Project 3 Logistic Regression

Black_Boopathy

2025-11-17

Read in the data

The following dataset will be used to predict `loan_default` . We are using train and holdout sets for the model creation and testing the performance. Train set has 226434 obs.of 35 variables and holdout has 75555 obs. of 36 variables. We will be utilizing different strategies to receive the best ROC, AUC, sensitivity, and specificity possible. The strategy we will be using in this part will be logistic regression.

```
df=readRDS("group5AA_Black-Boopathy_train.rds")
holdout=readRDS("holdout_df.rds")
levels(df$loan_default)
```

```
## [1] "No"  "Yes"
```

Setting the Reference level

we are using relevel() for making the baseline as “Yes” and ensure by levels().

```
df$loan_default<-relevel(df$loan_default, ref= "Yes")
levels(df$loan_default)
```

```
## [1] "Yes" "No"
```

Downsample majority to match minority

Here, majority class is “No” with 182927 observations.For making a balanced samples we did Downsample majority to match minority and save the details in df.us dataframe.

```
# Downsample majority to match minority

set.seed(123)
df$loan_default <- as.factor(df$loan_default)

df.us=downSample(x=df %>% select(-loan_default),
                      y=df$loan_default,
                      yname="loan_default"


  )
```

Check the balanced Dataset

Now our dataset has balanced and ready for model fitting.

```
table(df$loan_default)
```

```
##
##    Yes     No
##  43507 182927
```

```
table(df.us$loan_default)
```

```
##
##   Yes    No
## 43507 43507
```

Fit a Logistic Regression Model Using ElasticNet selection with caret

Set up a trainControl object

method = “cv”, number = 10, We are using 10-fold cross validation. We will use this to select the lasso parameter, λ.

```
ctrl <- trainControl(method = "cv", number = 10,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = 'final')
```

TuneGrid for setting alpha and lambda

Our tuneGrid has two variables. One is alpha sequence starts with 0.1 and ends with 1 having increment as 0.2. lambda has sequence starts with 0.001 and ends with 0.1 and it has 10 equally spaced values. we are using the elastic Net approach.

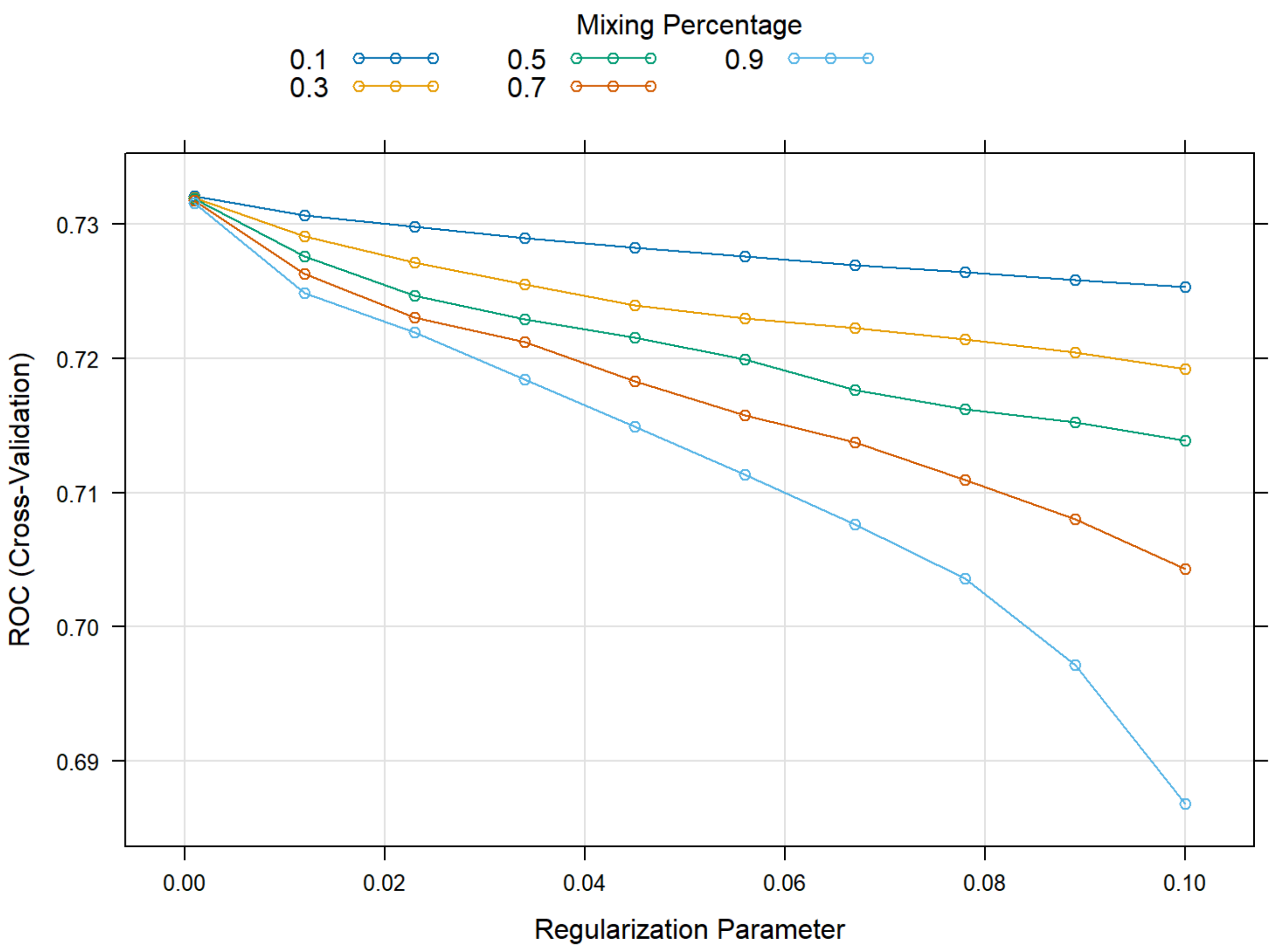
```
tuneGrid <- expand.grid(
  alpha = seq(0.1, 1, by = 0.2),         # Elastic Net mixing
  lambda = seq(0.001, 0.1, length = 10)    # Regularization strength
)
```

Fit the Model

```
logit_fit <- train(
   loan_default~ .,
  data = df.us,
  method = "glmnet",
  trControl = ctrl,
  tuneGrid =tuneGrid,
  metric = "ROC",
  family = "binomial"
)
```

Below we look at the optimal value of lambda and explore the model

```
plot(logit_fit)
```



```
logit_fit$bestTune
```

```
##   alpha lambda
## 1   0.1  0.001
```

Analysis

The above results shows that, the best tune for alpha= 0.1 and lambda= 0.001.

```
logit_fit$results
```

```
##    alpha lambda       ROC      Sens      Spec       ROCSD      SensSD
## 1    0.1  0.001 0.7320810 0.6623990 0.6710182 0.005109855 0.007067412
## 2    0.1  0.012 0.7306454 0.6558023 0.6767414 0.005390501 0.007935525
## 3    0.1  0.023 0.7297970 0.6540095 0.6780746 0.005489227 0.007454022
## 4    0.1  0.034 0.7289557 0.6516880 0.6797295 0.005559164 0.007734586
## 5    0.1  0.045 0.7282641 0.6493666 0.6803042 0.005630399 0.007492343
## 6    0.1  0.056 0.7275661 0.6473439 0.6812236 0.005709109 0.007870143
## 7    0.1  0.067 0.7269493 0.6450914 0.6823268 0.005764026 0.008416698
## 8    0.1  0.078 0.7263989 0.6432526 0.6834761 0.005780193 0.009057949
## 9    0.1  0.089 0.7258690 0.6420803 0.6844185 0.005797779 0.008919017
## 10   0.1  0.100 0.7253280 0.6407243 0.6854068 0.005817677 0.008129314
## 11   0.3  0.001 0.7319804 0.6613876 0.6722364 0.005143880 0.007128670
## 12   0.3  0.012 0.7291146 0.6515961 0.6801202 0.005454485 0.007661692
## 13   0.3  0.023 0.7271393 0.6417126 0.6852919 0.005733352 0.007620563
## 14   0.3  0.034 0.7254907 0.6341735 0.6903026 0.005825534 0.008055464
## 15   0.3  0.045 0.7239325 0.6288870 0.6918195 0.005819474 0.007462729
## 16   0.3  0.056 0.7230033 0.6269793 0.6918655 0.005820672 0.007729245
## 17   0.3  0.067 0.7222951 0.6257381 0.6923482 0.005838273 0.007753064
## 18   0.3  0.078 0.7214374 0.6236236 0.6931066 0.005856649 0.007372080
## 19   0.3  0.089 0.7204105 0.6214630 0.6937962 0.005868488 0.007761315
## 20   0.3  0.100 0.7192217 0.6190726 0.6944398 0.005863053 0.008434487
## 21   0.5  0.001 0.7318794 0.6605142 0.6730409 0.005168831 0.007147667
## 22   0.5  0.012 0.7276109 0.6441031 0.6845105 0.005697409 0.007507361
## 23   0.5  0.023 0.7246440 0.6278987 0.6938422 0.005801150 0.007738804
## 24   0.5  0.034 0.7229191 0.6213021 0.6969911 0.005813350 0.007427670
## 25   0.5  0.045 0.7215773 0.6173028 0.6984390 0.005824510 0.007515397
## 26   0.5  0.056 0.7199104 0.6151882 0.6981633 0.005801060 0.008552531
## 27   0.5  0.067 0.7176734 0.6133034 0.6959338 0.005771661 0.008725232
## 28   0.5  0.078 0.7162372 0.6137861 0.6957040 0.005770593 0.007834188
## 29   0.5  0.089 0.7152388 0.6130737 0.6949684 0.005741352 0.008106282
## 30   0.5  0.100 0.7138905 0.6133725 0.6940030 0.005702814 0.008407594
## 31   0.7  0.001 0.7317407 0.6598017 0.6740062 0.005193689 0.006839356
## 32   0.7  0.012 0.7262659 0.6367249 0.6890844 0.005770057 0.007782306
## 33   0.7  0.023 0.7230113 0.6188428 0.6989908 0.005806199 0.007225123
## 34   0.7  0.034 0.7211957 0.6153491 0.6995883 0.005783085 0.007833944
## 35   0.7  0.045 0.7182817 0.6121772 0.6972209 0.005769776 0.008626908
## 36   0.7  0.056 0.7157751 0.6115337 0.6966233 0.005740206 0.007900108
## 37   0.7  0.067 0.7137695 0.6107522 0.6952212 0.005697658 0.008719972
## 38   0.7  0.078 0.7109737 0.6099247 0.6929917 0.005674799 0.008359256
## 39   0.7  0.089 0.7080038 0.6114647 0.6880041 0.005776682 0.007900254
## 40   0.7  0.100 0.7043361 0.6113727 0.6821659 0.005832881 0.007976026
## 41   0.9  0.001 0.7315618 0.6591581 0.6744660 0.005223455 0.007063771
## 42   0.9  0.012 0.7248344 0.6288640 0.6935664 0.005763518 0.007952330
## 43   0.9  0.023 0.7219677 0.6151882 0.7008985 0.005773492 0.008069799
## 44   0.9  0.034 0.7184568 0.6119014 0.6981403 0.005764113 0.008376583
## 45   0.9  0.045 0.7149121 0.6097639 0.6972209 0.005715929 0.008166785
## 46   0.9  0.056 0.7113576 0.6087526 0.6941180 0.005668679 0.007813152
## 47   0.9  0.067 0.7076252 0.6105223 0.6873604 0.005827885 0.008213465
## 48   0.9  0.078 0.7036153 0.6111201 0.6804881 0.005874682 0.007245336
## 49   0.9  0.089 0.6971737 0.6104765 0.6732709 0.005914476 0.007576879
## 50   0.9  0.100 0.6868089 0.6036960 0.6650883 0.006035137 0.006932825
##         SpecSD
## 1  0.007653919
## 2  0.007986515
## 3  0.008352788
## 4  0.007810695
## 5  0.007561539
## 6  0.007406778
## 7  0.007165441
## 8  0.007343962
## 9  0.006778450
## 10 0.007132841
## 11 0.007986296
## 12 0.007804238
## 13 0.007562796
## 14 0.006698164
## 15 0.007439151
## 16 0.007834114
## 17 0.007896367
## 18 0.007395381
## 19 0.007395082
## 20 0.006811396
## 21 0.008226030
## 22 0.007620363
## 23 0.007100333
## 24 0.007902944
## 25 0.007825364
## 26 0.007460814
## 27 0.006601458
## 28 0.006437042
## 29 0.006978210
## 30 0.007143883
## 31 0.008263773
## 32 0.006833979
## 33 0.007986092
## 34 0.008005186
## 35 0.007166763
## 36 0.007324056
## 37 0.007698136
## 38 0.007694415
## 39 0.007227619
## 40 0.006593401
## 41 0.008326281
## 42 0.006701222
## 43 0.007751244
## 44 0.007464023
## 45 0.007740282
## 46 0.008034637
## 47 0.006556804
## 48 0.006868586
## 49 0.007021582
## 50 0.007016852
```

Analysis:

The component logit_fit$results gives us a data frame of performance metrics for each combination of tuning parameters. In this case, lambda was selected using the AUC value because we set the metric to ROC. The best lambda value was 0.001.

coefficients of the model

```
coef(logit_fit$finalModel,logit_fit$bestTune$lambda)
```

```
## 35 x 1 sparse Matrix of class "dgCMatrix"
##                                               s=0.001
## (Intercept)                             -2.904017e+00
## loan_amnt                               -1.609213e-05
## int_rate                                -2.226985e-02
## annual_inc                               1.652622e-06
## dti                                     -3.011580e-02
## open_acc                                -1.763524e-02
## pub_rec                                 -8.201484e-02
## revol_bal                                3.637180e-06
## revol_util                              -6.918638e-04
## total_acc                                1.112370e-02
## mort_acc                                 3.354727e-02
## pub_rec_bankruptcies                     1.277691e-01
## fico_range_low                           3.424633e-03
## fico_range_high                          3.455276e-03
## inq_last_6mths                          -8.099625e-02
## `term_ 60 months`                       -5.344819e-01
## home_ownership_OWN                      -1.551721e-01
## home_ownership_RENT                     -2.553655e-01
## `verification_status_Source Verified` -1.080176e-01
## verification_status_Verified            -2.444405e-03
## grade_B                                 -2.965414e-01
## grade_C                                 -6.328532e-01
## grade_D                                 -8.114558e-01
## grade_E                                 -9.364500e-01
## grade_F                                 -1.018105e+00
## grade_G                                 -1.050885e+00
## purpose_debt_consolidation              -9.312350e-02
## purpose_home_improvement                -1.919233e-01
## purpose_other                           -2.169272e-01
## initial_list_status_w                    2.734102e-02
## debt_settlement_flag_Y                  -5.347096e+00
## `emp_length_4-6 years`                   3.950709e-02
## `emp_length_7-9 years`                   2.231083e-02
## `emp_length_10+ years`                   4.571754e-02
## interaction_dti_interest                 1.578940e-04
```

Predict on the Test Set

Below we will create a dataframe that includes the predictions on the test data along with the actual values.

```
pred.test <- holdout %>% dplyr::select(loan_default)
pred.test$netprob <- predict(logit_fit,
                           newdata = holdout,
                           type = "prob")[,"Yes"]

pred.test$netclass<- predict(logit_fit,
                                 newdata = holdout,
                                 type="raw")
```

Evaluating the performance with confusion matrix

```
table(pred.test$loan_default)
```

```
##
##    No   Yes
## 61026 14529
```

```
table(pred.test$netclass)
```

```
##
##   Yes    No
## 29579 45976
```

```
confusionMatrix(pred.test$netclass, pred.test$loan_default,positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  40964  5012
##        Yes 20062  9517
##
##                Accuracy : 0.6681
##                  95% CI : (0.6648, 0.6715)
##     No Information Rate : 0.8077
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.234
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6550
##             Specificity : 0.6713
##          Pos Pred Value : 0.3217
##          Neg Pred Value : 0.8910
##              Prevalence : 0.1923
##          Detection Rate : 0.1260
##    Detection Prevalence : 0.3915
##       Balanced Accuracy : 0.6631
##
##        'Positive' Class : Yes
##
```

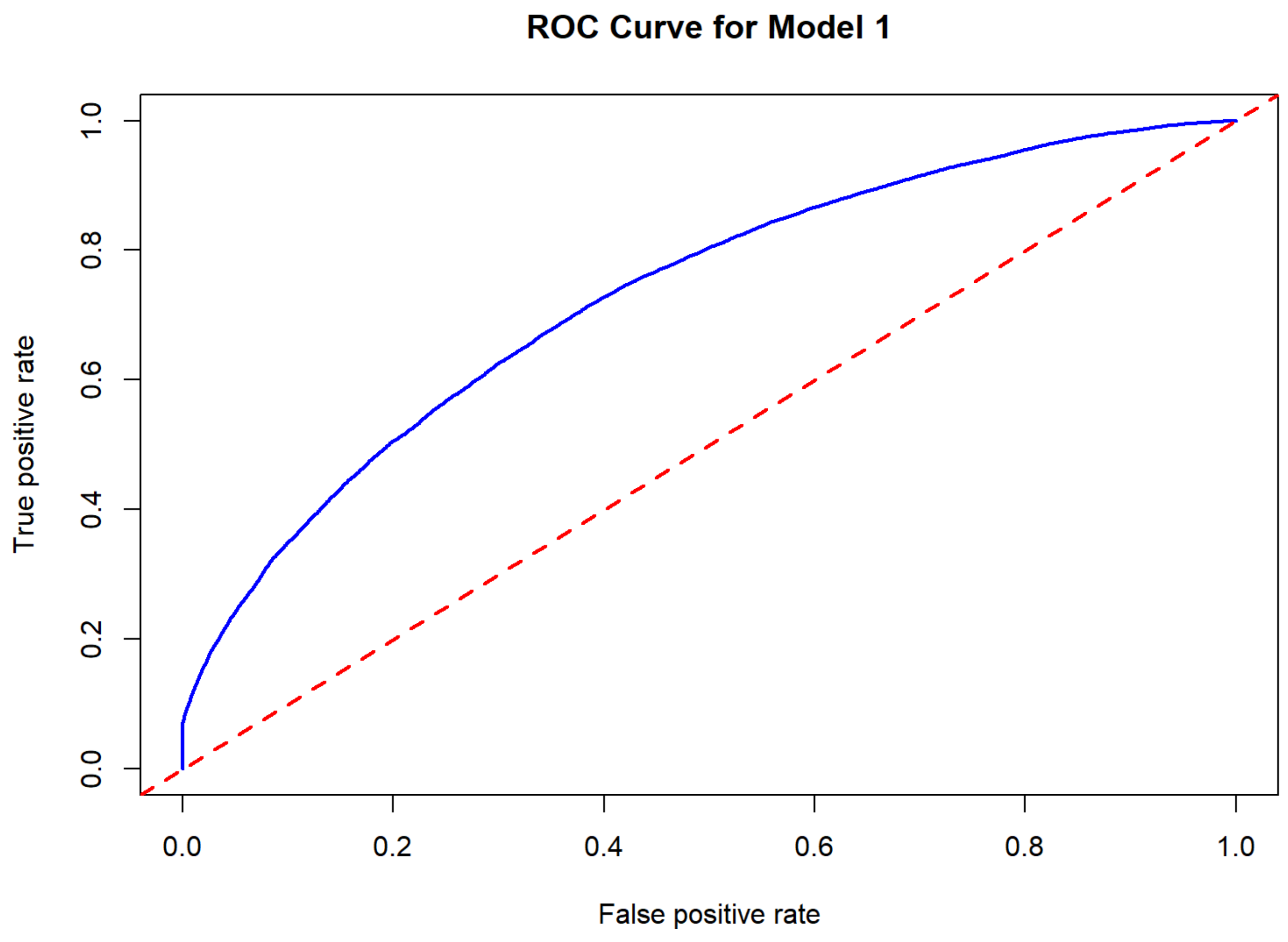
Analysis:

The sensitivity level is 65% Specificity has 67% and Balanced Accuracy has 66%. the model stability is reasonably good in finding the true positives.

ROC Curve

Now we access the Visualize trade-offs between sensitivity and specificity across all thresholds.

```
pred <- prediction(pred.test$netprob, pred.test$loan_default)
perf <- performance(pred, "tpr", "fpr")
# Plot Curve
plot(perf, col = "blue", lwd = 2, main = "ROC Curve for Model 1")
abline(a = 0, b = 1, col = "red", lty = 2, lwd = 2)
```



```
auc <- performance(pred, "auc")@y.values[[1]]
print(paste("The AUC is ",round(auc,2)))
```

```
## [1] "The AUC is  0.73"
```

Step 1- Finding the Optimal Threshold using ROCR package

Generally we use the F1 Score to find the optimal threshold. Using the ROCR package, we can compute the precision and recall for each threshold We use this vector of values to compute the F1 Score for each threshold. The threshold that gives the highest F1 Score is the optimal threshold.

```
pred <- ROCR::prediction(pred.test$netprob,holdout$loan_default)
prec <- ROCR::performance(pred, "prec")
rec <- ROCR::performance(pred, "rec")
precision <- prec@y.values[[1]]
recall <- rec@y.values[[1]]
f1=2*precision*recall/(precision+recall)
f1[is.nan(f1)]=0
cutoffs=prec@x.values[[1]]
df_f1=data.frame(f1,cutoffs)

opt_idx <- which.max(f1)
opt_f1 <- df_f1[opt_idx,]
opt_f1
```

```
##              f1   cutoffs
## 27130 0.4342023 0.5187898
```

Analysis

Our best possible cutoff of f1 is 0.5187898

Step-2 Finding the model performance Using Youden’s J statistic

```
pred <- ROCR::prediction(pred.test$netprob,holdout$loan_default)
perf=performance(pred,"tpr","fpr")
tpr=perf@y.values[[1]]
fpr=perf@x.values[[1]]
thr=perf@alpha.values[[1]]
j=tpr-fpr
best_j=thr[which.max(j)]
print(best_j)
```

```
## [1] 0.4683588
```

Analysis

Our best possible cutoff for Youden’s J statistic is 0.4683588

Evaluating the performance with the F1 threshold

value

```
pred.test$netclass_51 <- factor(ifelse(pred.test$netprob > 0.5187898,
                                       "Yes","No"),
                                levels=c("Yes","No"))

table(pred.test$loan_default)
```

```
## 
##    No   Yes 
## 61026 14529
```

```
table(pred.test$netclass_51)
```

```
## 
##   Yes    No 
## 27128 48427
```

```
confusionMatrix(pred.test$netclass_51, pred.test$loan_default,positive="Yes")
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    No   Yes
##        No  42941  5486
##        Yes 18085  9043
##                                           
##                Accuracy : 0.688           
##                  95% CI : (0.6847, 0.6913)
##     No Information Rate : 0.8077          
##     P-Value [Acc > NIR] : 1               
##                                           
##                   Kappa : 0.2451          
##                                           
##  Mcnemar's Test P-Value : <2e-16          
##                                           
##             Sensitivity : 0.6224          
##             Specificity : 0.7037          
##          Pos Pred Value : 0.3333          
##          Neg Pred Value : 0.8867          
##              Prevalence : 0.1923          
##          Detection Rate : 0.1197          
##    Detection Prevalence : 0.3590          
##       Balanced Accuracy : 0.6630          
##                                           
##        'Positive' Class : Yes             
## 
```

Evaluating the performance with the Youden threshold value

```
pred.test$netclass_46 <- factor(ifelse(pred.test$netprob > 0.4683588,
                                       "Yes","No"),
                                levels=c("Yes","No"))

table(pred.test$loan_default)
```

```
##
##    No   Yes
## 61026 14529
```

```
table(pred.test$netclass_46)
```

```
##
##   Yes    No
## 33888 41667
```

```
confusionMatrix(pred.test$netclass_46, pred.test$loan_default,positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  37525  4142
##        Yes 23501 10387
##
##                Accuracy : 0.6341
##                  95% CI : (0.6307, 0.6376)
##     No Information Rate : 0.8077
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2188
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7149
##             Specificity : 0.6149
##          Pos Pred Value : 0.3065
##          Neg Pred Value : 0.9006
##              Prevalence : 0.1923
##          Detection Rate : 0.1375
##    Detection Prevalence : 0.4485
##       Balanced Accuracy : 0.6649
##
##        'Positive' Class : Yes
##
```

Comparision of Two optimal cutoff for selecting the best performance model based on the confusion matrix

We are in the stage of finding the best possible optimal cutoffs impact on tp,fn values.We checked with the two optimal cutoffs of 0.5187898 and 0.4683588 so we made decision with the level of 0.4683588 as possible threshold. So Youden threshold gives the best accuracy of finding the loan_default with its best in capturing the tp(Actual Defaulters) and minimizing fn (false negatives). Since our goal is to reduce missed defaulters, the Youden threshold achieves the best recall at an acceptable false-positive rate. Hence, using the selected youden's threshold in logistic regression model, we achieve the better predictions in loan_default.