Fraud Detection in an Imbalanced Dataset

Step 3: Building a prediction model

Samiul Azam

October 30, 2017

Synopsis

In this analysis, I develop a fraud prediction model considering imbalances of data. Moreover, I discuss about what validation metric will be appropriate, how to handle imbalanced data and the experimental design I use in this analysis.

Let's load the data, select variables (based on previous step) and do normalization.

```
# Reading data
mydata <- read.csv(file="creditcard.csv", head=TRUE, sep=",")

# Make Fraud transactions as positive class ("No" is the positive class)
mydata$Class = ifelse(mydata$Class == 1,"No","Yes")

# Selecting 21 features based on previous analysis
model_predictors <- mydata[, c("V1","V2","V3","V4","V5","V6","V7","V8","V9","V10",
                      "V11","V12","V14","V16","V17","V18","V19","V20","V21",
                      "V27","Time")]

# Normalization (transform to zero mean and unit standard deviation)
proc_values <- preProcess(model_predictors, method = c("center", "scale"))
norm_predictors <- predict(proc_values, model_predictors)
```

Normalization is used to standardize the range of variables' data before applying any machine learning algorithm. In this analysis, I make all variables to have zero mean and unit standard deviation.

Let's discuss about the experimental design. Here, I consider 30% of data for model validation and 70% for model training, testing and tuning. During training, testing and tuning, I consider 4-fold cross-validation of that 70% data. After model build-up, I validate it using the validation data.

```
# Slicing data into training and Validation set
set.seed(1) # For reproducibility
# Generate training data indices
ind_train <- createDataPartition(y = mydata$Class, p = 0.70, list = FALSE)

# Splitting training (70%) and validation (30%) data
X_training <- norm_predictors[ind_train,]
X_validation <- norm_predictors[-ind_train,]

# Splitting the responses (or class labels)
Y_training = mydata[ind_train, "Class"]
Y_validation = mydata[-ind_train, "Class"]

# Check the size of the testing and validation data
dim(X_training)
```

```
## [1] 199366     21
```

```r
dim(X_validation)
```

```
## [1] 85441    21
```

```r
# Number of positive classes (Fraud cases) in training and validation data
sum(Y_training == "No")
```

```
## [1] 345
```

```r
sum(Y_validation == "No")
```

```
## [1] 147
```

Here, we see that the training data contains only 345 Fraud transactions ("No" Class). The rest of them is non-Fraud transactions ("Yes" Class). Usually, a two-class classification problem use "Accuracy" as metric. However, in our case one class contains more than 99% of the data. A silly classifier (which always predict every transactions as non-Fraud) will have atleast 99% accuracy. So, accuracy is not a good choice for highly imbalanced data. Thus, in credit card fraud detection, I choose both precision (positive predictive value) and recall (true positive rate) as metrics for evaluation. For a perfect model they will be 1.0. To maximize precision and recall, I need to minimize both FP and FN errors.

Both precision and recall are also a good metric for parameter selection. But these two are measured based on a cut-off point or threshold. So, a better metric for parameter selection is the area-under-the-precision-recall curve (AUPRC), which is a threshold invariant metric.

Let's try to build a single classifier model (Fraud detection model) using the imbalanced training data. Initially, I consider recursive partioning tree (RPT) as a model, as it is faster and has only one parameter (tree complexity) to tune. It is better than KNN (K nearest neighbor) and SVM (support vector machine) in terms of time (as testing in KNN and training in SVM is time consuming for large number of observations).

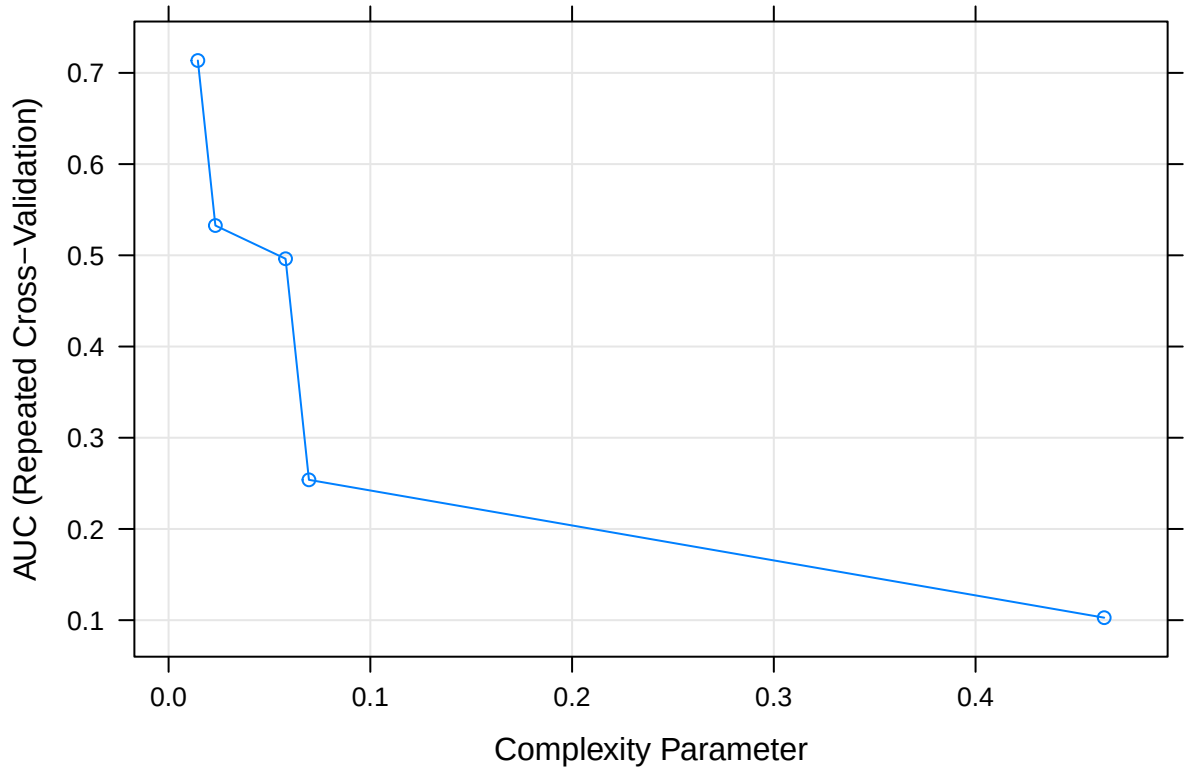
To tune RPT, I consider AUPRC as a metric. Select a parameter when AUC of PR curve is maximum.

```r
# Using AUPRC as metric to model buildup
set.seed(1)
# Training setup:
      # 4 fold cross validation, repeats for 1 time, generate class probabilities to
      # draw the PR (precision-recall) curve, summary function is PR summary
ctrl <- trainControl(method="repeatedcv", number = 4, repeats = 1, classProbs = TRUE,
                     summaryFunction = prSummary, verboseIter = FALSE)

# Train and tune the RPT model. Here, the metric for selecting tuning parameter
#(out of five values) is AUC (Area under the PR curve).
modelFit_AUPRC <- train(X_training, Y_training, method = "rpart",
                  trControl = ctrl, metric = "AUC", tuneLength = 5)

# Contains the trained model and training results
modelFit_AUPRC
```

```
## CART
##
## 199366 samples
##     21 predictor
##      2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold, repeated 1 times)
## Summary of sample sizes: 149525, 149524, 149525, 149524
## Resampling results across tuning parameters:
```

```
##
##   cp          AUC        Precision  Recall     F
##   0.01449275  0.7135644  0.8906616  0.6985432  0.7802719
##   0.02318841  0.5326711  0.8354573  0.7159516  0.7698927
##   0.05797101  0.4962286  0.7904944  0.7073643  0.7439072
##   0.06956522  0.2538486  0.7916083  0.6753876  0.7277148
##   0.46376812  0.1027260  0.7613386  0.4837944  0.6940265
##
## AUC was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.01449275.
```

```
plot(modelFit_AUPRC)
```



From the above, we can see how the training and tuning has been conducted.

Now, validate the RPT model with the validation data.

```
# Function to predict and generate validation results
validation_result <- function(modelFit_AUPRC, X_validation, Y_validation)
{
      # Predict Fraud or non-Frad using the trained model.
      result_AUPRC <- predict(modelFit_AUPRC, X_validation)
      mat <- confusionMatrix(result_AUPRC, Y_validation)

      TP = mat$table[1] # Number of true positiveS
      FN = mat$table[2] # Number of false negativeS
      FP = mat$table[3] # Number of false positiveS
```

```
      # Calculate precision and recall
      precision <- TP/(TP+FP)
      recall <- TP/(TP+FN)

      message("Precision: ", precision)
      message("Recall   : ", recall)

      # See the confusion matrix
      message("Confusion Matrix  : No is the positive class (Fraud)")
      mat$table
}

# Calling validation function
validation_result(modelFit_AUPRC, X_validation, Y_validation)
```

```
## Precision: 0.834586466165414
```

```
## Recall   : 0.755102040816326
```

```
## Confusion Matrix  : No is the positive class (Fraud)
```

```
##           Reference
## Prediction    No   Yes
##        No    111    22
##        Yes    36 85272
```

Here, the model has good precision (Fraud predictive value): 83%. Only 0.025% genuine transactions (22 out of 85294 observations) are mistakenly predicted as Fraud. The recall (True positive rate) has acceptable value (76%) even though the data is highly imbalanced. However, 36 out of 147 Frauds transactions are not detected. One way to improve the performance can be adding more observations for the positive class (Fraud transactions). But this is not possible at this moment.

Let's try to improve the performance of RPT model by balancing (both classes have almost equal number of observations) the training data. Some of the popular ways to balance an imbalance data are as follows: 1) Up-sampling (Increasing samples of minority class by repeat) 2) Down-sampling (Decreasing samples of majority class by remove) 3) SMOTE (Synthetic Minority Over-sampling Technique) However, improvement in performance is not guaranteed.

The following code segments are showing the validation results for up-sampling, down-sampling and SMOTE.

```
set.seed(1)
# Downsampling of major class
ctrl$sampling = "down"
# Training RPT with downsampling
modelFit_AUPRC <- train(X_training, Y_training, method = "rpart",
                  trControl = ctrl, metric = "AUC", tuneLength = 5)

# Calling validation function
validation_result(modelFit_AUPRC, X_validation, Y_validation)
```

```
## Precision: 0.0500376222723853
```

```
## Recall   : 0.904761904761905
```

```
## Confusion Matrix  : No is the positive class (Fraud)
```

```
##           Reference
## Prediction    No   Yes
##        No    133  2525
```

```
##        Yes    14 82769
```

Result summary (Down-sampling): Recall (True positive rate) is 90% (which is excellent). However, precision is very low. Unfortunately 2.9% of the genuine transactions are misclassified as Fraud.

```
set.seed(1)
# Upsampling of minor class
ctrl$sampling = "up"
# Training RPT with upsampling
modelFit_AUPRC <- train(X_training, Y_training, method = "rpart",
                  trControl = ctrl, metric = "AUC", tuneLength = 5)

# Calling validation function
validation_result(modelFit_AUPRC, X_validation, Y_validation)
```

```
## Precision: 0.0512129380053908
```

```
## Recall   : 0.904761904761905
```

```
## Confusion Matrix  : No is the positive class (Fraud)
```

```
##           Reference
## Prediction    No   Yes
##        No    133  2464
##        Yes    14 82830
```

Result summary (Up-sampling): Recall (True positive rate) is 90% (which is excellent). However, again precision is very low. Unfortunately, 2.9% of genuine transactions are misclassified as Fraud.

```
set.seed(1)
# Apply SMOTE
ctrl$sampling = "smote"
# Training RPT with SMOTE
# It will add synthesized data for minor class and remove data from major class
modelFit_AUPRC <- train(X_training, Y_training, method = "rpart",
                  trControl = ctrl, metric = "AUC", tuneLength = 5)
```

```
## Loading required package: grid
```

```
# Calling validation function
validation_result(modelFit_AUPRC, X_validation, Y_validation)
```

```
## Precision: 0.0629575402635432
```

```
## Recall   : 0.877551020408163
```

```
## Confusion Matrix  : No is the positive class (Fraud)
```

```
##           Reference
## Prediction    No   Yes
##        No    129  1920
##        Yes    18 83374
```

Result summary (SMOTE): Recall (True positive rate) is 88%. However, again precision is very low. Unfortunately, 1% of genuine transactions are misclassified as Fraud.

Overall, above three sampling methods are unable to improve both precision and recall at the same time.

Another way to handle imbalanced data is to use any penalized machine learning algorithm, such as penalized-SVM or penalized-LDA. These algorithms assign high cost when misclassify a minority class during training. It will force the model to give attention to minority class. Let's try with the penalized-LDA and see the result.

```r
set.seed(1)
# No sampling technique is applied as their performance was not good with RPT
ctrl$sampling = NULL
# Training penalized-LDA (Linear Discriminant Analysis)
modelFit_AUPRC <- train(X_training, Y_training, method = "pda",
                  trControl = ctrl, metric = "AUC", tuneLength = 5)
```

```
## Loading required package: class
```

```
## Loaded mda 0.4-9
```

```r
# Calling validation function
validation_result(modelFit_AUPRC, X_validation, Y_validation)
```

```
## Precision: 0.8671875
```

```
## Recall    : 0.755102040816326
```

```
## Confusion Matrix  : No is the positive class (Fraud)
```

```
##            Reference
## Prediction    No   Yes
##        No    111    17
##        Yes    36 85277
```

Penalized-LDA is better than RPT (recursive partition tree) model. Penalized-LDA gives 87% of precision whereas RPT shows 83%. Both of them have 76% recall.

Due to the imbalances of data, a single classifier performs weekly. So, the concept of Adaboost (ensemble of week classifiers) may improve the result, as boosting algorithm will strategically assign more weights to the misclassified data at each iteration. I have found boosted logistic regression (LogitBoost) works better with our data. The codes are as follows.

```r
set.seed(1)
# Training boosted logistic regression.
modelFit_AUPRC <- train(X_training, Y_training, method = "LogitBoost",
                  trControl = ctrl, metric = "AUC", tuneLength = 5)

# Calling validation function
validation_result(modelFit_AUPRC, X_validation, Y_validation)
```

```
## Precision: 0.857142857142857
```

```
## Recall    : 0.775510204081633
```

```
## Confusion Matrix  : No is the positive class (Fraud)
```

```
##            Reference
## Prediction    No   Yes
##        No    114    19
##        Yes    33 85275
```

From the above result, we see that LogitBoost works slightly better (precision is 86%, recall is 78%) than the penalized-LDA (precision is 87%, recall is 76%) and RPT (precision is 83%, recall is 76%).

Bootstrap aggregating or bagging can be applied to handle imbalanced data. In bagging, a set of classifiers are trained using a set of new smaller size training data generated using bootstrapping (observations are selected uniformly) from the original data. Bootstrap generates M smaller training set to train M classifiers. Thus, reduce the impact of imbalanced data on the model learning. As they are trained on a smaller size training set, they are expected to be week in nature. Finally, aggregation (majority voting) of M predictions is applied to achieve better prediction result.

Lets try bagging with tree for the credit card fraud database.

```
# Training of Tree based bagging.
set.seed(1)
ctrl <- trainControl(method="none", verboseIter = FALSE)
modelFit <- train(X_training, Y_training, method = "treebag", trControl = ctrl)

# Calling validation function
validation_result(modelFit, X_validation, Y_validation)
```

```
## Precision: 0.944444444444444
```

```
## Recall   : 0.80952380952381
```

```
## Confusion Matrix  : No is the positive class (Fraud)
```

```
##            Reference
## Prediction    No   Yes
##        No    119     7
##        Yes    28 85287
```

Bagging provides good improvement in both precision (94%) and recall (81%). Here, only 7 cases out of 85294 (0.008%) are incorrectly detected as fraud transactions. Among all above, bagging algorithm provides the best solution for credit card fraud detection.

In summary, following strategy has been applied to find a well performed predictor in the case of imbalanced dataset:

```
1) Select appropriate metric for model evaluation. Accuracy is not
   a good choice. Whole confusion matrix is a good way to understand
   the model performance in the case of imbalance data.
   Need to maximize both precision and recall.

2) Applied simple recursive partioning tree (RPT) to see how the data
   performs with a simple machine learning algorithm.
   (Precision: 83%, Recall: 76%)

3) Tried conventional data resampling techniques, such as up-sampling,
   down-sampling and SMOTE to make balanced training data. Didn't get
   any improvement.

4) Applied a penalized model (penalized discriminant analysis).
   (Precision: 87%, Recall: 76%)

5) Applied logistic regression with Boosting.
   (Precision: 86%, Recall: 78%)

6) Applied Tree bagging to handle imbalance data.
   (Precision: 94%, Recall: 81%)
```

The database provided for this task is highly skewed which makes it challenging to improve both precision and recall at the same time. Small improvement in recall has big impact on precision (false positives increase drastically). However, in practice, we can select a suitable cut-off (threshold) based on the system's area-under-the-precision-recall curve (AUPRC). Suitability means, we can increase recall (high detection rate of fraud transactions), but it will cost precision (will increase number of non-Fraud cases to be detected as Fraud).

Following actions may improve the performance:

1) Tune the parameters in larger space.

2) Try other bagging and boosting algorithms.

3) Apply powerful algorithms such as deep-learning.

4) Think the problem from different point of view: anomaly or change detection.

5) Gather more observations of minor class (Fraud observations).

6) Remove outliers.