HW – 3
Ensemble Techniques
Machine Learning, Section INET

Sharath Katta Sridhar
sk8671

sk8671@nyu.edu

Table of Contents

1) Loading Libraries: ... 2
2) Loading Data and Pre-processing based on EDA: ... 2
3) Train Test Split: ... 3
4) Bagging with Decision Trees ... 4
Setting up Parallel Procession Environment: ... 4
Bagging Classifier: ... 4
Running Bagging Classifier: ... 5
Results of Bagging : ... 5
Calculating Variance : ... 6
5) Random Forest ... 6
Function to perform Grid Search: ... 6
Grid Search – 1 ... 7
Grid Search – 2 ... 7
Result of Grid Search ... 8
Training the best model based on Grid Search ... 8
Results of Random Forest : ... 9
Calculating Variance of Random Forest : ... 9
6) Cross Validation ... 10
Function to perform CV ... 10
Function to get Evaluation Metric: ... 10
Performing CV ... 11
Function to get results on Test Data ... 11
Evaluating Test Set ... 11
Results of Cross Validation: ... 12
Calculating Variance of Cross Validation : ... 12
7) Q. What does CV do to bias and variance? ... 12
8) Summary ... 13
The Last attempt with Under and Over Sampling: ... 14
Under Sampling ... 14
1) Training Random Forest (Under Sampling) ... 14
2) Training KNN (Under Sampling) ... 15
Over Sampling ... 15
1) Training Random Forest (Over Sampling) ... 16
2) Training KNN (Over Sampling) ... 16
Observations: ... 17

1) Loading Libraries:

```
library(tidyverse)
library(RColorBrewer)
library(dlookr)
library(ggcorrplot)
library(plyr)
library(dplyr)
library(cowplot)
library(rsample)  # data splitting
library(rpart)
library(nnet)
library(ggplot2)  # data visualization
library(caret)
library(party)
library(pROC)
library(ROCR)
library(rpart.plot)
library(e1071)        # SVM methodology
library(class)
library(parallel)     # For running in parallel
library(foreach)
library(doParallel)
```

2) Loading Data and Pre-processing based on EDA:

```
#############################################################################
## Setting Seed
set.seed(20)


#############################################################################
## Loading Data
diabetes = read.csv("diabetic_data.csv", stringsAsFactors=F)

#############################################################################
## Feature List

numeric.features <- c("time_in_hospital","num_lab_procedures","num_procedures","num_medications",
                      "number_outpatient","number_emergency","number_inpatient","number_diagnoses")

cat.features <- c("race","gender","age","admission_type_id","discharge_disposition_id",
                  "admission_source_id","change","diabetesMed")

medical.features <- c("metformin","repaglinide","nateglinide","chlorpropamide", "glimepiride",
                      "glipizide","glyburide","tolbutamide","pioglitazone",
                      "rosiglitazone","acarbose","miglitol","troglitazone","tolazamide",
                      "insulin","glyburide.metformin","glipizide.metformin")

target.attr <- c("readmitted")


all.features <- c(numeric.features,cat.features,medical.features)



## Significant Features based on EDA
sig.attrs <- c("discharge_disposition_id","admission_source_id","insulin","diabetesMed",
               "admission_type_id","age","race","change","metformin","repaglinide",
               "number_inpatient","number_diagnoses","number_emergency","number_outpatient",
               "num_medications","time_in_hospital","num_procedures","num_lab_procedures")
```

```
## Function Preprocess the data
preprocess_data <- function(input.df) {

  ### Selecting Required Features
  df <- select(input.df, append(all.features,target.attr))

  df$readmitted[df$readmitted=="<30"]<-as.integer(0)
  df$readmitted[df$readmitted==">30"]<-as.integer(1)
  df$readmitted[df$readmitted=="NO"]<-as.integer(2)

  ### Converting Categorical attribute into Factors
  colls.to.factor <-c(cat.features,medical.features,target.attr)

  df[colls.to.factor] <- lapply(df[colls.to.factor], as.factor)


  ### Converting Age attribute into Ordered Factor
  age.range <- c("[0-10)", "[10-20)","[20-30)","[30-40)", "[40-50)",
                 "[50-60)","[60-70)","[70-80)", "[80-90)","[90-100)")

  df$age <- factor(df$age, order = TRUE, levels = age.range)

  ### Square Root transform based on EDA
  cols.to.sqrt <- c("num_lab_procedures","num_medications")

  df[cols.to.sqrt] <- lapply(df[cols.to.sqrt], sqrt)

  df
}

diabetes.processed <- preprocess_data(diabetes)

diabetes.imp <- select(diabetes.processed, c(all_of(sig.attrs),"readmitted"))
```

3) Train Test Split:

```
split <- initial_split(diabetes.imp, prop = .7, strata = "readmitted")
train <- training(split)
test  <- testing(split)

## Checking distribution of target
table(train$readmitted) %>% prop.table()
## <30       >30        NO
## 0.1115900 0.3492855 0.5391246

table(test$readmitted) %>% prop.table()
## <30       >30        NO
## 0.1116206 0.3492729 0.5391065


###############################################################################
## Creating a split for variance calculation
split_v <- initial_split(diabetes.imp, prop = .9, strata = "readmitted")
train_v <- training(split_v)
test_v  <- testing(split_v)

## Making two training sets for training:
split_train <- initial_split(train_v, prop = .5, strata = "readmitted")
train_v1 <- training(split_train)
train_v2  <- testing(split_train)

## Number of Observations in Train Set 1
nrow(train_v1)
## [1] 45793

## Number of Observations in Train Set 2
nrow(train_v2)
## [1] 45795

## Number of Observations in Test Set
nrow(test_v)
## [1] 10178
```

4) Bagging with Decision Trees

This is similar to Random Forest, since Random Forest also uses bagging ensemble technique. But I have done following variation so that it is different from normal Random Forest.

- All the observations are being used for training, instead of taking a sample of observation for each tree.
- All the features are being used when creating individual trees.
- Max Voting is being used to predict the combined outcome of predictors.

Setting up Parallel Procession Environment:

```
# Create a parallel socket cluster
cl <- makeCluster(64) # use 64 workers
registerDoParallel(cl)
```

Here we have setup a parallel processing cluster with **100 cores** , which can process our algorithm in parallel. Well this is a tremendous power !!

```
## Mode function to perform voting between predictors
mode<-function(x){which.max(tabulate(x))}
```

Bagging Classifier:

```
bagging_classifier <- function(train, test) {

  predictions <- foreach(
    icount(160),
    .packages = "rpart",
    .combine = cbind
  ) %dopar% {
    # bootstrap copy of training data
    index <- sample(nrow(train), replace = TRUE)
    train_boot <- train[index, ]

    # fit tree to bootstrap copy
    bagged_tree <- rpart(
      readmitted ~ .,
      control = rpart.control(minsplit = 100,maxdepth = 7, cp = 0),
      data = train_boot,
      method = "class"
    )

    pred <- predict(bagged_tree, newdata = test)

    df <- data.frame(pred)
    colnames(df) <- c(0,1,2)

    factor(colnames(df)[max.col(df)], levels = c(0,1,2))
  }

  ## Performing Max Voting
  output <- apply(predictions,1,mode)
  test.predictions = as.factor(output-1)
  test.predictions
}
```

Running Bagging Classifier:

```
system.time({

  test.predictions <- bagging_classifier(train, test)
})

##  user  system elapsed
## 5.971   3.761 154.322
```

This is amazing, it only took around 2.5 minutes to run 160 predictors with 71234 observations each.

Results of Bagging :

```
## Test results evaluation
confusionMatrix(test$readmitted, test.predictions)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0    75  1288  2044
##          1    70  3781  6812
##          2    27  2642 13790
##
## Overall Statistics
##
## Accuracy : 0.578
## 95% CI : (0.5724, 0.5836)
## No Information Rate : 0.7418
## P-Value [Acc > NIR] : 1
##
## Kappa : 0.1746
##
## Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity          0.436047   0.4903   0.6089
## Specificity          0.890239   0.6984   0.6614
## Pos Pred Value       0.022014   0.3546   0.8378
## Neg Pred Value       0.996424   0.8022   0.3706
## Prevalence           0.005634   0.2526   0.7418
## Detection Rate       0.002457   0.1238   0.4517
## Detection Prevalence 0.111599   0.3493   0.5391
## Balanced Accuracy    0.663143   0.5944   0.6352
```

	Bagging	Individual Classifier
Accuracy	0.578	0.5844

	Class : <30	Class: >30	Class : NO
Sensitivity (Recall)	0.436047	0.4903	0.6089
Precision	0.02201	0.35459	0.8378
Specificity	0.890239	0.6984	0.6614

The performance was not much improved as compared to the same individual classifier.

Calculating Variance :

```
### Function to calculate variance of model
cal_variance <- function(pred1, pred2) {

  ## Taking the mean of count of observations that are missmatching
  ## in the two different test predictions

  cf_tab <- table(pred1 , pred2)
  variance <- (length(pred1) - sum(diag(cf_tab)))/length(pred1)
  variance
}

### Calculating Variance
test.pred.m1 <- bagging_classifier(train_v1, test_v)
test.pred.m2 <- bagging_classifier(train_v2, test_v)

cal_variance(test.pred.m1,test.pred.m2)
## [1] 0.0839312
```

Now we can see the improvements, the variance is 0.08 which is half as compared to individual classifier (0.18). The Bias cannot be calculated.

5) Random Forest

Making a cluster of 64 cores to perform hyperparameter grid search for best random forest model.

```
library(ranger)
library(parallel)
library(foreach)
library(doParallel)

cl <- makeCluster(64) # use 64 workers
registerDoParallel(cl)

(n_features <-length(sig.attrs))
## [1] 18
```

Function to perform Grid Search:

```
############
## Function to Perform Grid Search for best model
search_grid_rf <- function(grid, train_data) {

  grid_size <- nrow(grid)

  oob.output <- foreach(
    i = icount(grid_size),
    .packages = "ranger",
    .combine = cbind
  ) %dopar% {

    # train model
    model <- ranger(
      formula         = readmitted ~ .,
      data            = train_data,
      num.trees       = 500,
      mtry            = grid$mtry[i],
      min.node.size   = grid$node_size[i],
      sample.fraction = grid$sampe_size[i],
      seed            = 20
    )

    sqrt(model$prediction.error)
  }
  grid$OOB_RMSE <- oob.output[1,]
  grid

}
```

Grid Search – 1

```
######## Grid Search 1 ( For mtry )
hyper_grid.1 <- expand.grid(
  mtry       = floor(n_features * c(.05, .15, .25, .333, .4)),
  node_size  = 6,
  sampe_size = 0.7,
  OOB_RMSE   = 0
)

(grid_size <- nrow(hyper_grid.1))
## [1] 5

#######
###  Scanning the hyper_grid.1

system.time({

  hyper_grid.1 <- search_grid_rf(hyper_grid.1, train)
})

##  user  system elapsed
## 1.908   0.706  12.813

hyper_grid.1 %>%
  dplyr::arrange(OOB_RMSE) %>%
  head(10)

##  mtry node_size sampe_size  OOB_RMSE
## 1    2         6        0.7 0.6453441
## 2    0         6        0.7 0.6454202
## 3    4         6        0.7 0.6454202
## 4    5         6        0.7 0.6481549
## 5    7         6        0.7 0.6503710
```

Grid Search – 2

```
######## Grid Search 2 ( For node_size & sampe_size )
hyper_grid.2 <- expand.grid(
  mtry       = 2,
  node_size  = seq(1, 12, by = 2),
  sampe_size = c(.55, .632, .70, .80),
  OOB_RMSE   = 0
)

(grid_size <- nrow(hyper_grid.2))
## [1] 24

#######
###  Scanning the hyper_grid.2

system.time({

  hyper_grid.2 <- search_grid_rf(hyper_grid.2, train)
})
##   user  system elapsed
## 91.135  14.894 147.354

hyper_grid.2 %>%
  dplyr::arrange(OOB_RMSE) %>%
  head(10)

##    mtry node_size sampe_size  OOB_RMSE
## 1     2         7      0.550 0.6439722
## 2     2        11      0.800 0.6440594
## 3     2         9      0.700 0.6442120
## 4     2         9      0.800 0.6443100
## 5     2        11      0.700 0.6443972
## 6     2         7      0.800 0.6443972
## 7     2        11      0.632 0.6445714
## 8     2         9      0.632 0.6446258
## 9     2         3      0.800 0.6446803
## 10    2         7      0.632 0.6447021
```

Result of Grid Search

Two separate grid searches were performed because the IBM machine was unable to process a grid of size 120 in one go.

Final RMSE shows that the best hyper parameter combination is as follows:

mtry	2
node.size	7
Sample.fraction	0.55

The Grid Search was performed in parallel with 64 cores. The processing included training of a total of 24*500 (12000) decision trees and it only took 2 minutes 27 seconds to do so. Without parallel processing this working take a tremendous amount of time.

Training the best model based on Grid Search

```
## Using the best model parameters
model.rf <- ranger(
  formula         = readmitted ~ .,
  data            = train,
  num.trees       = 500,
  mtry            = 2,
  min.node.size   = 7,
  sample.fraction = 0.55,
  seed            = 20
)

test.pred <- predict(model.rf, test[sig.attrs])
confusionMatrix(test.pred$predictions, test$readmitted)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0    14     3     0
##          1  1125  3134  1929
##          2  2268  7526 14530
##
## Overall Statistics
##
## Accuracy : 0.5791
## 95% CI : (0.5735, 0.5846)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16
##
## Kappa : 0.1574
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                       Class: 0 Class: 1 Class: 2
## Sensitivity          0.0041092   0.2939   0.8828
## Specificity          0.9998894   0.8463   0.3039
## Pos Pred Value       0.8235294   0.5065   0.5974
## Neg Pred Value       0.8887979   0.6907   0.6891
## Prevalence           0.1115988   0.3493   0.5391
## Detection Rate       0.0004586   0.1027   0.4759
## Detection Prevalence 0.0005568   0.2027   0.7968
## Balanced Accuracy    0.5019993   0.5701   0.5934
```

Results of Random Forest :

	Random Forest
Accuracy	0.5791

	Class : <30	Class: >30	Class : NO
Sensitivity (Recall)	0.0041092	0.2939	0.8828
Precision	0.8235	0.50646	0.5973
Specificity	0.890239	0.8463	0.3039

The Precision for the class "<30" was improved as compared to the other algorithms.

Calculating Variance of Random Forest :

```
######### Calculating Variance of Random Forest

## Training the model with train_v1
model.rf.v1 <- ranger(
  formula         = readmitted ~ .,
  data            = train_v1,
  num.trees       = 500,
  mtry            = 2,
  min.node.size   = 7,
  sample.fraction = 0.55,
  seed            = 20
)

test.pred.m1 <- predict(model.rf.v1, test_v[sig.attrs])

## Training the model with train_v2
model.rf.v2 <- ranger(
  formula         = readmitted ~ .,
  data            = train_v2,
  num.trees       = 500,
  mtry            = 2,
  min.node.size   = 7,
  sample.fraction = 0.55,
  seed            = 20
)

test.pred.m2 <- predict(model.rf.v2, test_v[sig.attrs])

## Variance
cal_variance(test.pred.m1$predictions,test.pred.m2$predictions)
## [1] 0.07321867
```

We can see the improvements, the variance is 0.073 which is half as compared to individual classifier (0.18) of Decision Trees.

The Bias cannot be calculated.

6) Cross Validation

Making a cluster of 64 cores to perform cross validation over 10 folds in parallel.

```
library(foreach)
library(caret)
library(doParallel)
cl <- makeCluster(64)
registerDoParallel(cl) # Use 64 cores for parallel CV
```

Function to perform CV

```
perfom_cv <- function(train, NF=10){

  ### No. of rows in train set
  N<-nrow(train)

  ## Creating NF Folds
  folds<-split(1:N,cut(1:N, quantile(1:N, probs = seq(0, 1, by =1/NF))))

  ## Shuffling
  ridx<-sample(1:nrow(train),nrow(train),replace=FALSE)

  data <- train[ridx,]

  # 'dopar' here would run this on multiple threads (change to just 'do' for synchronous runs)
  outputs <- foreach(idx = folds, .packages = "e1071", .combine = rbind ) %dopar% {

    # Get the fold data where 'idx' is a list of indexes for test observations in the data
    data.train <- data[-idx,] # Get the opposite of the test observations to train on
    data.test <- data[idx,]

    # Fit the model and make predictions
    m <- naiveBayes(readmitted~., data=data.train )   # Fit the model
    p <- predict(m,data.test[,-c(19)], type='raw')
    pc <- unlist(apply(round(p),1,which.max))-1
    pc <- as.factor(pc)
    pred_cfm<-caret::confusionMatrix(pc, data.test$readmitted)

    list(fold=idx,m=m,cfm=pred_cfm) # store the fold, model,cfm
  }

  outputs
}
```

Function to get Evaluation Metric:

```
## Function to get the Evaluation Metric of the results of Cross Validation
get_Eval_Metrix <- function(results) {
  # Results is a list, so you could process it to extract the accuracies like this:

  tstres.perf<-as.data.frame(do.call('rbind',lapply(results[,"cfm"],FUN=function(cfm)c(cfm$overall))))
  tst.overall<-apply(tstres.perf,2,mean)

  ## Class 0: "<30"
  tstres.perf.1<-as.data.frame(do.call('rbind',lapply(results[,"cfm"],FUN=function(cfm)c(cfm$byClass[1,]))))
  cv.tst.perf.1<-apply(tstres.perf.1,2,mean)

  ## Class 1: ">30"
  tstres.perf.2<-as.data.frame(do.call('rbind',lapply(results[,"cfm"],FUN=function(cfm)c(cfm$byClass[2,]))))
  cv.tst.perf.2<-apply(tstres.perf.2,2,mean)

  ## Class 2: "NO"
  tstres.perf.3<-as.data.frame(do.call('rbind',lapply(results[,"cfm"],FUN=function(cfm)c(cfm$byClass[3,]))))
  cv.tst.perf.3<-apply(tstres.perf.3,2,mean)


  df <- as.data.frame(do.call("rbind", list(cv.tst.perf.1, cv.tst.perf.2, cv.tst.perf.3))) %>%
    select(c("Specificity","Precision","Recall","Balanced Accuracy"))

  rownames(df) <- c("Class: <30","Class: >30","Class: NO")

  list(overall=tst.overall, byClass=df)
}
```

Performing CV

```
## Performing 10 Fold Cross Validation in Parallel with 64 Cores
results  <- perfom_cv(train, 10)



get_Eval_Metrix(results)

## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   5.510226e-01   1.249835e-01   5.393832e-01   5.626203e-01   5.391180e-01
## AccuracyPValue  McnemarPValue
##   3.340418e-02  3.126661e-294
##
## $byClass
##            Specificity Precision    Recall Balanced Accuracy
## Class: <30   0.9341373 0.2264230 0.1534932         0.5438152
## Class: >30   0.9205516 0.5065966 0.1519341         0.5362429
## Class: NO    0.2649705 0.5866739 0.8918740         0.5784223
```

Function to get results on Test Data

```
### Function to get the Confusion Metrix on Test Data
get_test_cfm <- function(results, test) {

  test_preds<-lapply(results[,"m"],FUN=function(M,D=test[,-c(19)])predict(M,D,type='raw'))
  test_cfm <- do.call('rbind',lapply(test_preds,FUN=function(P,A=test[[19]]) {
    pred_class<-unlist(apply(round(P),1,which.max))-1
    pred_tbl<-table(pred_class,A)
    pred_cfm<-caret::confusionMatrix(pred_tbl)
    list(cfm=pred_cfm)
  }))

  test_cfm
}

test_cfm <- get_test_cfm(results, test)

get_Eval_Metrix(test_cfm)
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   5.532687e-01   1.315771e-01   5.476710e-01   5.588563e-01   5.391065e-01
## AccuracyPValue  McnemarPValue
##   6.277683e-07   0.000000e+00
##
## $byClass
##            Specificity Precision    Recall Balanced Accuracy
## Class: <30   0.9326316 0.2284803 0.1587148         0.5456732
## Class: >30   0.9191615 0.5096897 0.1565548         0.5378581
## Class: NO    0.2747157 0.5899196 0.8919806         0.5833482
```

Evaluating Test Set

```
test_cfm <- get_test_cfm(results, test)

get_Eval_Metrix(test_cfm)
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   5.532687e-01   1.315771e-01   5.476710e-01   5.588563e-01   5.391065e-01
## AccuracyPValue  McnemarPValue
##   6.277683e-07   0.000000e+00
##
## $byClass
##            Specificity Precision    Recall Balanced Accuracy
## Class: <30   0.9326316 0.2284803 0.1587148         0.5456732
## Class: >30   0.9191615 0.5096897 0.1565548         0.5378581
## Class: NO    0.2747157 0.5899196 0.8919806         0.5833482
```

Results of Cross Validation:

	Cross Validation
Accuracy	0.5532

	Class : <30	Class: >30	Class : NO
Sensitivity (Recall)	0.1587148	0.1565548	0.89198
Precision	0.2284803	0.5096897	0.58992
Specificity	0.9326316	0.9191615	0.27472

The performance is low as compared to Random Forest and Bagging algorithms.

Calculating Variance of Cross Validation :

```
tstres.perf<-as.data.frame(do.call('rbind',lapply(results[,"cfm"],FUN=function(cfm)c(cfm$overall))))
(tst.overall.var<-apply(tstres.perf,2,sd))
##     Accuracy          Kappa  AccuracyLower  AccuracyUpper
##  0.006917018    0.007971722    0.006930605    0.006897766
## AccuracyNull AccuracyPValue  McnemarPValue
##  0.005350887    0.058022248    0.000000000
```

The Variance comes out to be 0.036, which is impressively low as compared to Random Forest and Bagging algorithms. This can be because the model is getting trained on a much larger amount of data here in Cross Validation as compared to other algorithms.

7) Q. What does CV do to bias and variance?

Both bias and variance is reduced by Cross Validation.

In Cross Validation, every data point gets to be in a validation set exactly once, and gets to be in a training set *k-1* times (for k-fold CV). This significantly reduces bias as we are using most of the data for fitting, and also significantly reduces variance as most of the data is also being used in validation set. Interchanging the training and test sets also adds to the effectiveness of this method.

However the training time is significantly increased when performing LOOCV, and generally K-Fold cross validation is preferred.

8) Summary

Algo	Accuracy	Variance
Bagging	0.578	0.0839
Random Forest	0.5791	0.0732
Cross Validation	0.5532	0.036

All the algorithms performed comparatively same. The bagging and cross validation methods take a lot of time in execution. The variance is least for Cross Validation as covers all the data these is, in its training process as well as for testing process. However all the 3 algorithms significantly reduces the variance as compared to Individual Classifiers.

Algorithm	Measure	Class : <30	Class: >30	Class : NO
Bagging	Sensitivity (Recall)	0.436047	0.4903	0.6089
	Precision	0.02201	0.35459	0.8378
	Specificity	0.890239	0.6984	0.6614
Random Forest	Sensitivity (Recall)	0.0041092	0.2939	0.8828
	Precision	0.8235	0.50646	0.5973
	Specificity	0.890239	0.8463	0.3039
Cross Validation	Sensitivity (Recall)	0.1587148	0.15655	0.89198
	Precision	0.2284803	0.50969	0.58992
	Specificity	0.9326316	0.91916	0.27472

Specificity

- The specificity for the "<30" class is high in all the algorithms, this means that all the algorithms are able to easily identify those patients who are not going to come back to the hospital within 30 days after they were discharged.
- However if you see the specificity for the "NO" class, it is low for low accuracy algorithms and becomes higher as the accuracy is increasing. This means that the algorithm is learning to identify the observations which should not be in "NO" class. (Initially, it is trying to classify more observations to "NO" class, just because their ratio is high). Sadly these algorithms are not performing well in this task.

Sensitivity:

- This is the most important measure for the "<30" class, since we need to correctly identify those patients which will be readmitted to the hospital within 30 days. The Bagging algorithms significantly performs better than the other two, and the random forest algorithm is the worst performing algorithm for this measure. However, the value is very low for all the algorithms. This can be because the data is heavily imbalanced and number of observations for the class "<30" are very less, and thus the algorithms are not able to learn this class properly.
- If we see the sensitivity of "NO" class, it is significantly high for all the algorithms, and is highest for Cross Validation algorithm. The algorithms are classifying most of the observations as "NO", without learning much about the other classes.

To conclude, the imbalanced data has taught us that the algorithm should learn to classify correctly the observations which do not belong to class "NO" (class with high proportion) and the observations which should be in class "<30" (class with low proportion).

The Last attempt with Under and Over Sampling:

Under Sampling

```
library(caret)

## Current Sample
table(train$readmitted)
##    0     1     2
## 7949 24881 38404

x <- train %>% select(-readmitted)
y <- train$readmitted

train_down <- downSample(x, y, yname = "readmitted")

table(train_down$readmitted)

##    0    1    2
## 7949 7949 7949
```

1) Training Random Forest (Under Sampling)

```
library(ranger)

model.rf <- ranger(
  formula         = readmitted ~ .,
  data            = train_down,
  num.trees       = 500,
  mtry            = 2,
  min.node.size   = 7,
  sample.fraction = 0.55,
  seed            = 20
)

test.pred <- predict(model.rf, test[sig.attrs])
confusionMatrix(test.pred$predictions, test$readmitted)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0  1472  3362  3075
##          1   750  3044  2816
##          2  1186  4258 10569
##
## Overall Statistics
##
## Accuracy : 0.4941
## 95% CI : (0.4884, 0.4997)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : 1
##
## Kappa : 0.1743
##
## Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity           0.43192   0.2854   0.6421
## Specificity           0.76268   0.8205   0.6131
## Pos Pred Value        0.18612   0.4605   0.6600
## Neg Pred Value        0.91442   0.6815   0.5943
## Prevalence            0.11162   0.3493   0.5391
## Detection Rate        0.04821   0.0997   0.3462
## Detection Prevalence  0.25904   0.2165   0.5245
## Balanced Accuracy     0.59730   0.5530   0.6276
```

2) Training KNN (Under Sampling)

```
train_knn <- train_down
test_knn <- test
train_knn[sig10.cat.attrs] <- lapply(train_knn[sig10.cat.attrs], as.numeric)
test_knn[sig10.cat.attrs] <- lapply(test_knn[sig10.cat.attrs], as.numeric)

knn.model <- knn(train_knn,test_knn,cl = train_knn$readmitted, k = 3)

confusionMatrix(knn.model, test$readmitted)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0  2430  2213   418
##          1   883  6485  3399
##          2    95  1966 12643
##
## Overall Statistics
##
## Accuracy : 0.7061
## 95% CI : (0.7009, 0.7112)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16
##
## Kappa : 0.5091
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity           0.71303   0.6081   0.7681
## Specificity           0.90300   0.7845   0.8535
## Pos Pred Value        0.48014   0.6023   0.8598
## Neg Pred Value        0.96160   0.7886   0.7588
## Prevalence            0.11162   0.3493   0.5391
## Detection Rate        0.07959   0.2124   0.4141
## Detection Prevalence  0.16576   0.3526   0.4816
## Balanced Accuracy     0.80801   0.6963   0.8108
```

Over Sampling

```
library(caret)

## Current Sample
table(train$readmitted)
##    0     1     2
## 7949 24881 38404

x <- train %>% select(-readmitted)
y <- train$readmitted

train_over <- upSample(x, y, yname = "readmitted")

table(train_over$readmitted)

##     0     1     2
## 38404 38404 38404
```

1) Training Random Forest (Over Sampling)

```
model.rf.2 <- ranger(
  formula         = readmitted ~ .,
  data            = train_over,
  num.trees       = 500,
  mtry            = 2,
  min.node.size   = 7,
  sample.fraction = 0.55,
  seed            = 20
)



test.pred.2 <- predict(model.rf.2, test[sig.attrs])
confusionMatrix(test.pred.2$predictions, test$readmitted)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0   989  2071  1971
##          1  1150  4140  3412
##          2  1269  4453 11077
##
## Overall Statistics
##
## Accuracy : 0.5308
## 95% CI : (0.5252, 0.5364)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : 0.9983
##
## Kappa : 0.1985
##
## Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity           0.29020   0.3882   0.6730
## Specificity           0.85098   0.7704   0.5934
## Pos Pred Value        0.19658   0.4758   0.6594
## Neg Pred Value        0.90514   0.7011   0.6080
## Prevalence            0.11162   0.3493   0.5391
## Detection Rate        0.03239   0.1356   0.3628
## Detection Prevalence  0.16478   0.2850   0.5502
## Balanced Accuracy     0.57059   0.5793   0.6332
```

2) Training KNN (Over Sampling)

```
train_knn <- train_over
test_knn <- test
train_knn[sig10.cat.attrs] <- lapply(train_knn[sig10.cat.attrs], as.numeric)
test_knn[sig10.cat.attrs] <- lapply(test_knn[sig10.cat.attrs], as.numeric)


knn.model <- knn(train_knn,test_knn,cl = train_knn$readmitted, k = 3)
```

```
confusionMatrix(knn.model, test$readmitted)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0  2306  1666   190
##          1  1066  7312  3159
##          2    36  1686 13111
##
## Overall Statistics
##
## Accuracy : 0.7444
## 95% CI : (0.7395, 0.7493)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16
##
## Kappa : 0.5675
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity           0.67664   0.6857   0.7965
## Specificity           0.93157   0.7873   0.8776
## Pos Pred Value        0.55406   0.6338   0.8839
## Neg Pred Value        0.95821   0.8235   0.7867
## Prevalence            0.11162   0.3493   0.5391
## Detection Rate        0.07553   0.2395   0.4294
## Detection Prevalence  0.13632   0.3779   0.4858
## Balanced Accuracy     0.80411   0.7365   0.8371
```

Observations:

Method	Number of Observation Per Class	Algorithm	Accuracy	Recall ("<30" Class)
Under Sampling	7949	Random Forest	0.4941	0.43192
		KNN (k=3)	0.7061	0.713
Over Sampling	38404	Random Forest	0.5308	0.2902
		KNN (k=3)	0.7444	0.6766

Even the Under and Over Sampling technique is not helping with the accuracy. However the Recall of "<30" class is significantly improved by these method along with KNN. Earlier it was 0.4 with KNN and now its near to 0.7 which is a significant improvement.

This increase in Recall can be help full in identifying the patients who have a high chances of returning back to the hospital within 30 days, and hence they can be monitored more closely with their health.