Dhrumel Shah

12/27/16

DATA MINING PROJECT

**SANTANDER CUSTOMER SATISFACTION**

1. Overview

This project involves analyzing a dataset from Santander Bank to identify dissatisfied customers, with the goal of practicing and researching on data mining and machine learning techniques using R. The dataset is available on the Kaggle website, containing 370 anonymous features for each customer. The training set has 76,020 instances and the test set has 75,818 instances. Submissions can be made on the website, and the performance on the test set is determined by area under the ROC curve between the predicted probability and the observed target.

1. Summary of Techniques
2. Handling noise (convert noise to attribute mode)
3. Feature construction (binary and sumOfZeros features)
4. Normalization (min-max)
5. Feature selection (maximal correlation)
6. Outlier removal (clarans and abod)
7. Training model (ensemble of xgbTree and reg. random forest)
8. Validating model (2x10-fold repeated cv) and performing predictions
9. Handling Noise

Noise refers to erroneous data most likely present due to faulty data collection. They are identified as values that show random and extreme deviation from the rest of the data. Each attribute was inspected for noise by plotting histograms in R.

The var3 attribute for example had a large number of instances with the value **-999,999**. The remaining instances had values in the range (0,238). Clearly this is an extreme deviation with no logical explanation, as it is unlikely that so many values will exactly equal that number. The value was replaced with the mode of the attribute, i.e. 0.

The remaining attributes were also inspected by observing their distribution. Most of them had some values exactly equal to **9,999,999,999**. For the same reasons as explained above, this value was **replaced with the attribute mode**.

1. Feature Construction

21 additional features were constructed in order to add more meaning to our dataset.

**Binary features** were created for all non-binary features, mainly because in a bank dataset, negative and positive values can hold very different meaning. For example, an account balance of 100 is a lot similar in meaning to 1000 than it is to -100, even though it is linearly closer to -100.

A **‘Sum of Zeros’ feature** was also created as the data is heavily sparse, and it allows us to differentiate between instances that have many 0s and those that have other values.

1. Normalization

**Min-max normalization** was applied in order to standardize all features, giving them a range of (0,1) and maintaining the same distribution.

Unused technique- Logarithmic transformation:

The distribution of data in most of the attributes was heavily skewed. Applying a logarithmic transformation was identified as a possible solution to improve the model performance, but it drastically reduced it instead and hence was not used. On carrying out further research, I understood that log transforms should be very carefully used as changing the distribution can often lead to changing the meaning of those values.

1. Feature Selection

Feature selection was done in order to get rid of redundant features that are usually either very similar to other features, or do not contribute much to the overall variance in the data. It helps reduce overfitting of the model to the training set.

I first eliminated 44 features that had **0 variance**, and then used **Maximal Correlation** to eliminate 68 more, resulting in a reduced dataset with 279 features. This technique captures non-linear relationships between pairs of features by calculating the maximal correlation coefficient. I used the ‘ace()’ function from the ‘acepack’ library in R to implement it. [1][2]

I first identified all the pairs that had a maximal correlation of greater than 0.999. I tried the following values: 0.9, 0.99, 0.999, 0.9999 and 1. 0.999 gave the best ROC Area on 2 x 10-fold repeated CV. Then for each feature in every pair, I found the maximal correlation with the class label and eliminated the feature that showed lower correlation.

Unused techniques:

PCA: Principal Component Analysis is a popular technique for dimensionality reduction. It finds a projection that captures the largest amount of variance in the data. The input data is first normalized and then k orthogonal vectors or principal components are computed from the n features, where k <=n. Each vector k is a linear combination of the n variables with different proportions of each variable contributing to the vector. The principal components are uncorrelated as their directions are perpendicular to each other. They are sorted in decreasing order of variance and the cumulative variance proportion is plotted on a graph. Since the slope is close to 0 at 250, the top 250 components are chosen as the features. While PCA showed an improvement in 2x10-fold repeated CV, it was outperformed by Maximal Correlation and hence not used. [3]

NLPCA: The Non-Linear PCA technique produces non-linear transformations of the features. It can be performed using the homals() function in R and extracting the eigenvalues from the result. While NLPCA usually performs a better feature reduction than PCA, it is very slow in computation and is optimized mainly for discrete features. Since this dataset has a large number of numeric attributes, this method was not used. [4]

1. Outlier Removal

The performance of a predictive model can be improved by making it more robust to outliers. Using CLARANS and ABOD, 13,603 or 18% of the instances were identified as outliers and removed.

I first clustered the data using **CLARANS** (Clustering Large Applications based on Randomized Search). This method uses PAM (Partitioning Around Medoids) on a given number of samples. PAM is performed on all the data, and hence is inefficient and unsuitable for large datasets. Based on previous research, CLARANS has proven to be a lot faster and almost as accurate. [5]

This clustering technique was implemented using the clara() function from the ‘cluster’ package in R. First, a sample is drawn and k-medoids are computed from it. The remaining instances are then assigned to the closest medoid or cluster. The average dissimilarity is calculated and repeated for the next sample. The final k values are chosen from the sample with the lowest average dissimilarity, and then each instance in the entire dataset is assigned to nearest k value.

10 samples were collected, each of size 1000, and the Silhouette Width (measure of intra-cluster cohesion and inter-cluster separation) was calculated for the following k values:

|  |  |
| --- | --- |
| **k** | **Silhouette Width** |
| 10 | 0.3265 |
| 15 | 0.3565 |
| 20 | 0.3888 |
| 25 | 0.3926 |
| 30 | 0.3647 |
| 35 | 0.4107 |
| 40 | 0.3651 |
| 50 | 0.2970 |

35 was chosen as the number of clusters as it resulted in the highest Silhouette Width.

1. The first set of outliers were removed by **eliminating small clusters**. Those clusters containing a small number of observations compared to the entire dataset were removed. I tried the values 0, 0.2%, 0.5%, 1% and 2% on the trained model, and based on the ROC Area, I eliminated clusters containing less than 1% of total observations. Only 23 of the 35 clusters and 70,025 of the 76,020 instances remained from the original dataset.
2. Next, I eliminated clusters with small ABOD factors. The angle-based outlier factor of A is the variance over the angles between the difference vectors of A to all pairs of points weighted by the distance of the points. Due to the “curse of dimensionality”, this Angle-Based Outlier Detection technique is far more effective than common distance techniques for high dimensional data. [6]

The abod() function from the ‘abodOutlier’ package in R is used to calculate the abod factors for each point or cluster medoid. The input to the function was basically the 23 cluster medoids. For each medoid:

* Calculate the angle it forms with each pair of the 22 remaining points
* Calculate the average of those angles
* Find the cosine of that average, which gives you the abod factor

With this technique, you don’t want to eliminate too many clusters as it can result in loss of important information. I tried eliminating clusters with abod factors < mean – (0, 1, 2, 3) \* SD. 2 gave the best results, and hence, clusters with abod factors 2 SDs smaller than the mean were removed. Only 1 of the 23 clusters had that small an abod factor, and thus, 22 clusters remained containing 64,505 of the 70,025 instances.

1. Finally, I eliminated observations within clusters that were far from the cluster center. I found the Euclidean distance between each point in a cluster to the medoid, and removed those points that had a distance of greater than 2 SDs from the mean. Having tried 1, 2 and 3 SDs, 2 gave the best results. 62,417 instances of the 64,505 remained.
2. Training Model

A linear ensemble of R’s XGBTree and Regularized Random Forest was used for the final model. This stacked ensemble outperformed other classifiers such as R’s AdaBoost and H2O’s Feedforward Backpropagation Neural Network. The model was built from the pre-processed training set containing 62,417 instances and 279 features.

Regularized Random Forest is basically an ensemble of many weak Decision Trees. Each tree is built on a random sample of instances and a random subset of features. Using this technique, although each individual tree performs just slightly better than random guessing, each one models a different aspect of the dataset and thus an ensemble of all these trees is very powerful. A big advantage of using Random Forests is that it helps reduce overfitting. To further enhance its performance, Regularized Random Forest is used with L1 regularization.

This classification algorithm is implemented using the “RRF” package in R. The parameters chosen are as follows:

* Maxnodes = 1000 (Maximum number of terminal nodes trees in the forest can have)
* Mtry = 20 (Number of variables randomly sampled as candidates at each split)
* coefReg = 0.5 (L1 regularization coefficient)
* coefImp = 0.8 (Importance coefficient)

XGBTree is an Extreme Gradient Boosting Algorithm using decision trees. Boosting is an ensemble technique where weights are assigned to each training instance, and these weights are increased for each misclassified record so that they have a greater chance of being sampled for the next classifier. In gradient boosting, a gradient descent algorithm is used to minimize the loss function when adding new models. Extreme Gradient Boosting is a high performance algorithm engineered for maximum utilization of CPU and memory resources.

This algorithm is implemented using the “xgboost” package in R. The parameters chosen are as follows:

* eta = 0.02 (shrinking of feature weights at each step to prevent overfitting)
* max\_depth = 5 (max depth of tree)
* nrounds = 560 (max number of iterations)
* gamma = 1 (minimum loss reduction required to make a further partition on a leaf node, larger the value more conservative the algorithm to reduce overfitting)
* min\_child\_weight = 5 (minimum sum of instance weights needed in a child)
* subsample = 0.7 (proportion of training instances sampled)
* colsample\_bytree = 0.8 (subsample ratio of columns)

1. Validating Model and Predicting Results

2 x 10-fold repeated Cross Validation was used to validate the model. This technique partitions the training set into 10 mutually exclusive subsets. At each iteration, one subset is used as the test set and the remaining subsets are used as the training set. At the end of 10 iterations, after each subset has been used exactly once as a test set, the entire process is repeated for a second time. The ROC Area is calculated for each iteration and the average is displayed. This method of validation, although computationally expensive, yields close to unbiased results.

Once the ensemble model is trained, it is used to predict the probability of a customer being dissatisfied. The final ROC Area was 0.857 on the training set and 0.824 on the test set, finishing in the top 2% of participants.

1. Code (A .R file is also attached containing all the code)

# santander customer satisfaction

library(mlbench)

library(caret)

library(acepack)

library(homals)

library(abodOutlier)

library(cluster)

library(fpc)

library(clValid)

library(xgboost)

library(plyr)

library(fastAdaboost)

library(caretEnsemble)

library(randomForest)

library(pROC)

library(RRF)

main <- function()

{

trainData <- read.csv(file="C:/Users/Dhrumel/Desktop/custSat\_train.csv",header=TRUE) #(76020 \* 370)

TARGET <- trainData$TARGET

trainData$TARGET <- NULL

trainData <- clean(trainData)

trainData$TARGET <- TARGET

trainData <- cfs(trainData) # 279 remain

#trainData <- pca(trainData) # 250 remain

#trainData <- nlpca(trainData)

trainData <- clarans\_abod(trainData)

testData <- read.csv(file="C:/Users/Dhrumel/Desktop/custSat\_test.csv",header=TRUE) #(75818 \* 370)

id <- testData$Id

testData <- clean(testData)

testData <- testData[colnames(testData) %in% colnames(trainData)] #ensure same columns in train and test

TARGET <- trainData$TARGET

trainData <- trainData[colnames(trainData) %in% colnames(testData)]

trainData <-cbind(trainData,TARGET)

results <- train\_test(trainData,testData)

results$ID <- id

write.csv(results,file="C:/Users/Dhrumel/Desktop/sanResult\_e1.csv")

}

train\_test <- function(training,testing)

{

training$TARGET[training$TARGET==1] <- "dis"

training$TARGET[training$TARGET==0] <- "sat"

training$TARGET <- as.factor(training$TARGET)

set.seed(12579)

control1 <- trainControl(method="cv",number=2,savePredictions="final",classProbs=TRUE, summaryFunction = twoClassSummary)

modelList <- caretList(TARGET~.,data=training,trControl=control1,tuneList=list(xgb1=caretModelSpec(method="xgbTree",

tuneGrid=expand.grid(eta=0.02,max\_depth=5,nrounds=560,gamma=1,min\_child\_weight=5,subsample=0.7,colsample\_bytree=0.8)),

rf1=caretModelSpec(method="RRF",maxnodes=1000,tuneGrid=expand.grid(mtry=20,coefReg=0.5,coefImp=0.8))))

control2 <- trainControl(method="cv",number=2,savePredictions="final",classProbs=TRUE, summaryFunction = twoClassSummary)

ensemble <- caretEnsemble(modelList,metric="ROC",trControl=control2)

summary(ensemble)

p <- data.frame(predict(ensemble,newdata=testing,type="prob"))

}

clarans\_abod <- function(x) # remove outliers using clarans and abod ()

{

print(nrow(x))

k<-35

results <- clara(x,k,samples=10,sampsize = 1000)

x$cluster <- results$clustering

cluster <- data.frame(index=1:k,size=results$clusinfo[,"size"], med=results$i.med)

x$index = 1:nrow(x)

print(cluster)

#remove clusters with less than 1% of observations (23 of 35 remain)

outlier <- cluster[cluster[,2]<(0.01\*nrow(x)),"index"]

cluster <- cluster[cluster[,2]>=(0.01\*nrow(x)),]

x <- x[!(x$cluster %in% outlier),]

print(nrow(x))

print(cluster)

#eliminate clusters with small abod factors (22 of 23 remain)

cluster$abodFactor <- abod(x[x$index %in% cluster$med,(1:(ncol(x)-2))])

outlier<-cluster[cluster$abodFactor<((mean(cluster$abodFactor))-2\*sd(cluster$abodFactor)),"index"]

x <- x[!(x$cluster %in% outlier),]

cluster <- cluster[cluster$abodFactor>=((mean(cluster$abodFactor))-2\*sd(cluster$abodFactor)),]

print(nrow(x))

print(cluster)

#eliminate observations within clusters far from the median

for(i in 1:nrow(cluster))

{

dist <- numeric()

outlier <- integer()

clust <- x[x["cluster"]==cluster[i,"index"],]

median <- clust[clust$index==cluster[i,"med"],1:(ncol(clust)-1)]

for(j in 1:nrow(clust))

dist = c(dist,sqrt(sum((clust[j,1:(ncol(clust)-1)]-median)^2)))

for(j in 1:nrow(clust))

{

if(dist[j]>((mean(dist))+2\*sd(dist))&&dist[j]!=0)

outlier=c(outlier,clust[j,"index"])

}

cluster[i,"size"]<-cluster[i,"size"]-length(outlier)

x <- x[!(x$index %in% outlier),]

}

x <- x[,-c(ncol(x)-1,ncol(x))]

print(nrow(x))

x

}

nlpca <- function(x) # not used

{

x <- homals(x,ndim=2,level="numerical")

x <- data.frame(x$eigenvalues)

}

pca <- function(data) # not used

{

TARGET <- data$TARGET

data$TARGET <- NULL

prin\_comp <- prcomp(data)

variance <- (prin\_comp$sdev)^2

prop\_var <- variance/sum(variance)

plot(cumsum(prop\_var))

data <- data.frame(prin\_comp$x)

data <- data[,1:250]

data <- cbind(data,TARGET)

testData<-as.data.frame(predict(prin\_comp, newdata=testData))

testData<-testData[1:250]

}

#remove highly correlated attributes using maximal correlation to capture non-linear associations

#Betweeen the 2 correlated attributes, the one with smaller correlation with TARGET is removed

cfs<-function(data)

{

n <- ncol(data)

i<-1

while(i < (n-1))

{

j<-i+1

while(j < n)

{

argmax =ace(data[,i],data[,j])

c <- cor(argmax$tx, argmax$ty)

if(c>0.999)

{

argmax =ace(data[,i],data[,"TARGET"])

corTarget1 <- cor(argmax$tx, argmax$ty)

argmax =ace(data[,j],data[,"TARGET"])

corTarget2 <- cor(argmax$tx, argmax$ty)

n<-n-1

if(corTarget1 <= corTarget2)

{

data[,i]<-NULL

i<-i-1

break

}

else

{

data[,j] <-NULL

j<-j-1

}

}

j<-j+1

}

i<-i+1

}

print(ncol(data))

data

}

clean<-function(data)

{

data$Id<-NULL #remove id attribute

data$var3[data$var3 < 0] <-0 #convert -99999 from var3 to 0

#convert noise values > 999,999,999 to mode

for(j in (1:ncol(data)))

{

ux <- unique(data[,j])

data[(data[,j]>999999999),j] <- ux[which.max(tabulate(match(data[,j],ux)))]

}

#create new attributes for those columns with min < 0 and max > 1

new<-data[,(lapply(data,max)>1 & lapply(data,min)<0)]

new[new<0] <- 0

new[new>0] <- 1

data <- cbind(data,new)

#add min to attributes with min < 0 and max > 1 to allow log transformation

#data[,(lapply(data,max)>1 & lapply(data,min)<0)]<-as.data.frame(lapply(data[,(lapply(data,max)>1 & lapply(data,min)<0)],addmin))

#add 1 and log attributes with max > 1

#data[,(lapply(data,max)>1)] <- data[,(lapply(data,max)>1)] + 1

#data[,(lapply(data,max)>1)] <- as.data.frame(lapply(data[,(lapply(data,max)>1)],log10))

data<-data[,lapply(data,var)!=0] #remove attributes with variance 0

soz<-integer()

for(i in 1: nrow(data)) #add sum of zeros attribute

soz[i] <- sum(data[i,]==0)

data <- cbind(data,soz)

print(ncol(data))

data<-as.data.frame(lapply(data,normalize)) #min-max normalization

}

normalize <- function(x)

{

x <- ((x-min(x))/(max(x)-min(x)))

}

addmin <- function(x)

{

x <- x - min(x)

}

1. References

\* The sources include but are not restricted to academic journals. However, a thorough background check was carried out on each source for credibility. Also, since this is a self-assigned project that did not have to be submitted, I only posted webpage links for citing the sources, and did not use any standard APA or MLA format.

[1] <https://www.quora.com/Correlation-can-measure-only-the-linear-relationship-between-variables-What-are-the-methods-for-measuring-non-linear-relationships-between-two-variables>

[2] <http://www.stat.cmu.edu/~ryantibs/datamining/lectures/11-cor2-marked.pdf>

[3] <https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/>

[4] <https://pdfs.semanticscholar.org/feaa/d142ca1f7977991064ae737f04409cae691f.pdf>

[5] <http://www.gjaet.com/wp-content/uploads/2014/03/AN-EFFECTIVE-ALGORITHM-FOR-OUTLIER-DETECTION.pdf>

[6] <http://www.dbs.ifi.lmu.de/Publikationen/Papers/KDD2008.pdf>