary_error:0.0014185"
## [1] "[41]:  test's binary_error:0.000814584"
## [1] "[61]:  test's binary_error:0.000702227"
## [1] "[81]:  test's binary_error:0.000660094"
## [1] "[101]:  test's binary_error:0.00061796"
## [1] "[121]:  test's binary_error:0.000589871"
## [1] "[141]:  test's binary_error:0.00061796"
## [1] "[161]:  test's binary_error:0.000561782"
## [1] "[181]:  test's binary_error:0.000547737"
## [1] "[201]:  test's binary_error:0.000533693"
## [1] "[221]:  test's binary_error:0.000519648"
## [1] "[241]:  test's binary_error:0.000519648"
## [1] "[261]:  test's binary_error:0.000505604"
## [1] "[281]:  test's binary_error:0.000519648"
```

```
# Get feature importance
lgb.importance(m7, percentage = TRUE) %>%
  kable() %>%
  kable_styling(bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE)
```

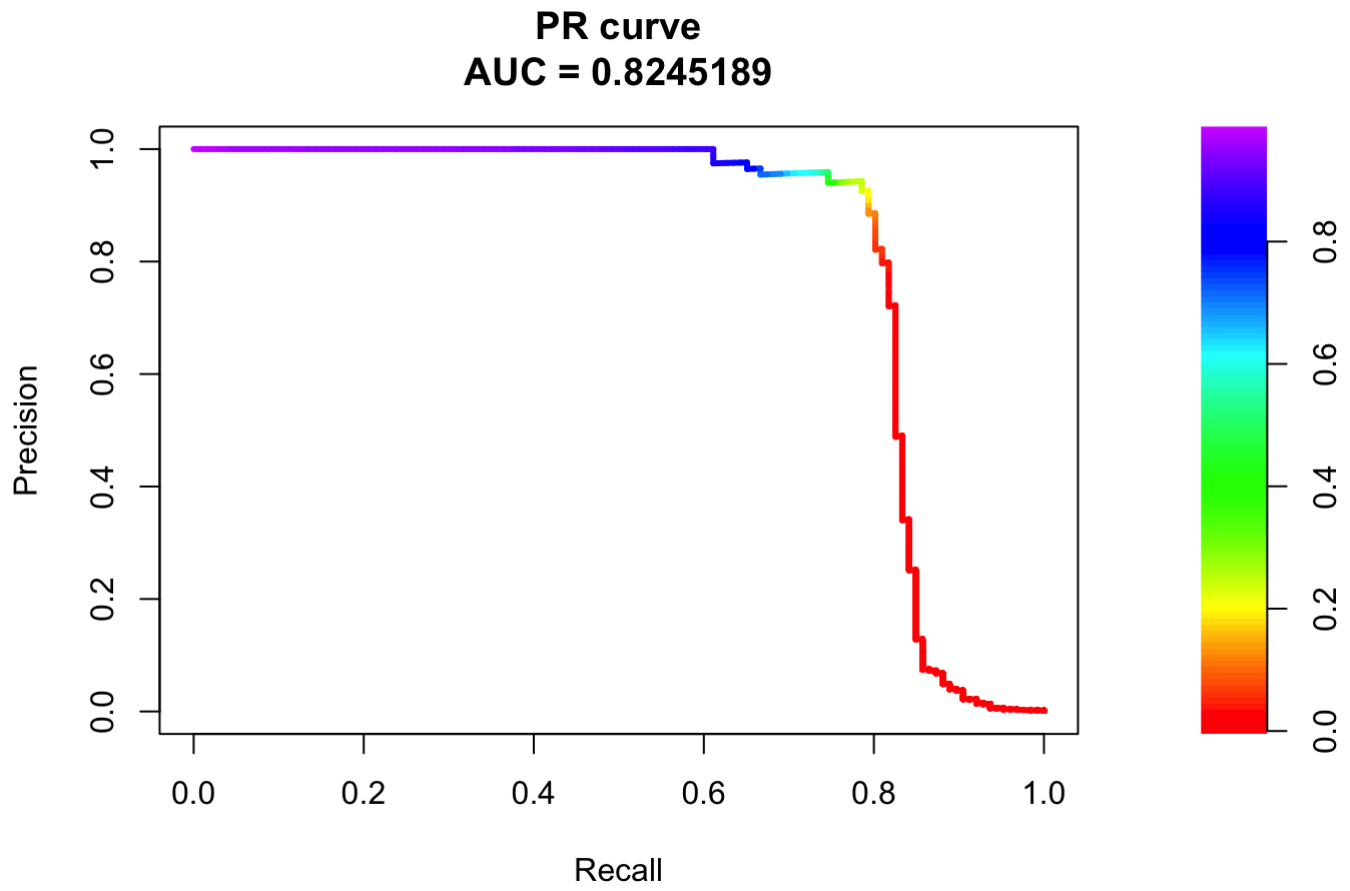
Feature	Gain	Cover	Frequency
V14	0.4272384	0.3074152	0.0629477
V10	0.2908238	0.0112319	0.0420110
V12	0.0328736	0.0195025	0.0477961
V7	0.0318638	0.0623183	0.0391185
V28	0.0256787	0.0012148	0.0327824
V4	0.0254066	0.2630260	0.0922865
V26	0.0239381	0.0183146	0.0681818
V11	0.0129220	0.0056612	0.0268595
V1	0.0124548	0.0014579	0.0195592
Amount	0.0122390	0.1190825	0.0818182

Feature	Gain	Cover	Frequency
V13	0.0112444	0.0220777	0.0538567
V24	0.0103290	0.0008845	0.0212121
V17	0.0075484	0.0055644	0.0165289
V20	0.0069187	0.0223476	0.0297521
V21	0.0067288	0.0712005	0.0418733
V25	0.0065261	0.0087081	0.0366391
V8	0.0062928	0.0067781	0.0417355
V15	0.0060594	0.0024742	0.0238292
V19	0.0058444	0.0020693	0.0268595
V3	0.0056322	0.0059191	0.0252066
V6	0.0049614	0.0009304	0.0155647
V9	0.0044812	0.0017177	0.0209366
V16	0.0039296	0.0071123	0.0276860
V5	0.0037873	0.0015594	0.0234160
V22	0.0034367	0.0260338	0.0235537
V23	0.0032935	0.0027672	0.0170799
V27	0.0028866	0.0005966	0.0152893
V2	0.0028861	0.0018565	0.0183196
V18	0.0017744	0.0001776	0.0073003

```
# Make predictions based on this model
p7 = predict(
  m7,
  data = as.matrix(test[, colnames(test) != "Class"]),
  n = m7$best_iter)

# compute the AUC and AUPRC values
pred7 <- prediction(
  p7,
  test$Class
)


auc7 <- performance(pred7, "auc")
AUPRC7 <- pr.curve(
  scores.class0 = p7[test$Class == 1],
  scores.class1 = p7[test$Class == 0],
  curve = T,
)

# have auc and AUPRC plots
plot(performance(pred7, 'sens', 'spec'), main=paste("AUC:", auc7@y.values))
```




```
plot(AUPRC7)
```




```
# add values to the result dataframe
results <- results %>% add_row(
  Model = "LightGBM Model",
  AUC = auc7@y.values[[1]],
  AUPRC = AUPRC7$auc.integral
)

# show results
results %>%
  kable() %>%
  kable_styling(latex_options = "HOLD_position",bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE) %>%
  column_spec(2,color = "white" , background ="blue") %>%
  column_spec(3,color = "white" , background ="red")
```

Model	AUC	AUPRC
Logistic Regression	0.7896122	0.5127894
Decision Tree	0.8769208	0.6909736
Random Forest	0.9600754	0.8172483
KNN Model	0.8610759	0.6870914
SVM Model	0.8115037	0.0267842

Model	AUC	AUPRC
GBM Model	0.9420057	0.7639005
LightGBM Model	0.9601093	0.8245189

Xgboost Model

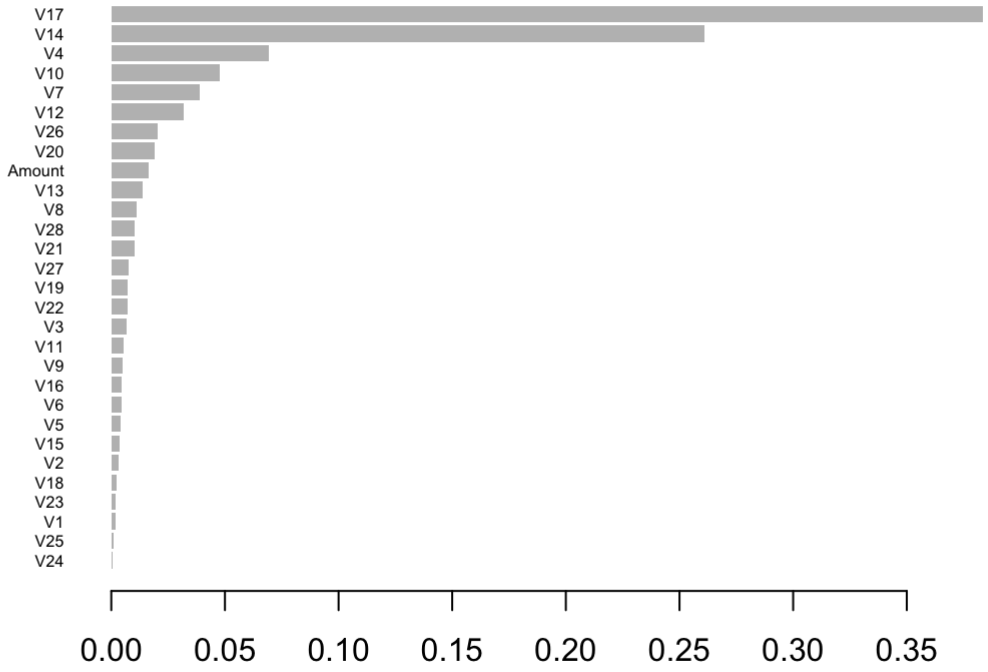
Xgboost model measures the 3rd best value for AUPRC. The result might be better if some of the hyperparameters like learning rate, depth of the trees, and regularization are tuned more. If lightGBM wasn't an option, xgboost model would be picked due to its efficiency over Random Forest.

Also, V17 and V14 are the most important variables in predicting Class.

```
##--------------------------xgboost--------------------------

# make the training and testing data, and train the model
m8train <- xgb.DMatrix(as.matrix(train[, colnames(train) != "Class"]),
                               label = as.numeric(as.character(train$Class)))

m8test <- xgb.DMatrix(as.matrix(test[, colnames(test) != "Class"]),
                              label = as.numeric(as.character(test$Class)))

m8 <- xgb.train(
  data = m8train,
  params = list(objective = "binary:logistic",
                eta = 0.1,
                max.depth = 3,
                nthread = 6,
                eval_metric = "aucpr"),
  watchlist = list(test = m8test),
  nrounds = 500,
  early_stopping_rounds = 40,
  print_every_n = 20
)
```

```
## [1]  test-aucpr:0.726365
## Will train until test_aucpr hasn't improved in 40 rounds.
##
## [21] test-aucpr:0.780936
## [41] test-aucpr:0.793346
## [61] test-aucpr:0.807768
## [81] test-aucpr:0.802385
## [101]    test-aucpr:0.800397
## Stopping. Best iteration:
## [63] test-aucpr:0.809229
```

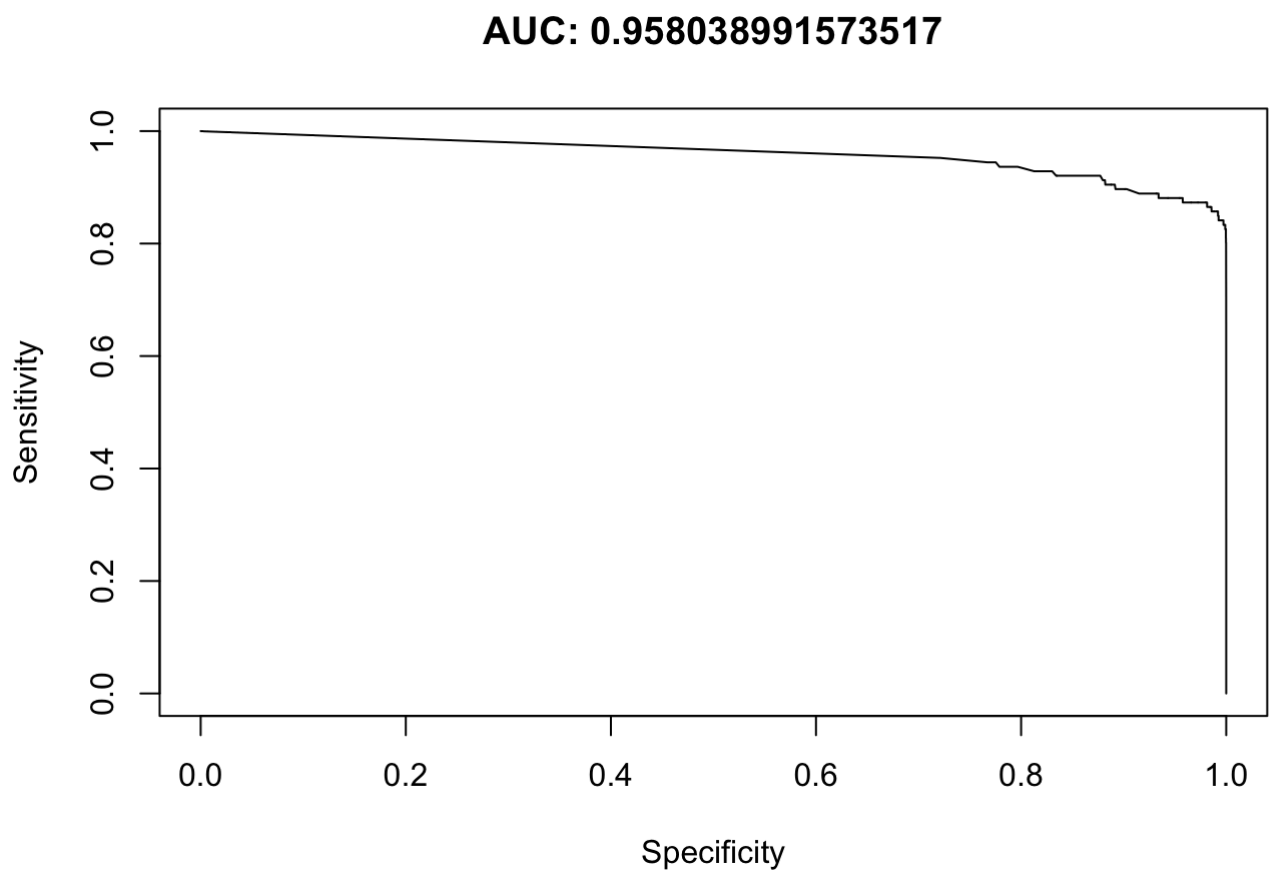
```
# check the importance of the predictors
xgb.plot.importance(xgb.importance(colnames(train$Class),m8))
```



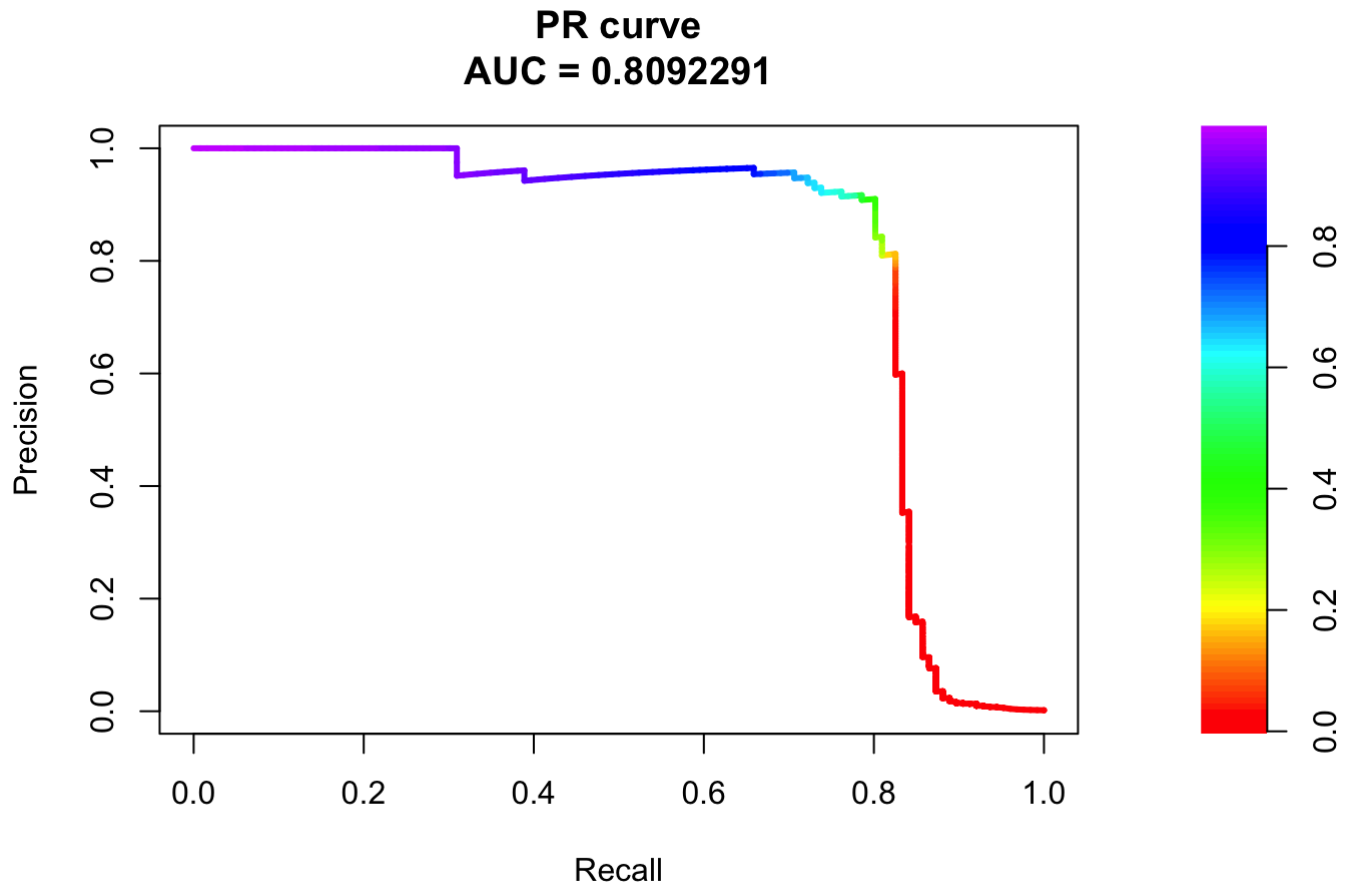

```
p8<-predict(m8,
  newdata = as.matrix(test[, colnames(test) != "Class"]),
  ntreelimit = m8$best_ntreelimit
)
```

```
## [21:08:09] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iter
ation_range` instead.
```

```
# compute the AUC and AUPRC values
pred8<-prediction(p8,test$Class)

auc8 <- performance(pred8, "auc")
AUPRC8 <- pr.curve(
  scores.class0 = p8[test$Class == 1],
  scores.class1 = p8[test$Class == 0],
  curve = T,
)

# have auc and AUPRC plots
plot(performance(pred8, 'sens', 'spec'), main=paste("AUC:", auc8@y.values))
```


AUC: 0.958038991573517
Sensitivity
Specificity
0.0
0.2
0.4
0.6
0.8
1.0

```
plot(AUPRC8)
```


PR curve
AUC = 0.8092291
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

```
# add values to the result dataframe
results <- results %>% add_row(
  Model = "Xgboost Model",
  AUC = auc8@y.values[[1]],
  AUPRC = AUPRC8$auc.integral
)

# show results
results %>%
  kable() %>%
  kable_styling(latex_options = "HOLD_position",bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE) %>%
  column_spec(2,color = "white" , background ="blue") %>%
  column_spec(3,color = "white" , background ="red")
```

Model	AUC	AUPRC
Logistic Regression	0.7896122	0.5127894
Decision Tree	0.8769208	0.6909736
Random Forest	0.9600754	0.8172483
KNN Model	0.8610759	0.6870914
SVM Model	0.8115037	0.0267842
GBM Model	0.9420057	0.7639005
LightGBM Model	0.9601093	0.8245189
Xgboost Model	0.9580390	0.8092291

Neural Network

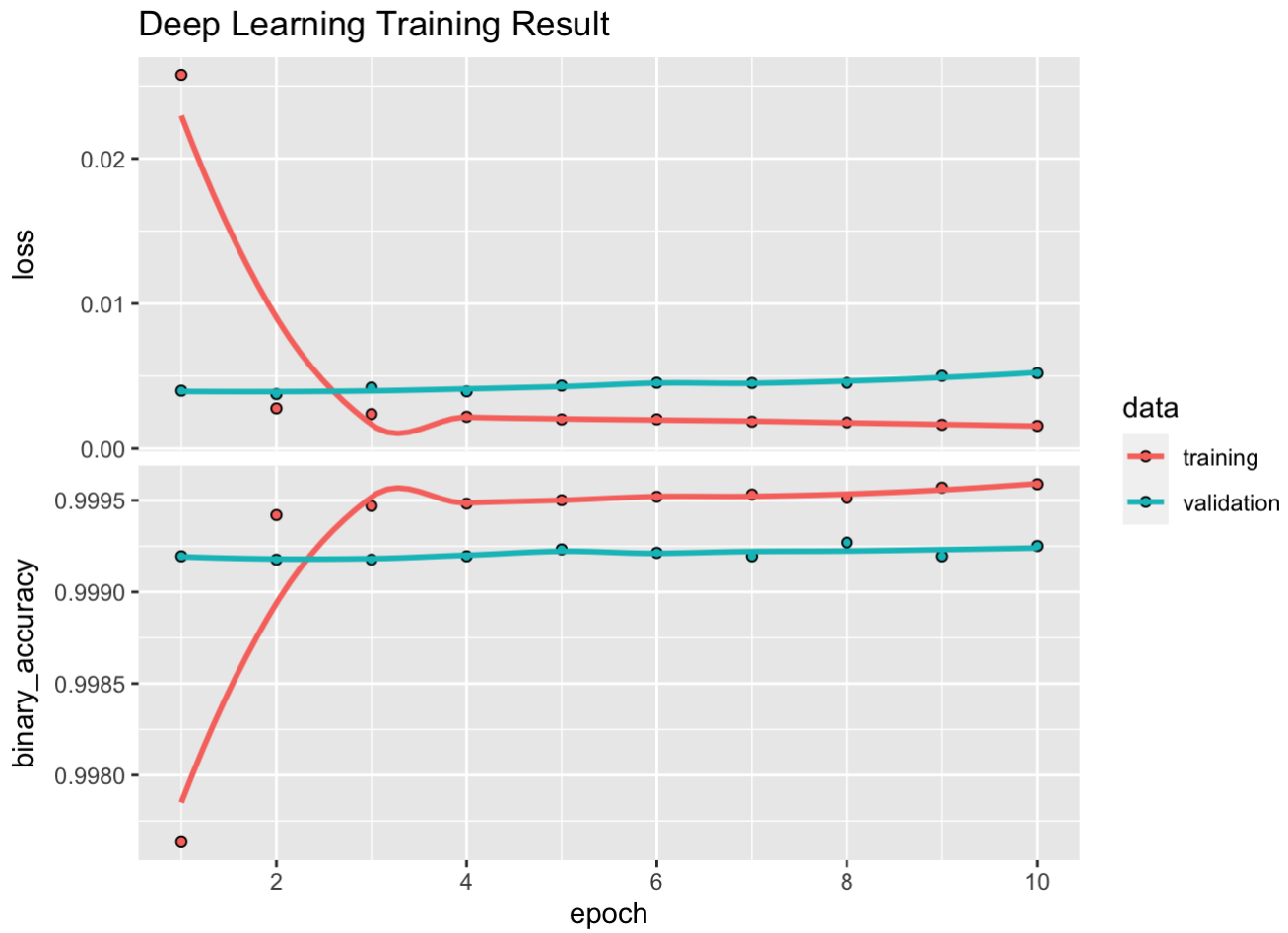
Keras_model_sequential funcion from kera and tensorflow libraries is used to train the neural network model. Though, the kerafit function does give us a validation accuracy around 99.94%, the evaluation part generates the 3rd lowest AUPRC among all models.

The model training is quite straightforward. Three layers are added. No matter how different the values of the first two layer's units are, the evaluation result stays as underperformed in comparision with tree classification and boosting models.

```
##---------------------------neural network---------------------------

X_train <- train[,-30]
X_test <- test[,-30]

y_train <- train[,30]
y_test <- test[,30]

m9 <- keras_model_sequential() %>%
  layer_dense(units = 64, kernel_initializer = "uniform", activation = "relu",
              input_shape = ncol(X_train)) %>%
  layer_dense(units = 32, kernel_initializer = "uniform", activation = "relu") %>%
  layer_dense(units = 1, kernel_initializer = "uniform", activation = "sigmoid")

compile9<-m9 %>% compile(optimizer = "adam",
               loss = "binary_crossentropy",
               metrics = c("binary_accuracy"))

compile9
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## =========================================================================
##  dense_2 (Dense)                    (None, 64)                       1920
##  dense_1 (Dense)                    (None, 32)                       2080
##  dense (Dense)                      (None, 1)                        33
## =========================================================================
## Total params: 4,033
## Trainable params: 4,033
## Non-trainable params: 0
## _________________________________________________________________________
```

```
kerasfit9 <- fit(object = m9,
               x = as.matrix(X_train),
               y = as.numeric(y_train),
               batch_size = 100,
               epochs = 10,
               validation_split = 0.25)

plot(kerasfit9) + labs(title = "Deep Learning Training Result")
```




```
pp9<-predict(object = m9,
             x = as.matrix(X_test),batch_size = 200,verbose = 0) %>% as.vector()
head(pp9,3)
```

```
## [1] 3.144612e-06 1.515744e-07 1.027680e-07
```

```
range(pp9)
```

```
## [1] 0.0000000 0.9996824
```

```
p9 <- ifelse(pp9>0.5, 1, 0)
table(p9)
```

```
## p9
##     0     1
## 71098   104
```

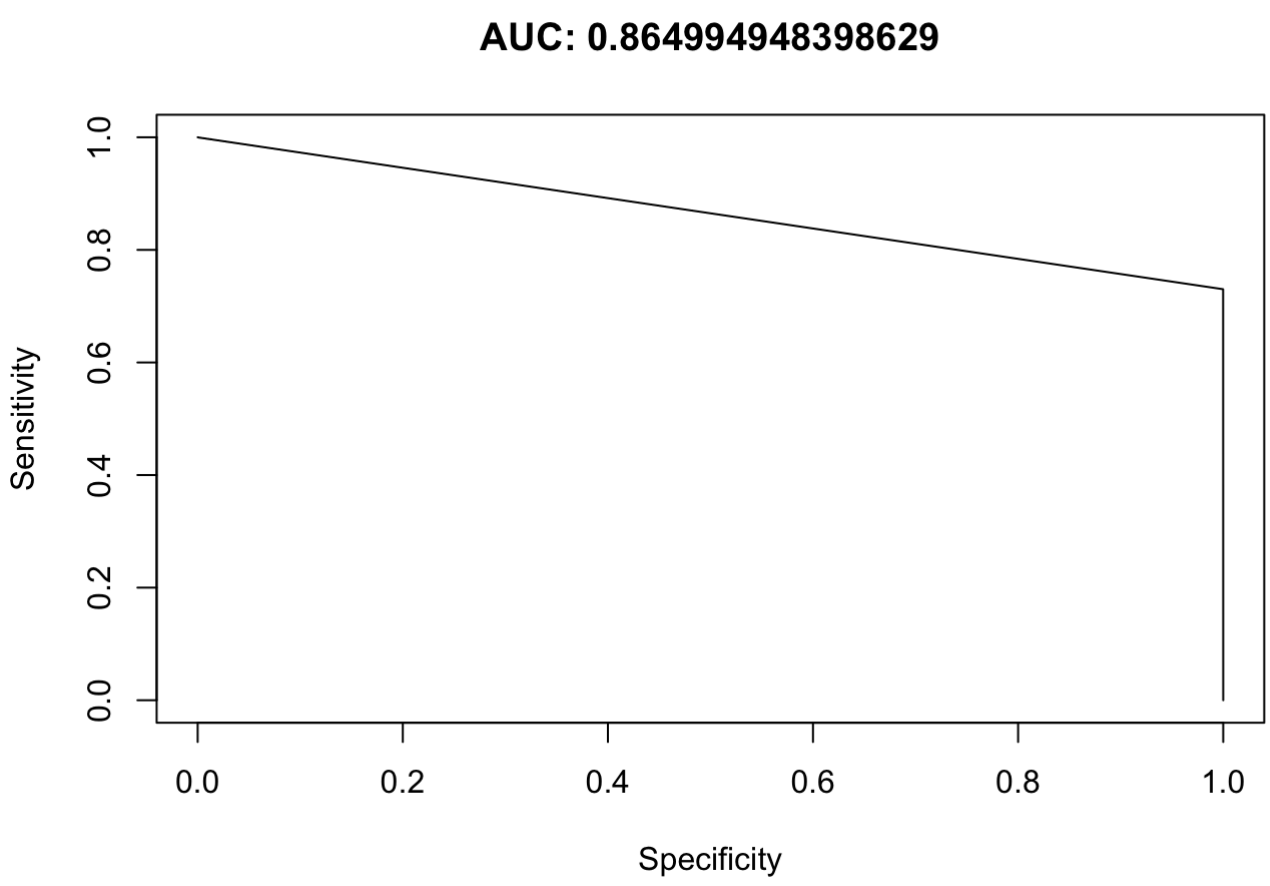
```
p9a<-m9 %>% predict(as.matrix(X_test)) %>% `>`(0.5) %>% k_cast("int32")  %>% as.vecto
r()
table(p9a)
```

```
## p9a
##     0     1
## 71098   104
```

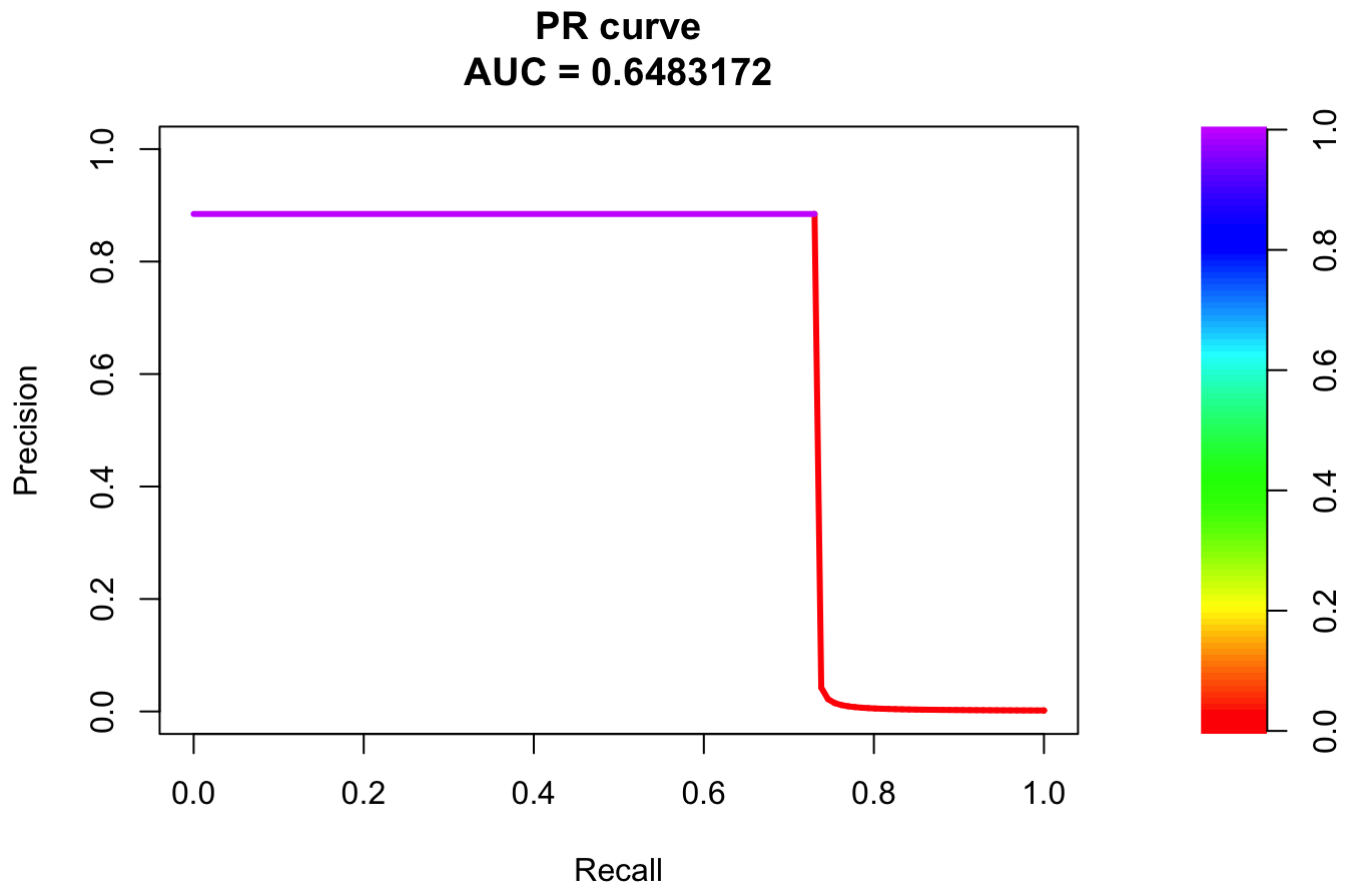
```
table(y_test)
```

```
## y_test
##     0     1
## 71076   126
```

```
# compute the AUC and AUPRC values
pred9<-prediction(p9,as.numeric(y_test))

auc9 <- performance(pred9, "auc")
AUPRC9 <- pr.curve(
  scores.class0 = p9[test$Class == 1],
  scores.class1 = p9[test$Class == 0],
  curve = T,
)

# have auc and AUPRC plots
plot(performance(pred9, 'sens', 'spec'), main=paste("AUC:", auc9@y.values))
```




```
plot(AUPRC9)
```




```
# add values to the result dataframe
results <- results %>% add_row(
  Model = "Neutral Network",
  AUC = auc9@y.values[[1]],
  AUPRC = AUPRC9$auc.integral
)

# show results
results %>%
  kable() %>%
  kable_styling(latex_options = "HOLD_position",bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE) %>%
  column_spec(2,color = "white" , background ="blue") %>%
  column_spec(3,color = "white" , background ="red")
```

Model	AUC	AUPRC
Logistic Regression	0.7896122	0.5127894
Decision Tree	0.8769208	0.6909736
Random Forest	0.9600754	0.8172483
KNN Model	0.8610759	0.6870914
SVM Model	0.8115037	0.0267842

Model	AUC	AUPRC
GBM Model	0.9420057	0.7639005
LightGBM Model	0.9601093	0.8245189
Xgboost Model	0.9580390	0.8092291
Neutral Network	0.8649949	0.6483172

Results

After training and testing of 9 different models, lightGBM captures the best AUPRC value at 0.8245189.

```
# show results
results %>%
  kable() %>%
  kable_styling(latex_options = "HOLD_position",bootstrap_options = "responsive",
                position = "center",
                full_width = FALSE) %>%
  column_spec(2,color = "white" , background ="blue") %>%
  column_spec(3,color = "white" , background ="red")
```

Model	AUC	AUPRC
Logistic Regression	0.7896122	0.5127894
Decision Tree	0.8769208	0.6909736
Random Forest	0.9600754	0.8172483
KNN Model	0.8610759	0.6870914
SVM Model	0.8115037	0.0267842
GBM Model	0.9420057	0.7639005
LightGBM Model	0.9601093	0.8245189
Xgboost Model	0.9580390	0.8092291
Neutral Network	0.8649949	0.6483172

Conclusion

GBM, lightGBM, and XGboost work really well for large dataset both in speed and in accuracy compared with more general classification models. Neural network is extremely fast, and accurate only in keras.fit part. Though, if it gets better tuned, a better accuracy result might be generated.

However, the dataset is very imbalanced, so undersampling, oversampling, and SMOTE technique could be applied to improve the result accuracy for future work.