Name: Chitipolu Sri Sudheera

Tool: R studio

Credit Card Fraud Detection – Imbalanced Data

Context

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.

Content

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.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and

'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Inspiration

Identify fraudulent credit card transactions.

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

Data Preparation

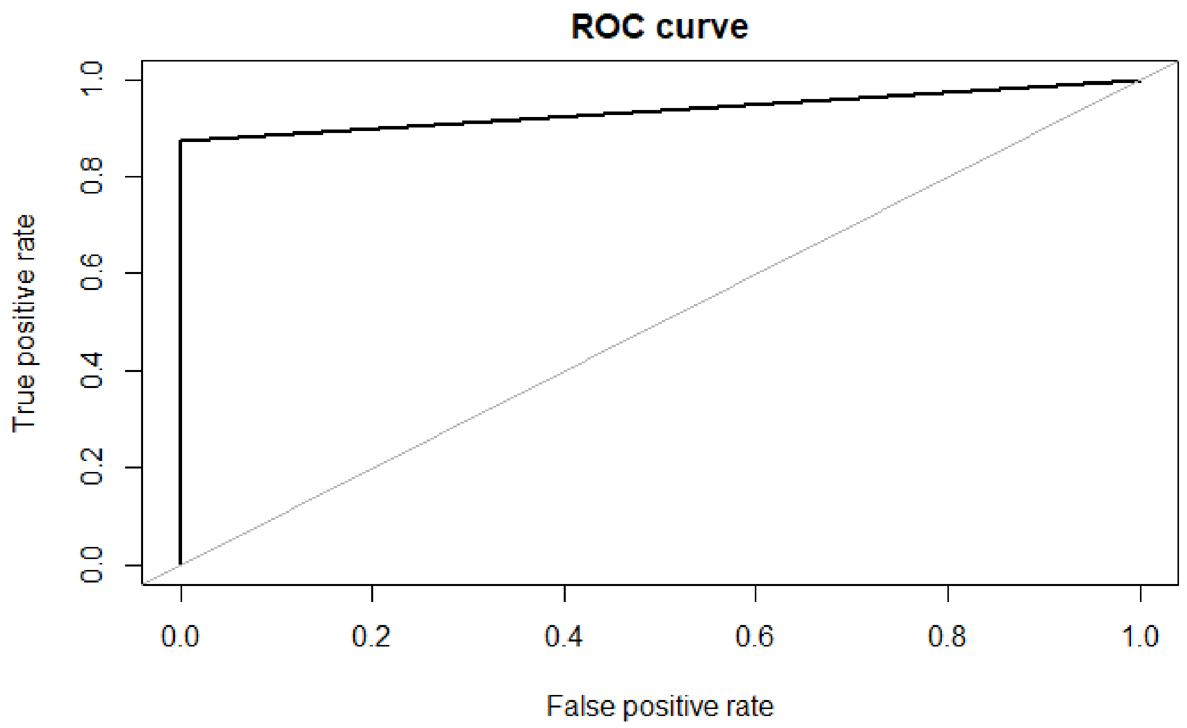
```r
library(rpart)
library(ROSE)
library(rattle)
library(caret)
library(mlbench)
library(caretEnsemble)
library(tidyverse)

mydata <- read.csv("creditcard.csv")
mydata$Class[mydata$Class==1]<- "Yes"
mydata$Class[mydata$Class==0]<- "No"
mydata$Class <- as.factor(mydata$Class)
training_size <- floor(0.80 * nrow(mydata))
train_ind <- sample(seq_len(nrow(mydata)), size = training_size)
training <- mydata[train_ind, ]
testing <- mydata[-train_ind, ]
```

Model Development

Given the imbalanced dataset, we will first evaluate different resampling methods to overcome the skewed distribution in class. We will then build different models and stack them together to increase the overall accuracy.The reason to resample the imbalanced data is because non-fraudulent class made up of 99.8% of the observations, and thus we can simply achieve close to 99.8% accuracy by guessing all the testing observations to be non-fraudulent.

Before we start resampling, let us first look at how CART (classification and regression tree) performs with imbalanced data.

```
#CART Model Performance on imbalanced data
fit1 <- rpart(Class~., data = training)
pred.fit1 <- predict(fit1, newdata = testing)
accuracy.meas(testing$Class, pred.fit1[,2])
##
## Call:
## accuracy.meas(response = testing$Class, predicted = pred.fit1[,
##     2])
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.936
## recall: 0.745
## F: 0.415
roc.curve(testing$Class, pred.fit1[,2])
## Area under the curve (AUC): 0.908
```



The performance is better than expected with 0.92 precision and 0.916 AUC, as we were expecting AUC to be somewhat close to 0.5; however, recall has relatively poorer performance with a value of 0.79, indicating certain amount of false negative predictions. Next, we will examine 3 different resampling approaches to see if we can improve the performance.

Undersampling

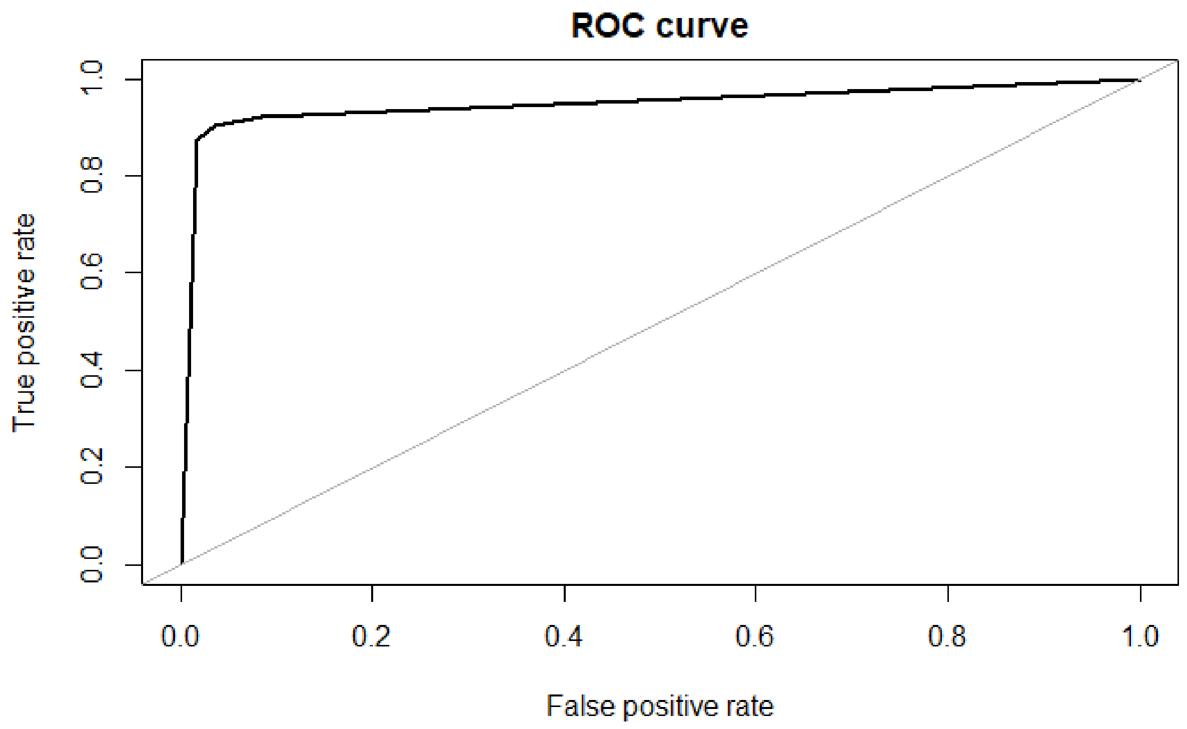
This method reduces the number of observations in the majority class (non-fraudulent transactions) to match the number of observations in the minority class (fraudulent transactions). However, we are also removing great amount of information from the majority class. We will undersample the data so the number of majority observations in the training data is reduced to 403.

```r
data_balanced_under <- ovun.sample(Class ~ ., data = training, method = "unde
r", N = 806, seed = 1)$data

table(data_balanced_under$Class)
```

```
##

##   No Yes

## 412 394
```

```r
fit.under <- rpart(Class ~ ., data = data_balanced_under)

pred.under <- predict(fit.under, newdata = testing)

roc.curve(testing$Class, pred.under[,2])
```

```
## Area under the curve (AUC): 0.956
```



Oversampling

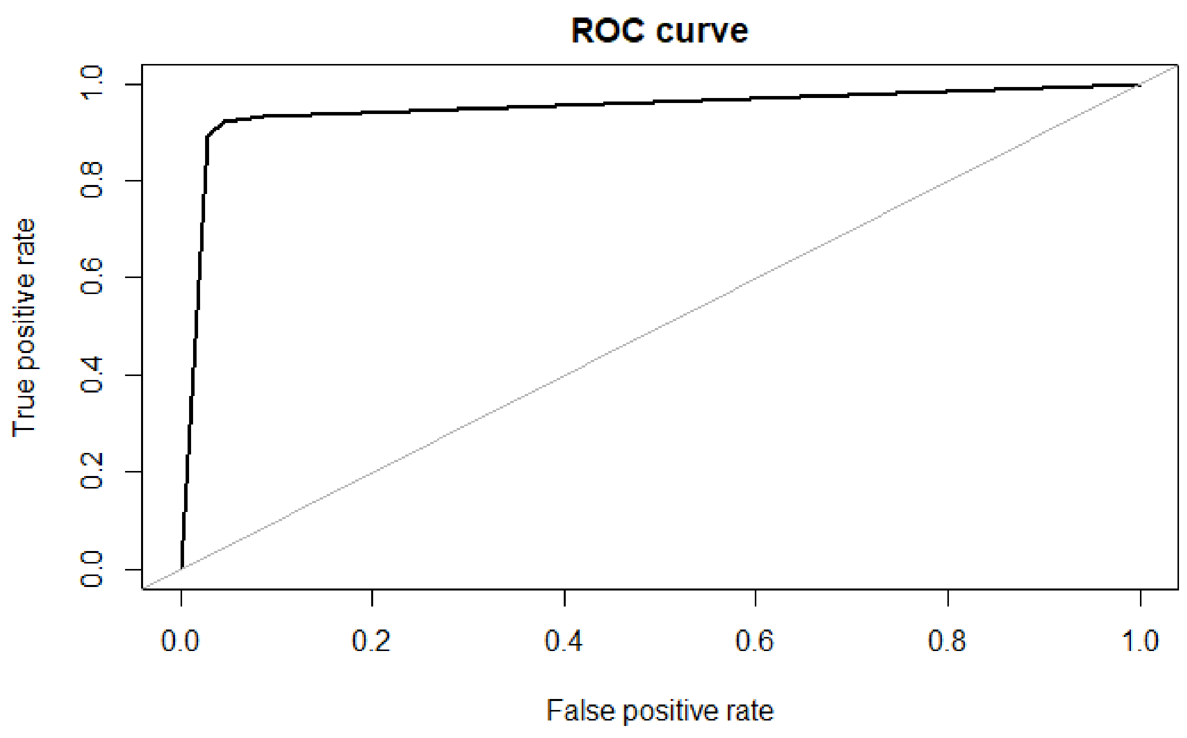
This method replicates observations from the minority class to match the number of observations in the majority class. By doing so we can avoid information loss, however, it might lead to overfitting from the duplicated minority class training data, and thus the accuracy on testing (uneseen) observations could be affected. We will oversample the data so the number of minority observations in the training data is increased to 227,442. The performance is slightly better than undersampling.

```
data_balanced_over <- ovun.sample(Class ~ ., data = training, method = "over"
,N = 454884)$data

table(data_balanced_over$Class)
```

```
##

##      No     Yes

## 227451 227433
```

```
fit.over <- rpart(Class ~ ., data = data_balanced_over)

pred.over <- predict(fit.over, newdata = testing)

roc.curve(testing$Class, pred.over[,2])
```

```
## Area under the curve (AUC): 0.960
```



Both

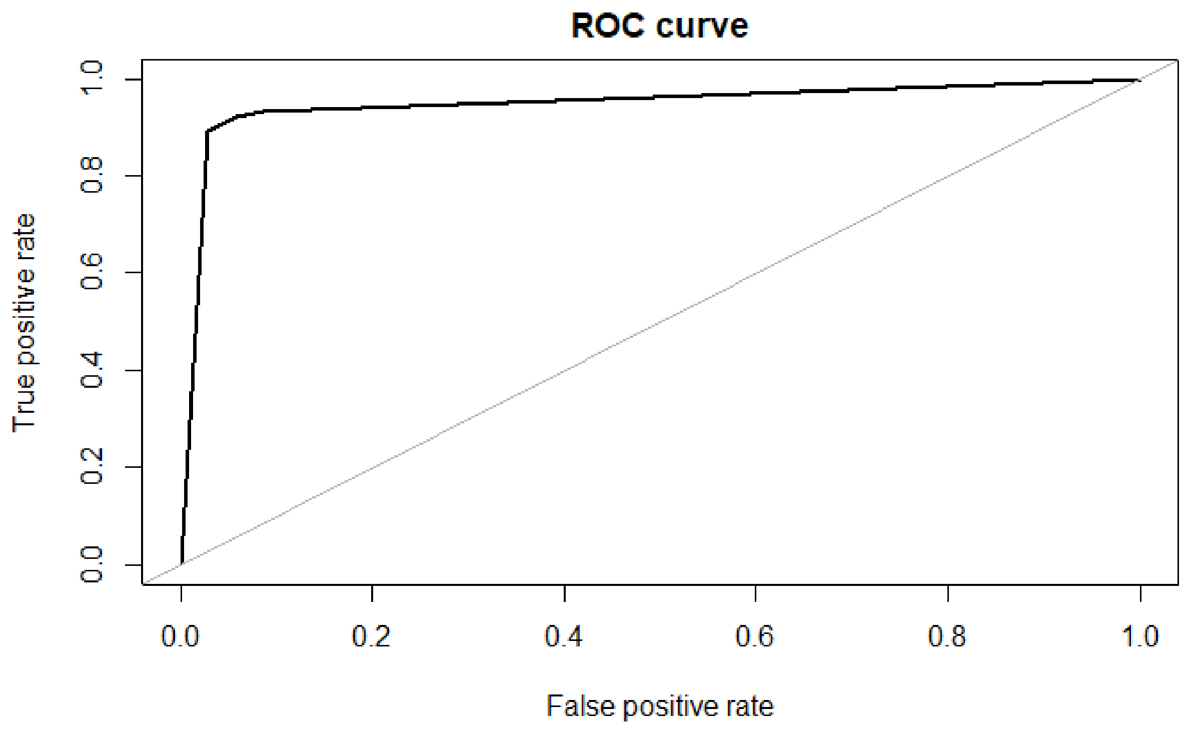
Now, we will combine both undersampling and oversampling to see if the performance improves. That is, we will undersample and majority class and oversample the minority class at the same time, to obtain 5,000:5,000 observations. The result is better than undersampling and oversampling.

```
data_balanced_both <- ovun.sample(Class ~ ., data = training, method = "both"
, p=0.5, N=10000, seed = 1)$data

table(data_balanced_both$Class)
```

```
##

##   No  Yes

## 5047 4953
```

```
fit.both <- rpart(Class ~ ., data = data_balanced_both)

pred.both <- predict(fit.both, newdata = testing)

roc.curve(testing$Class, pred.both[,2])
```

```
## Area under the curve (AUC): 0.955
```

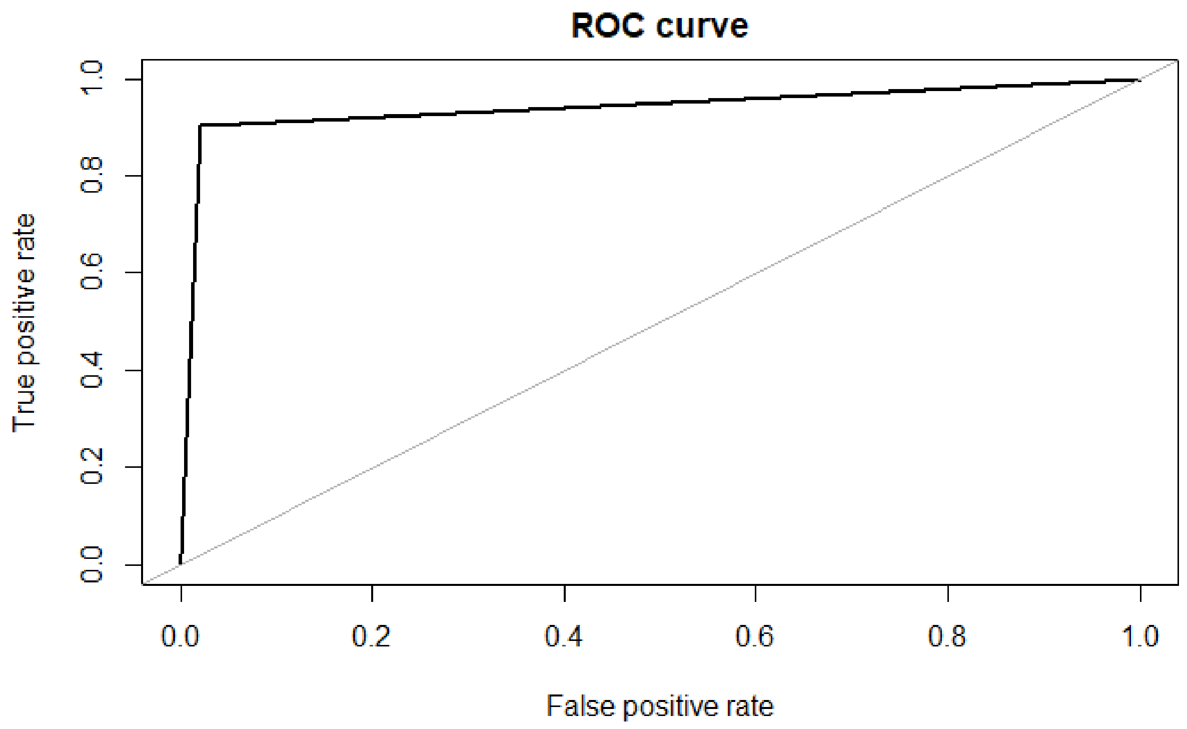


Synthetic Data Generation

Instead of removing or replicating observations, this method overcomes imbalances by generating artificial observations with boostrapping and k-nearest neighbors (kNN). We use the ROSE package in R to demonstrate. The resampled data has 113,175 non-fraudulent observations and 114,130 fraudulent observations. However, the AUC dropped to 0.939.

```
data.rose <- ROSE(Class ~ ., data = training, seed = 1)$data
table(data.rose$Class)
##
##     No    Yes
## 113715 114130
fit.rose <- rpart(Class ~ ., data = data.rose)
pred.rose <- predict(fit.rose, newdata = testing)
roc.curve(testing$Class, pred.rose[,2])
```

```
## Area under the curve (AUC): 0.944
```



It is observed that undersampling and oversampling combined has better performance in terms AUC, and thus we will train models using this data.

Classification Models

In addition to the CART (rpart) method, we will examine the performance from 4 other popular classification algorithms below.

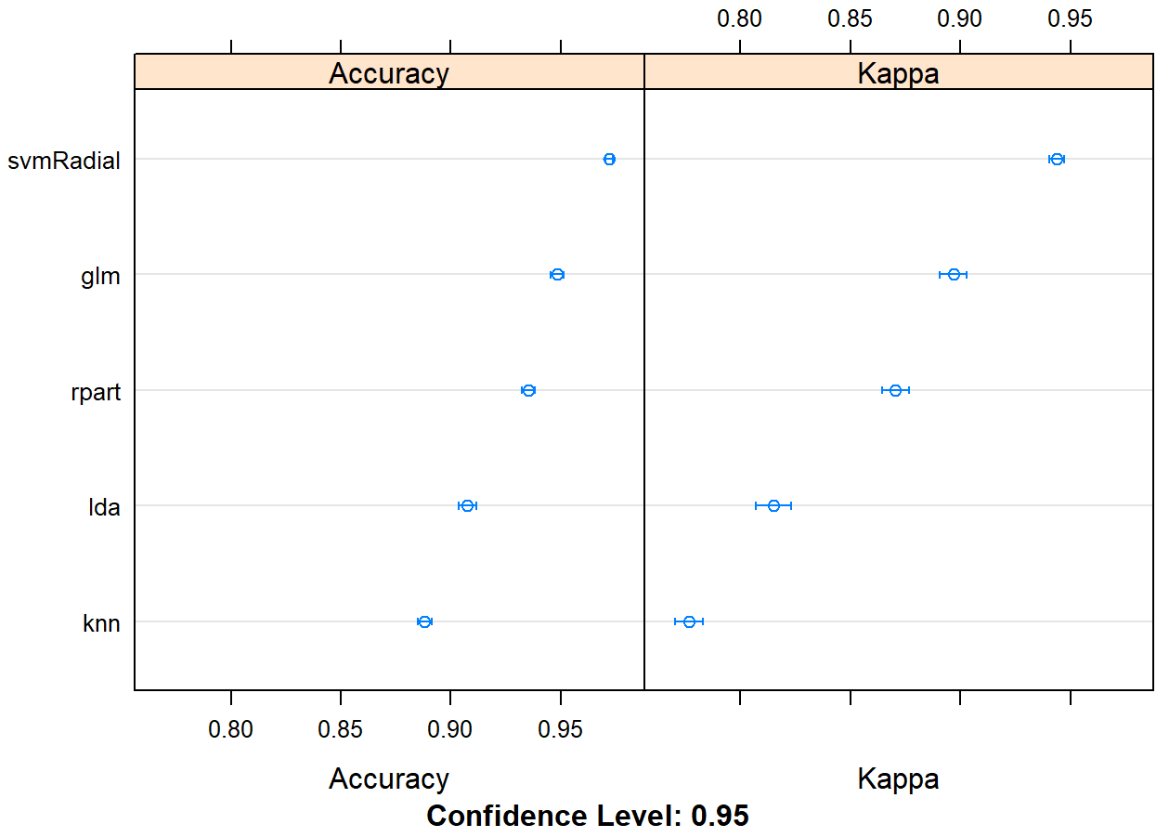
- Logistic Regression (GLM)
- Linear Discriminate Analysis (LDA)
- k-Nearest Neighbors (kNN)
- Support Vector Machine (SVM radial kernel)

```
control <- trainControl(method="repeatedcv", number=10, repeats=3, savePredic
tions=TRUE, classProbs=TRUE)

algorithmList <- c('rpart','glm','lda','knn','svmRadial')

set.seed(2)

models <- caretList(Class~., data=data_balanced_both, trControl=control, meth
odList=algorithmList)

results <- resamples(models)

summary(results)
```

```
##

## Call:
```

```
## summary.resamples(object = results)
##
## Models: rpart, glm, lda, knn, svmRadial
## Number of resamples: 30
##
## Accuracy
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## rpart     0.9199199 0.9295000 0.9340000 0.9352343 0.9387197 0.9570000    0
## glm       0.9330000 0.9432500 0.9499750 0.9484335 0.9537962 0.9649650    0
## lda       0.8780000 0.9045000 0.9079999 0.9076006 0.9150424 0.9280719    0
## knn       0.8690000 0.8842212 0.8885000 0.8881660 0.8950787 0.9010000    0
## svmRadial 0.9640000 0.9682662 0.9724865 0.9719333 0.9750000 0.9830170    0
##
## Kappa
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## rpart     0.8398211 0.8589407 0.8679921 0.8704344 0.8774299 0.9139828    0
## glm       0.8659249 0.8864279 0.8999028 0.8968087 0.9075405 0.9298901    0
## lda       0.7554767 0.8086653 0.8157295 0.8149068 0.8298354 0.8559718    0
## knn       0.7386377 0.7688682 0.7774740 0.7767816 0.7905457 0.8023715    0
## svmRadial 0.9279813 0.9365088 0.9449593 0.9438509 0.9499915 0.9660282    0
```

```r
dotplot(results)
```



It is observed that SVM has the best performance in terms of accuracy.

Model Ensembling

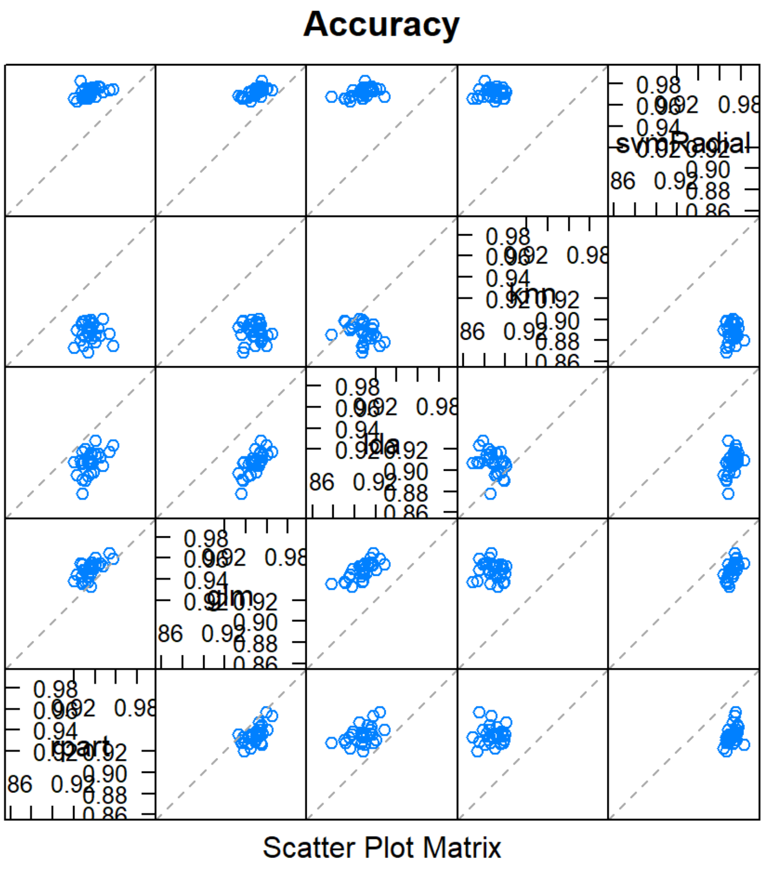
Common ensembling methods include bagging, boosting and stacking. Here we will use the stacking method via logistic regression with the caret package in R.

The idea of stacking is straightforward. First we will predict the class with each model, returning 5 sets of predictions from the models. We then treat each set of predictions as features to train a logistic regression model, giving us the coefficients of each model, which can be seen as a weighted representation of different classification sub-models in the stacked model.

In addition, when stacking models, it is important to see the correlations between each sub-model, as sub-models with low correlations could yield better results when stacked together, due to the fact that we can leverage different features from the sub-models. We do notice that logistic regression (glm) and CART (rpart) are highly correlated. Generally speaking, correlations lower than 0.75 are accepted, and thus we will still include the sub-models for stacking.

```
#Model Correlations
modelCor(results)
```

```
##                rpart        glm       lda         knn  svmRadial
## rpart     1.00000000  0.6313136 0.4476977  0.01976631 0.40666662
## glm       0.63131361  1.0000000 0.7305931 -0.11000193 0.70950541
## lda       0.44769767  0.7305931  1.0000000 -0.37265064 0.48765737
## knn       0.01976631 -0.1100019 -0.3726506  1.00000000 0.03402376
## svmRadial 0.40666662  0.7095054  0.4876574  0.03402376 1.00000000
splom(results)
```



```
#Model Stacking with GLM
stackControl <- trainControl(method="repeatedcv", number=10, repeats=3, saveP
redictions=TRUE, classProbs=TRUE)
set.seed(3)
stack.glm <- caretStack(models, method="glm", metric="Accuracy", trControl=st
ackControl)
print(stack.glm)
## A glm ensemble of 2 base models: rpart, glm, lda, knn, svmRadial
##
## Ensemble results:
## Generalized Linear Model
##
```

```
## 30000 samples
##     5 predictor
##     2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 26999, 27000, 27000, 27000, 27000, 27000, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9888666  0.9777295
```

The stacked model has an accuracy of 98.8%, which is slightly higher than the mean accuracy of the SVM sub-model at 97.2%. In addition, Kappa value of 0.975 also outperforms the 0.945 Kappa from SVM (usually Kappa value of 0.8-1 is considered excellent).