home_credit_modeling

2024-11-06

```r
# use the prcomp() function to calculate pca
pca <- prcomp( application_train, scale = TRUE )

# calculate cumulative variance explained
pca.var <- pca$sdev^2
pca.var.per <- pca.var / sum( pca.var )
cumulative_variance <- cumsum( pca.var.per )

# determine the number of PCs to retain that explain at least 80% of variance
num_components <- which( cumulative_variance >= 0.80 )[ 1 ]

# print summary
cat( "Number of PCs retained:", num_components, "\n" )
```

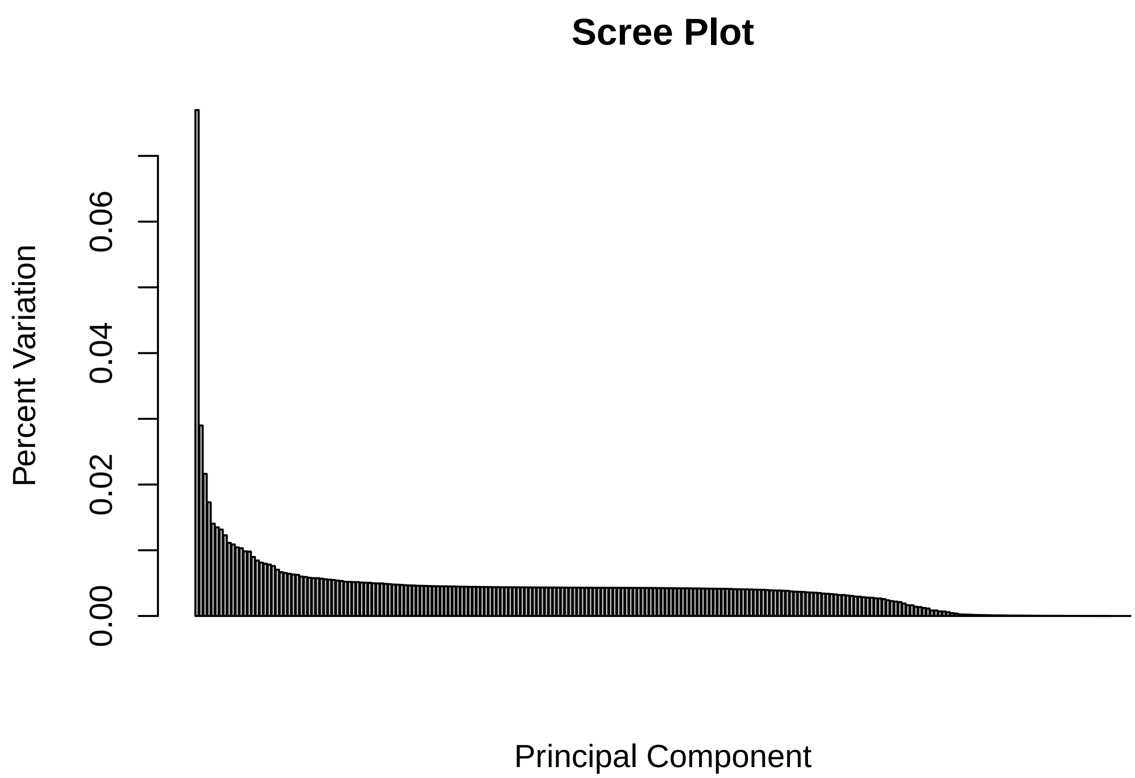
PCA

```
## Number of PCs retained: 125
```

```r
cat( "Cumulative variance explained:", cumulative_variance[num_components] * 100, "%\n" )
```

```
## Cumulative variance explained: 80.3812 %
```

```r
#plot the results
barplot( pca.var.per,
         main = "Scree Plot",
         xlab = "Principal Component",
         ylab = "Percent Variation" )
```




```r
# create PCA data frame using row numbers as sample IDs
pca.data <- data.frame( sample = 1:nrow( pca$x ),
                        X = pca$x[ ,1 ],
                        Y = pca$x[ ,2 ] )

# get the top 80% of the variance
loading_scores <- pca$rotation[ ,1 ]
scores <- abs( loading_scores )
score_ranked <- sort( scores, decreasing = TRUE )
top_pca <- names( score_ranked[ 1:125 ] )

# show the top variables and their contributions
data.frame( variable = top_pca,
            loading = pca$rotation[ top_pca,1 ],
            row.names = NULL )
```

```
##                                              variable     loading
## 1                                     LIVINGAREA_MEDI 0.218446065
## 2                                      LIVINGAREA_AVG 0.218443215
## 3                                      APARTMENTS_AVG 0.217410482
## 4                                     APARTMENTS_MEDI 0.217390880
## 5                                     LIVINGAREA_MODE 0.215160719
## 6                                     APARTMENTS_MODE 0.213961753
## 7                                      TOTALAREA_MODE 0.213817444
## 8                                       ELEVATORS_AVG 0.203990769
## 9                                      ELEVATORS_MEDI 0.203965528
```

```
## 10                                         ELEVATORS_MODE  0.202169290
## 11                                  LIVINGAPARTMENTS_MEDI  0.192108500
## 12                                   LIVINGAPARTMENTS_AVG  0.191421555
## 13                                  LIVINGAPARTMENTS_MODE  0.191295880
## 14                                      BASEMENTAREA_MEDI  0.166846170
## 15                                       BASEMENTAREA_AVG  0.166789722
## 16                                          FLOORSMAX_AVG  0.164002131
## 17                                      BASEMENTAREA_MODE  0.163824768
## 18                                         FLOORSMAX_MEDI  0.163734792
## 19                                         FLOORSMAX_MODE  0.163722926
## 20                                          ENTRANCES_AVG  0.142688920
## 21                                         ENTRANCES_MEDI  0.142128630
## 22                                         ENTRANCES_MODE  0.137758393
## 23                                        COMMONAREA_MEDI  0.129785373
## 24                                         COMMONAREA_AVG  0.129488132
## 25                                        COMMONAREA_MODE  0.128325059
## 26                                          LANDAREA_MEDI  0.123117397
## 27                                           LANDAREA_AVG  0.122514244
## 28                                          LANDAREA_MODE  0.121108222
## 29                                          FLOORSMIN_AVG  0.119931691
## 30                                         FLOORSMIN_MEDI  0.119713740
## 31                                         FLOORSMIN_MODE  0.118813897
## 32                                       YEARS_BUILD_MODE  0.097812861
## 33                                        YEARS_BUILD_AVG  0.097746824
## 34                                       YEARS_BUILD_MEDI  0.097576634
## 35                                      NONLIVINGAREA_AVG  0.083236095
## 36                                     NONLIVINGAREA_MEDI  0.082946427
## 37                                     NONLIVINGAREA_MODE  0.080599963
## 38                                WALLSMATERIAL_MODE_Panel  0.069907121
## 39                         WALLSMATERIAL_MODE_Stone, brick -0.048644051
## 40                              REGION_POPULATION_RELATIVE  0.046003109
## 41                               WALLSMATERIAL_MODE_Wooden -0.044008489
## 42                                 NONLIVINGAPARTMENTS_AVG  0.038711616
## 43                                NONLIVINGAPARTMENTS_MEDI  0.038678122
## 44                             REGION_RATING_CLIENT_W_CITY -0.038181801
## 45                                NONLIVINGAPARTMENTS_MODE  0.037823617
## 46                           WALLSMATERIAL_MODE_Monolithic  0.037512476
## 47                                    REGION_RATING_CLIENT -0.035394900
## 48                     NAME_EDUCATION_TYPE_Higher education  0.022859868
## 49                                  EMERGENCYSTATE_MODE_Yes -0.022759563
## 50   NAME_EDUCATION_TYPE_Secondary / secondary special -0.022251698
## 51                              YEARS_BEGINEXPLUATATION_AVG  0.021773924
## 52                             YEARS_BEGINEXPLUATATION_MEDI  0.021459819
## 53                      FONDKAPREMONT_MODE_org spec account  0.020851031
## 54                                             AMT_ANNUITY  0.020646449
## 55                                            EXT_SOURCE_2  0.020437928
## 56                             YEARS_BEGINEXPLUATATION_MODE  0.020360823
## 57                                          AMT_GOODS_PRICE  0.018617156
## 58                                  HOUR_APPR_PROCESS_START  0.018276288
## 59                                              AMT_CREDIT  0.017494166
## 60                                WALLSMATERIAL_MODE_Block -0.016544356
## 61                    NAME_INCOME_TYPE_Commercial associate  0.015996655
## 62                          HOUSETYPE_MODE_terraced house -0.015721071
## 63                                         FLAG_DOCUMENT_8  0.014126856
```

```
## 64                                FLAG_PHONE  0.012858506
## 65                     REG_CITY_NOT_WORK_CITY -0.012764557
## 66                          AMT_INCOME_TOTAL  0.011673030
## 67                 OCCUPATION_TYPE_Managers  0.010981489
## 68                 NAME_INCOME_TYPE_Working -0.010861316
## 69                          FLAG_DOCUMENT_3 -0.010789376
## 70                    LIVE_CITY_NOT_WORK_CITY -0.010182791
## 71                              EXT_SOURCE_1  0.009765311
## 72     ORGANIZATION_TYPE_Business Entity Type 3  0.009195123
## 73                 WALLSMATERIAL_MODE_Others -0.008633225
## 74     FONDKAPREMONT_MODE_reg oper spec account  0.008258600
## 75                               OWN_CAR_AGE -0.008242812
## 76                  AMT_REQ_CREDIT_BUREAU_MON  0.008168510
## 77                              FLAG_OWN_CAR  0.008011552
## 78               HOUSETYPE_MODE_block of flats  0.007845191
## 79                                FLAG_EMAIL  0.007803299
## 80                     EMERGENCYSTATE_MODE_No  0.007612181
## 81             HOUSETYPE_MODE_specific housing -0.007407588
## 82                                    TARGET -0.006963223
## 83                     REG_CITY_NOT_LIVE_CITY -0.006809516
## 84                 LIVE_REGION_NOT_WORK_REGION  0.006263323
## 85                  OCCUPATION_TYPE_Laborers -0.006230935
## 86             ORGANIZATION_TYPE_Self-employed -0.005764987
## 87                  OBS_30_CNT_SOCIAL_CIRCLE -0.005727094
## 88                  OBS_60_CNT_SOCIAL_CIRCLE -0.005657363
## 89            FONDKAPREMONT_MODE_not specified -0.005454572
## 90                   NAME_TYPE_SUITE_Family -0.005322716
## 91                  WALLSMATERIAL_MODE_Mixed  0.005207727
## 92                          FLAG_DOCUMENT_13  0.005130872
## 93                          FLAG_DOCUMENT_14  0.005123075
## 94               OCCUPATION_TYPE_Accountants  0.005112379
## 95                             CODE_GENDER_M  0.004948371
## 96        NAME_HOUSING_TYPE_Municipal apartment -0.004867813
## 97           ORGANIZATION_TYPE_Transport: type 4  0.004800406
## 98               NAME_TYPE_SUITE_Unaccompanied  0.004793122
## 99                    ORGANIZATION_TYPE_Bank  0.004687250
## 100                          FLAG_DOCUMENT_9  0.004627507
## 101                         FLAG_DOCUMENT_11  0.004593241
## 102          FONDKAPREMONT_MODE_reg oper account  0.004587181
## 103          NAME_HOUSING_TYPE_House / apartment  0.004525304
## 104                 REG_REGION_NOT_WORK_REGION  0.004391022
## 105                  DEF_60_CNT_SOCIAL_CIRCLE -0.004389529
## 106                 NAME_FAMILY_STATUS_Married  0.004293249
## 107                  DEF_30_CNT_SOCIAL_CIRCLE -0.004102940
## 108       OCCUPATION_TYPE_High skill tech staff  0.004092117
## 109               OCCUPATION_TYPE_Sales staff -0.003996212
## 110                  NAME_FAMILY_STATUS_Widow -0.003970730
## 111                            DAYS_EMPLOYED -0.003915343
## 112                           FLAG_EMP_PHONE  0.003915264
## 113               NAME_INCOME_TYPE_Pensioner -0.003914029
## 114                    ORGANIZATION_TYPE_XNA -0.003911674
## 115                AMT_REQ_CREDIT_BUREAU_YEAR -0.003874017
## 116             OCCUPATION_TYPE_Cooking staff -0.003645271
## 117                        DAYS_REGISTRATION  0.003580600
```

```
## 118             NAME_EDUCATION_TYPE_Incomplete higher  0.003558249
## 119               NAME_EDUCATION_TYPE_Lower secondary -0.003522814
## 120          ORGANIZATION_TYPE_Business Entity Type 1  0.003445275
## 121                          OCCUPATION_TYPE_Core staff  0.003159616
## 122                      OCCUPATION_TYPE_Cleaning staff -0.003147281
## 123                      OCCUPATION_TYPE_Medicine staff -0.003141932
## 124                        ORGANIZATION_TYPE_University  0.003114836
## 125                     ORGANIZATION_TYPE_Trade: type 2  0.003077345
```

```
# create new dfs with the remaining pca names
pca_application_train <- application_train |>
  select( all_of( top_pca ) )

# add target back
pca_application_train$TARGET <- application_train_dt$TARGET
```

```
# convert TARGET to factor in training data with explicit levels
application_train_dt$TARGET <- factor( application_train_dt$TARGET, levels = c( 1, 0 ) )

# split training data into training and validation sets
set.seed( 123 )
train_index <- createDataPartition( application_train_dt$TARGET,
                                    p = 0.7,
                                    list = FALSE )
train <- application_train_dt[ train_index, ]
valid <- application_train_dt[ -train_index, ]

# down sample to deal with imbalanced data issues
set.seed(123)
train <- downSample( x = train[ , -which( names( train ) == "TARGET" ) ],
                     y = train$TARGET,
                     yname = "TARGET" )

# train model on training set
tree_model <- rpart (TARGET ~ .,
                    data = train,
                    method = "class",
                    control = rpart.control( maxdepth = 5,
                                             minsplit = 20,
                                             cp = 0.01 ) )

# evaluate on validation set
valid_pred <- predict( tree_model, valid, type = "class" )
confusion_matrix <- confusionMatrix( valid_pred, valid$TARGET )
print( confusion_matrix )
```

Base Decision Tree Model

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction     1     0
##          1  3919 21241
##          0  3528 63564
##
##                Accuracy : 0.7315
##                  95% CI : (0.7286, 0.7344)
##     No Information Rate : 0.9193
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1323
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.52625
##             Specificity : 0.74953
##          Pos Pred Value : 0.15576
##          Neg Pred Value : 0.94742
##              Prevalence : 0.08072
##          Detection Rate : 0.04248
##    Detection Prevalence : 0.27273
##       Balanced Accuracy : 0.63789
##
##        'Positive' Class : 1
##
```

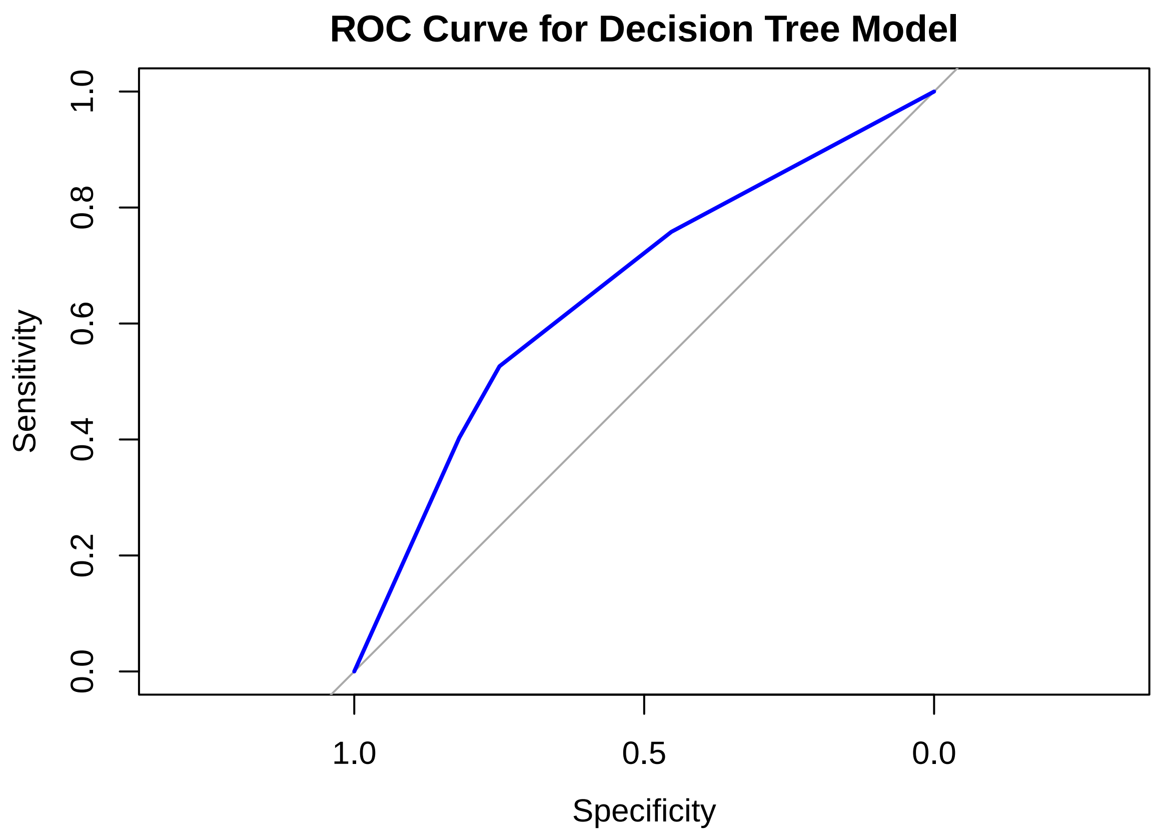
```r
# predict on test data
test_pred <- predict( tree_model, application_test_dt, type = "class" )
application_test_dt$TARGET <- test_pred

# Create a new data frame with only SK_ID_CURR and TARGET columns
kaggle_sub <- application_test_dt[ , c( "SK_ID_CURR", "TARGET" ) ]

# Export the new data frame as a CSV file
write.csv( kaggle_sub, "kaggle_sub.csv", row.names = FALSE )
```

```r
# get predicted probabilities for the validation set
valid_probabilities <- predict( tree_model, valid, type = "prob" )

# extract probabilities for positive class
valid_prob_class1 <- valid_probabilities[ , "1" ]

# generate ROC curve
roc_curve <- roc( valid$TARGET, valid_prob_class1, levels = rev(levels( valid$TARGET ) ) )
```

ROC Curve

```
## Setting direction: controls < cases
```

```r
# plot the ROC curve
plot( roc_curve, col = "blue", lwd = 2, main = "ROC Curve for Decision Tree Model" )
```




```r
# calculate AUC
auc_value <- auc( roc_curve )
print(paste( "AUC:", auc_value ) )
```

```
## [1] "AUC: 0.657368129273448"
```

```r
# convert TARGET to factor in training data with explicit levels
pca_application_train$TARGET <- factor( pca_application_train$TARGET, levels = c( 1, 0 ) )

# split training data into training and validation sets
set.seed(123)
train_index <- createDataPartition( pca_application_train$TARGET,
                                    p = 0.7,
                                    list = FALSE )
train <- pca_application_train[ train_index, ]
valid <- pca_application_train[ -train_index, ]

# down sample to deal with imbalanced data issues
set.seed(123)
```

```r
train <- downSample( x = train[ , -which( names( train ) == "TARGET" ) ],
                     y = train$TARGET,
                     yname = "TARGET" )

# train model on training set
tree_model <- rpart ( TARGET ~ .,
                      data = train,
                      method = "class",
                      control = rpart.control( maxdepth = 5,
                                               minsplit = 20,
                                               cp = 0.01 ) )

# evaluate on validation set
valid_pred <- predict( tree_model, valid, type = "class" )
confusion_matrix <- confusionMatrix( valid_pred, valid$TARGET )
print( confusion_matrix )
```

Decision Tree Model w/ PCA Data

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     1     0
##          1  4240 28427
##          0  3207 56378
##
##                Accuracy : 0.6571
##                  95% CI : (0.654, 0.6602)
##     No Information Rate : 0.9193
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.092
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.56936
##             Specificity : 0.66480
##          Pos Pred Value : 0.12979
##          Neg Pred Value : 0.94618
##              Prevalence : 0.08072
##          Detection Rate : 0.04596
##    Detection Prevalence : 0.35411
##       Balanced Accuracy : 0.61708
##
##        'Positive' Class : 1
##
```

```r
# predict on test data
test_pred <- predict( tree_model, application_test_dt, type = "class" )
application_test_dt$TARGET <- test_pred
```

```r
# create a function to evaluate model with different parameters
evaluate_tree <- function( depth, minsplit, cp, train_data, valid_data ) {

  # train model
  tree_model <- rpart( TARGET ~ .,
                       data = train_data,
                       method = "class",
                       control = rpart.control( maxdepth = depth,
                                                minsplit = minsplit,
                                                cp = cp ) )

  # predict on validation set
  valid_pred <- predict( tree_model, valid_data, type = "class" )

  # calculate metrics
  conf_matrix <- confusionMatrix( valid_pred, valid_data$TARGET )

  # return relevant metrics
  return( list( accuracy = conf_matrix$overall[ "Accuracy" ],
                sensitivity = conf_matrix$byClass[ "Sensitivity" ],
                specificity = conf_matrix$byClass[ "Specificity" ],
                balanced_accuracy = conf_matrix$byClass[ "Balanced Accuracy" ],
                kappa = conf_matrix$overall[ "Kappa" ] ) )

}

# create parameter grid
param_grid <- expand.grid( depth = c( 5, 7, 10, 15 ),
                           minsplit = c( 10, 20, 50, 100 ),
                           cp = c( 0.001, 0.01, 0.05 ) )

# store results
results <- data.frame()

# perform grid search
for( i in 1:nrow( param_grid ) ) {

  set.seed(123)

  # down sample training data
  train_balanced <- downSample( x = train[ , -which( names( train ) == "TARGET" ) ],
                                y = train$TARGET,
                                yname = "TARGET" )

  # evaluate parameters
  metrics <- evaluate_tree( depth = param_grid$depth[ i ],
                            minsplit = param_grid$minsplit[ i ],
                            cp = param_grid$cp[ i ],
                            train_data = train_balanced,
                            valid_data = valid )

  # store results
```

```r
  results <- rbind( results,
                    data.frame( depth = param_grid$depth[ i ],
                                minsplit = param_grid$minsplit[ i ],
                                cp = param_grid$cp[ i ],
                                accuracy = metrics$accuracy,
                                sensitivity = metrics$sensitivity,
                                specificity = metrics$specificity,
                                balanced_accuracy = metrics$balanced_accuracy,
                                kappa = metrics$kappa ) )
}

# sort results by balanced accuracy
results <- results[order( -results$balanced_accuracy ), ]

# print top 5 parameter combinations
print( "Top 5 parameter combinations by balanced accuracy:" )
```

Find Optimal Hyperparameters

```
## [1] "Top 5 parameter combinations by balanced accuracy:"
```

```r
print( head( results, 5 ) )
```

```
##             depth minsplit    cp  accuracy sensitivity specificity
## Accuracy11    15       50 0.001 0.6239540   0.6483148   0.6218148
## Accuracy15    15      100 0.001 0.6239540   0.6483148   0.6218148
## Accuracy3     15       10 0.001 0.6317261   0.6374379   0.6312246
## Accuracy7     15       20 0.001 0.6317261   0.6374379   0.6312246
## Accuracy2     10       10 0.001 0.6250813   0.6434806   0.6234656
##             balanced_accuracy      kappa
## Accuracy11          0.6350648 0.09634213
## Accuracy15          0.6350648 0.09634213
## Accuracy3           0.6343312 0.09769467
## Accuracy7           0.6343312 0.09769467
## Accuracy2           0.6334731 0.09557423
```

```r
# convert TARGET to factor in training data with explicit levels
application_train_dt$TARGET <- factor( application_train_dt$TARGET, levels = c( 1, 0 ) )

# split training data into training and validation sets
set.seed( 123 )
train_index <- createDataPartition( application_train_dt$TARGET,
                                    p = 0.7,
                                    list = FALSE )
train <- application_train_dt[ train_index, ]
valid <- application_train_dt[ -train_index, ]

# down sample to deal with imbalanced data issues
set.seed( 123 )
train <- downSample( x = train[, -which( names( train ) == "TARGET" )],
```

```r
                     y = train$TARGET,
                     yname = "TARGET" )

# train model on training set
tree_model <- rpart ( TARGET ~ .,
                      data = train,
                      method = "class",
                      control = rpart.control( maxdepth = 10,
                                               minsplit = 100,
                                               cp = 0.001 ) )

# evaluate on validation set
valid_pred <- predict( tree_model, valid, type = "class" )
confusion_matrix <- confusionMatrix( valid_pred, valid$TARGET )
print( confusion_matrix )
```

Decision Tree w/ Optimal Hyper Parameters

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     1     0
##          1  4881 28618
##          0  2566 56187
##
##                Accuracy : 0.662
##                  95% CI : (0.6589, 0.665)
##     No Information Rate : 0.9193
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1225
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.65543
##             Specificity : 0.66254
##          Pos Pred Value : 0.14571
##          Neg Pred Value : 0.95633
##              Prevalence : 0.08072
##          Detection Rate : 0.05291
##    Detection Prevalence : 0.36312
##       Balanced Accuracy : 0.65899
##
##        'Positive' Class : 1
##
```

```r
# predict on test data
test_pred <- predict( tree_model, application_test_dt, type = "class" )
application_test_dt$TARGET <- test_pred
```