Working with Individual Classifiers

Sharath Katta Sridhar

sk8671@nyu.edu

Table of Contents

1) Loading Libraries:

```
library(tidyverse)
library(RColorBrewer)
library(dlookr)
library(ggcorrplot)
library(plyr)
library(dplyr)
library(cowplot)
library(rsample)
library(rpart)
library(nnet)
library(ggplot2)
library(caret)
library(party)
library(pROC)
library(ROCR)
library(class)
```

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

Following Operations were performed in Pre-processing:

- The attributes with only 1 unique value were removed.
- The attributes with 99% of the values same were removed.
- Unknow/Invalid was left as a separate class instead of imputing it with mean/mode.
  This reduces the pre-processing time and it's good to follow this approach when the such values are few in number.
- Attributes with more than 25% of the null values were removed.
- Age was converted to ordered factor based on the ranges provided
- All other categorical attributes were converted into factors.
- Two numerical attributes were transformed using square root function in order make them Normally Distributed.
  However, most of the numerical variables were left not-normal, as none of the transformation can make them normally distributed.
- There were many outliers which cannot be simply removed from the dataset since this might reduce the variability on data points hence making it difficult for models to identify any patterns.
  Thus, outliers were left as is.
- Resampling was not done in this assignment to balance the imbalance data, will try to do it in the next assignment.

We have 3 Target Class that we need to predict.

Readmitted	Whether the patient is readmitted after discharge from the hospital
<30	Patient readmitted within 30 days of discharge
>30	Patient readmitted after 30 days of discharge
NO	Patient does not get readmitted

```
###############################################################################
## 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)
```

Following features were dropped

Seq	Features of Medication	Keep	Comment
1	examide	No	Only one value = "No"
2	citoglipton	No	Only one value = "No"
3	glimepiride.pioglitazone	No	99% value = "No"
4	metformin.rosiglitazone	No	99% value = "No"
5	metformin.pioglitazone	No	99% value = "No"
6	acetohexamide	No	99% value = "No"

Seq	Attr	Type	Preprocessing	Keep	Comment
1	race	Nominal	Convert to Factor	Yes	Leaving ? as a separate class
2	gender	Nominal	Convert to Factor	Yes	Leaving ? as a separate class
3	age	Ordinal	Convert to Ordered Factor	Yes	
4	admission_type_id	Nominal	Convert to Factor	Yes	
5	discharge_disposition_id	Nominal	Convert to Factor	Yes	
6	admission_source_id	Nominal	Convert to Factor	Yes	
7	payer_code	Nominal		No	Too many unknown
8	medical_specialty	Nominal		No	Too many unknown
9	max_glu_serum	Ordinal		No	Too many unknown
10	A1Cresult	Ordinal		No	Too many unknown
11	change	Nominal	Convert to Factor	Yes	
12	diabetesMed	Nominal	Convert to Factor	Yes	
13	Features of Medication	Nominal	Convert to Factor	Yes	

Seq	Attr	Type	Preprocessing	Keep	Comment
1	weight	Numeric		No	Too many unknown
2	time_in_hospital	Numeric	Not Normal Distribution	Yes	
3	num_lab_procedures	Numeric	Sqrt Transform can be used | Many Outliers	Yes	
4	num_procedures	Numeric	Not Normal Distribution	Yes	
5	num_medications	Numeric	Sqrt Transform can be used | Many Outliers	Yes	
6	number_outpatient	Numeric	Not Normal Distribution | Many Outliers	Yes	
7	number_emergency	Numeric	Not Normal Distribution | Many Outliers	Yes	
8	number_inpatient	Numeric	Not Normal Distribution | Many Outliers	Yes	
9	number_diagnoses	Numeric	Not Normal Distribution | Many Outliers	Yes	

Following Helper function is used to pre-process the data

```
################################################################################
## 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)
```

3) Helper Functions for Variable Importance:

Following is the helper function to perform Chi-Squared Test

```
### Function to Perform Chi-Square Test
perform_chisq <- function(df, attr_list, target, significance_threshold = 0.05) {
  attr_names <- c()
  X_sqr_stat <- c()
  p_value <- c()
  significant <- c()
  for(attrs in attr_list) {
    c_test <- chisq.test(table(df[,target], df[,attrs]))
    attr_names <- append(attr_names, attrs)
    X_sqr_stat <- append(X_sqr_stat, c_test$statistic)
    p_value <- append(p_value, c_test$p.value)
    significant <- append(significant, (c_test$p.value < significance_threshold))
  }
```

Following is the helper function to perform Anova Test

```
## Function Calculate One Way Anova Test Results
perform_anova <- function(df, attr_list, target, significance_threshold = 0.05) {

  attr_names <- c()
  F_stat <- c()
  p_value <- c()
  significant <- c()

  for(attrs in attr_list)
  {
    anova_test <- aov(df[,attrs]~df[,target], data = df)

    p <- summary(anova_test)[[1]][["Pr(>F)"]][1]
    f <- summary(anova_test)[[1]][["F value"]][1]

    attr_names <- append(attr_names, attrs)
    F_stat <- append(F_stat, f)
    p_value <- append(p_value, p)
    significant <- append(significant, (p < significance_threshold))
  }

  df <- data.frame(attr_names,F_stat,p_value,significant)

  arrange(df, p_value)
}
```

4) Variable Importance based on EDA:

Chi-Squared Test for Categorical Features:

```
###############################################################################
### Variable Importance

## 1. Using chi-squared test to select categorical attributes
chisq_df <- perform_chisq(diabetes.processed, c(cat.features,medical.features), "readmitted",
significance_threshold = 0.05)
chisq_df

##                   attr_names  X_sqr_stat       p_value significant
## 1  discharge_disposition_id 3587.291304  0.000000e+00        TRUE
## 2       admission_source_id 1150.971841 2.317985e-221        TRUE
## 3                   insulin  516.695761 2.126586e-108        TRUE
## 4               diabetesMed  386.510884  1.175514e-84        TRUE
## 5         admission_type_id  415.760978  6.037493e-80        TRUE
## 6                       age  313.171753  9.348415e-56        TRUE
## 7                      race  282.594801  7.379469e-55        TRUE
## 8                    change  215.825001  1.362061e-47        TRUE
## 9                 metformin  104.841777  2.445917e-20        TRUE
## 10              repaglinide   58.964857  7.302647e-11        TRUE
## 11                glipizide   54.255694  6.551146e-10        TRUE
## 12            rosiglitazone   43.008602  1.161873e-07        TRUE
## 13                   gender   37.461170  1.447272e-07        TRUE
## 14                 acarbose   35.683669  3.175614e-06        TRUE
## 15             pioglitazone   29.935808  4.042850e-05        TRUE
## 16              glimepiride   16.654439  1.064076e-02        TRUE
## 17                 miglitol   11.594220  7.165816e-02       FALSE
## 18                glyburide    9.993778  1.249143e-01       FALSE
## 19           chlorpropamide    8.955602  1.760904e-01       FALSE
## 20      glyburide.metformin    8.524489  2.021388e-01       FALSE
## 21                tolazamide    5.086302  2.785564e-01       FALSE
## 22      glipizide.metformin    2.047992  3.591569e-01       FALSE
## 23                tolbutamide    1.634978  4.415389e-01       FALSE
## 24              troglitazone    1.435693  4.878016e-01       FALSE
## 25              nateglinide    3.423678  7.540948e-01       FALSE

### Extracting top 10 significant categorical features
sig.cat.attrs <- chisq_df[chisq_df$significant=="TRUE",]$attr_names
sig10.cat.attrs <- head(sig.cat.attrs, 10)
sig10.cat.attrs

## [1] "discharge_disposition_id" "admission_source_id"
## [3] "insulin"                  "diabetesMed"
## [5] "admission_type_id"        "age"
## [7] "race"                     "change"
## [9] "metformin"                "repaglinide"
```

Stored top 10 attributes in sig10.cat.attrs vector.

Anova Test for Numerical Features:

```
## 2. Using Anova test to select numerical attributes
anova_df <- perform_anova(diabetes.processed, numeric.features, "readmitted", significance_threshold =
0.05)
anova_df
##           attr_names     F_stat       p_value significant
## 1   number_inpatient 2963.32384  0.000000e+00        TRUE
## 2   number_diagnoses  655.46495 1.422802e-283        TRUE
## 3   number_emergency  573.25719 2.688984e-248        TRUE
## 4  number_outpatient  355.23269 1.821591e-154        TRUE
## 5    num_medications  213.42809  3.185800e-93        TRUE
## 6   time_in_hospital  170.33089  1.411815e-74        TRUE
## 7     num_procedures  103.54127  1.197541e-45        TRUE
## 8 num_lab_procedures   74.59916  4.224000e-33        TRUE

### Extracting all significant  numeric features
sig.numeric.attrs <- anova_df[anova_df$significant=="TRUE",]$attr_names
sig.numeric.attrs

## [1] "number_inpatient"   "number_diagnoses"   "number_emergency"
## [4] "number_outpatient"  "num_medications"    "time_in_hospital"
## [7] "num_procedures"     "num_lab_procedures"

sig.attrs <- c(sig10.cat.attrs,sig.numeric.attrs)
```

5) Train Test Split:

Making a 70/30 split for Training and Testing data.

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

str(diabetes.imp)
## 'data.frame':       101766 obs. of  19 variables:
## $ discharge_disposition_id: Factor w/ 26 levels "1","2","3","4",..: 24 1 1 1 1 1 1 1 1 3 ...
## $ admission_source_id     : Factor w/ 17 levels "1","2","3","4",..: 1 7 7 7 7 2 2 7 4 4 ...
## $ insulin                 : Factor w/ 4 levels "Down","No","Steady",..: 2 4 2 4 3 3 3 2 3 3 ...
## $ diabetesMed             : Factor w/ 2 levels "No","Yes": 1 2 2 2 2 2 2 2 2 2 ...
## $ admission_type_id       : Factor w/ 8 levels "1","2","3","4",..: 6 1 1 1 1 2 3 1 2 3 ...
## $ age                     : Ord.factor w/ 10 levels "[0-10)"<"[10-20)"<..: 1 2 3 4 5 6 7 8 9 10 ...
## $ race                    : Factor w/ 6 levels "?","AfricanAmerican",..: 4 4 2 4 4 4 4 4 4 4 ...
## $ change                  : Factor w/ 2 levels "Ch","No": 2 1 2 1 1 2 1 2 1 1 ...
## $ metformin               : Factor w/ 4 levels "Down","No","Steady",..: 2 2 2 2 2 2 3 2 2 2 ...
## $ repaglinide             : Factor w/ 4 levels "Down","No","Steady",..: 2 2 2 2 2 2 2 2 2 2 ...
## $ number_inpatient        : int  0 0 1 0 0 0 0 0 0 0 ...
## $ number_diagnoses        : int  1 9 6 7 5 9 7 8 8 8 ...
## $ number_emergency        : int  0 0 0 0 0 0 0 0 0 0 ...
## $ number_outpatient       : int  0 0 2 0 0 0 0 0 0 0 ...
## $ num_medications         : num  1 4.24 3.61 4 2.83 ...
## $ time_in_hospital        : int  1 3 2 2 1 3 4 5 13 12 ...
## $ num_procedures          : int  0 0 5 1 0 6 1 0 2 3 ...
## $ num_lab_procedures      : num  6.4 7.68 3.32 6.63 7.14 ...
## $ readmitted              : Factor w/ 3 levels "0","1","2": 3 2 3 3 3 2 3 2 3 3 ...

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
```

6) Helper Functions for calculating variance of a model:

```
### 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
}
```

7) Multinomial Logistic Regression:

```
### Multinomial Logistic Regression (using Significant Features)

multinom.model <- multinom(readmitted ~ . , data=train)

summary(multinom.model)
## Residual Deviance: 125146.1
## AIC: 125474.1

### train Results
train.fitted.results <- predict(multinom.model,select(train,all_of(sig.attrs) ))

confusionMatrix(train.fitted.results, train$readmitted)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0   170   165    97
##          1  2603  7420  4475
##          2  5176 17296 33832
##
## Overall Statistics
##
## Accuracy : 0.5815
## 95% CI : (0.5779, 0.5851)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16
##
## Kappa : 0.1665
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity          0.021386   0.2982   0.8809
## Specificity          0.995860   0.8473   0.3155
## Pos Pred Value       0.393519   0.5118   0.6009
## Neg Pred Value       0.890130   0.6922   0.6938
## Prevalence           0.111590   0.3493   0.5391
## Detection Rate       0.002387   0.1042   0.4749
## Detection Prevalence 0.006065   0.2035   0.7904
## Balanced Accuracy    0.508623   0.5728   0.5982

### test Results
test.fitted.results <- predict(multinom.model,select(test,all_of(sig.attrs)))

confusionMatrix(test.fitted.results, test$readmitted)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0    82    75    43
##          1  1110  3191  1935
##          2  2216  7398 14482
##
## Overall Statistics
##
## Accuracy : 0.5815
## 95% CI : (0.576, 0.5871)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16
##
## Kappa : 0.1672
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##                      Class: 0 Class: 1 Class: 2
## Sensitivity          0.024061   0.2992   0.8798
## Specificity          0.995650   0.8467   0.3168
## Pos Pred Value       0.410000   0.5117   0.6010
## Neg Pred Value       0.890347   0.6924   0.6927
## Prevalence           0.111621   0.3493   0.5391
## Detection Rate       0.002686   0.1045   0.4743
## Detection Prevalence 0.006551   0.2042   0.7892
## Balanced Accuracy    0.509855   0.5730   0.5983
```

Calculating AUC:

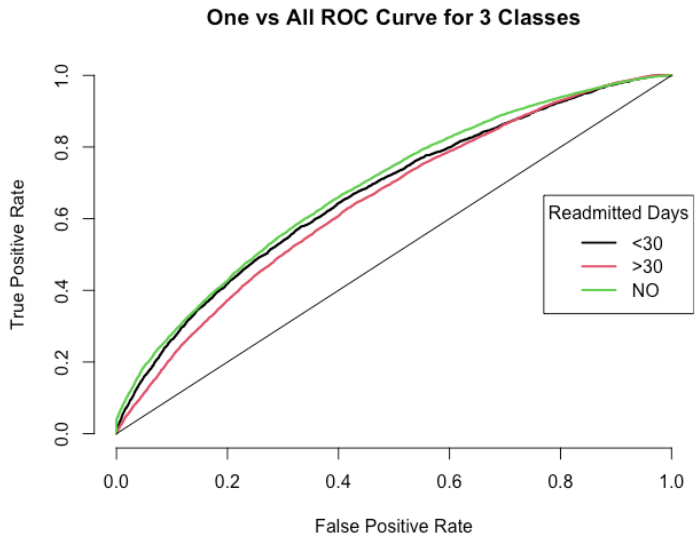
```
test.results.probs <- predict(multinom.model,select(test,all_of(sig.attrs)), type='probs')

library(pROC)
roc.multi <- multiclass.roc(test$readmitted, test.results.probs)
auc(roc.multi)
## Multi-class area under the curve: 0.6468
```

Plotting ROC:

Following Helper Function is used to plot ROC in this multi-class classification:

```
### Function to Plot ROC Curve for Multiclass
multiclass_roc_plot <- function(df, probs) {

  class.0.probs <- probs[,1]
  class.1.probs <- probs[,2]
  class.2.probs <- probs[,3]

  actual.0.class <- as.integer(df$readmitted == "0")
  actual.1.class <- as.integer(df$readmitted == "1")
  actual.2.class <- as.integer(df$readmitted == "2")


  plot(x=NA, y=NA, xlim=c(0,1), ylim=c(0,1),
       ylab='True Positive Rate',
       xlab='False Positive Rate',
       bty='n')

  legend(x = "right",                    # Position
         title = "Readmitted Days",
         legend = c("<30", ">30","NO"),  # Legend texts
         col = c(1, 2, 3),               # Line colors
         lwd = 2)                        # Line width

  title("One vs All ROC Curve for 3 Classes")

  pred.0 = prediction(class.0.probs, actual.0.class)
  nbperf.0 = performance(pred.0, "tpr", "fpr")

  roc.x = unlist(nbperf.0@x.values)
  roc.y = unlist(nbperf.0@y.values)
  lines(roc.y ~ roc.x, col=0+1, lwd=2)


  pred.1 = prediction(class.1.probs, actual.1.class)
  nbperf.1 = performance(pred.1, "tpr", "fpr")

  roc.x = unlist(nbperf.1@x.values)
  roc.y = unlist(nbperf.1@y.values)
  lines(roc.y ~ roc.x, col=1+1, lwd=2)

  pred.2 = prediction(class.2.probs, actual.2.class)
  nbperf.2 = performance(pred.2, "tpr", "fpr")

  roc.x = unlist(nbperf.2@x.values)
  roc.y = unlist(nbperf.2@y.values)
  lines(roc.y ~ roc.x, col=2+1, lwd=2)

  lines(x=c(0,1), c(0,1))

}
```

```
## ROC Curve
multiclass_roc_plot(test, test.results.probs)

##Function on next page
```



Observations Logistic Regression:

Accuracy	0.5815
AUC	0.6468

	Class : <30	**Class : >30**	**Class : NO**
Sensitivity (Recall)	0.024061	0.2992	0.8798
Precision	0.41000	0.5117	0.6010
Specificity	0.99565	0.8467	0.3168

The model is underfitting and is not able to generate better predictions.
One of the reason is that the classes are imbalanced, the "<30" class has nearly 10% observation in our dataset.

Other problem here is that many of the attributes are not normally distributed, as we saw during the EDA. Also we were able to transform only 2 attributes to normal distribution and the remaining are not normal.

Another major problem observed was the number of outliers in the numerical attributes. These outliers were so large in number that they cannot be removed.

High **recall** for "NO" class shows that 87% of the observations of "NO" class were predicted correctly. The recall value is 2% and 30% for "<30" and ">30" class, which shows that the model was not able to extract all the observations of these classes correctly.

41% **Precision** of "<30" class show that only 41% of the total observations predicted to be this class are actually of "<30" class, and rest 69% are wrongly predicted as "<30" class.

ROC curve also shows that the model is not performing, and AUC for all the 3 class is low and hence the average AUC i.e. 0.64, is low.

Calculating Variance of algorithm:

```
### Calculating the Variance of Multinomial Logistic Regression
## Training with Train Set 1
multinom.model.v1 <- multinom(readmitted ~ . , data=train_v1)
test.results.v1 <- predict(multinom.model.v1,select(test_v,all_of(sig.attrs)))

## Training with Train Set 2
multinom.model.v2 <- multinom(readmitted ~ . , data=train_v2)
test.results.v2 <- predict(multinom.model.v2,select(test_v,all_of(sig.attrs)))

cal_variance(test.results.v1,test.results.v2)
## [1] 0.04077422
```

The variance is quite low (0.04) which is a good sign for Logistic Regression model.

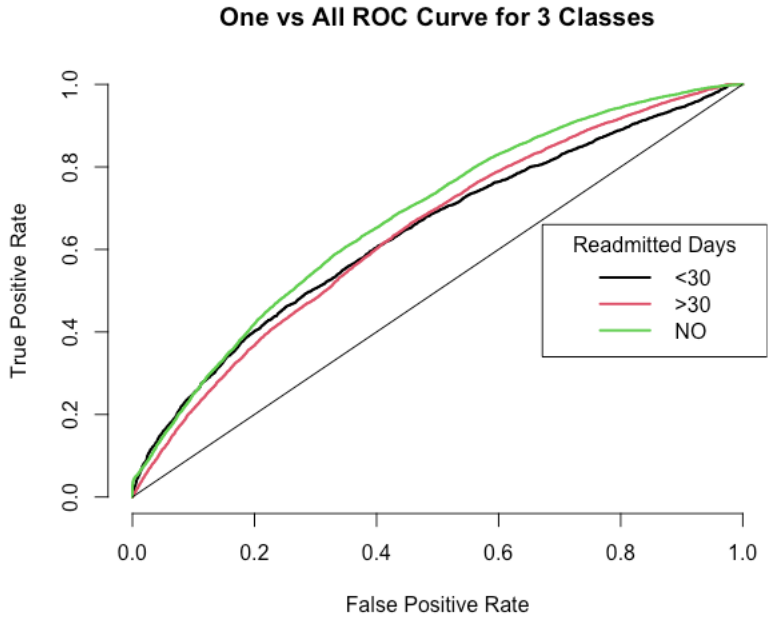
The Bias calculation is not possible.

8) Support Vector Machine:

```
################################################################################
### 2. SVM

library(e1071)        # SVM methodology

svmfit <- svm(readmitted~., data = train, kernel = "radial", probability = TRUE)



# Evaluating the Train set
pred.svm.train <- predict(svmfit, train[sig.attrs])

confusionMatrix(pred.svm.train, train$readmitted)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0    60    33    32
##          1  2482  6398  3615
##          2  5407 18450 34757
##
## Overall Statistics
##
## Accuracy : 0.5786
## 95% CI : (0.5749, 0.5822)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16
##
## Kappa : 0.1485
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                       Class: 0 Class: 1 Class: 2
## Sensitivity          0.0075481  0.25714   0.9050
## Specificity          0.9989729  0.86847   0.2733
## Pos Pred Value       0.4800000  0.51204   0.5930
## Neg Pred Value       0.8890576  0.68534   0.7110
## Prevalence           0.1115900  0.34929   0.5391
## Detection Rate       0.0008423  0.08982   0.4879
## Detection Prevalence 0.0017548  0.17541   0.8228
## Balanced Accuracy    0.5032605  0.56280   0.5892
```

```
# Evaluating the Test set
pred.svm.test <- predict(svmfit, test[sig.attrs])

confusionMatrix(pred.svm.test, test$readmitted)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0    18    17    12
##          1  1093  2795  1509
##          2  2297  7852 14939
##
## Overall Statistics
##
## Accuracy : 0.5814
## 95% CI : (0.5759, 0.587)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16

## Kappa : 0.1546
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                       Class: 0 Class: 1 Class: 2
## Sensitivity          0.0052817  0.26210   0.9076
## Specificity          0.9989308  0.86904   0.2788
## Pos Pred Value       0.3829787  0.51788   0.5955
## Neg Pred Value       0.8887978  0.68693   0.7206
## Prevalence           0.1116206  0.34927   0.5391
## Detection Rate       0.0005895  0.09154   0.4893
## Detection Prevalence 0.0015394  0.17677   0.8217
## Balanced Accuracy    0.5021063  0.56557   0.5932
```

```
### Calculating AUC
test.svm.probs <- predict(svmfit,select(test,all_of(sig.attrs)), probability = TRUE)
test.svm.probabs <- attr(test.svm.probs, "probabilities")

roc.multi <- multiclass.roc(test$readmitted, test.svm.probabs)
auc(roc.multi)
## Multi-class area under the curve: 0.6295

## Plotting ROC Curve
multiclass_roc_plot(test, test.svm.probabs)
```



Observations SVM:

Accuracy	0.5814
AUC	0.6295

	Class : <30	**Class: >30**	**Class : NO**
Sensitivity (Recall)	0.005282	0.2621	0.9076
Precision	0.38297	0.51788	0.59546
Specificity	0.998931	0.86904	0.2788

The model is underfitting and is not able to generate better predictions.
One of the reason is that the **size of data** is **huge** for SVM. It took a **huge amount of time** to train this model and was probably not an efficient algorithm to use for this dataset.

Also SVMs are not recommended for imbalanced datasets and our dataset is heavily imbalanced. This is one of the reason for poor performance.

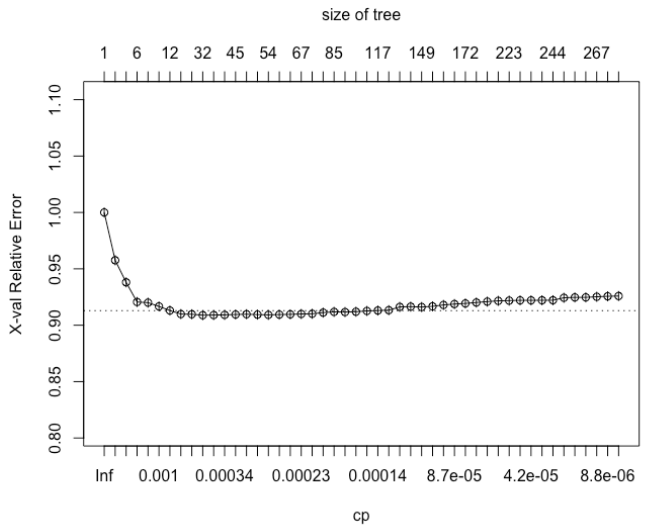
Note:
I am not calculating variance for SVM model, as it is running very in-efficiently on this data set and taking a lot of time, around 2 hrs to train.

The Bias calculation is not possible.

9) Decision Tree Algorithm:

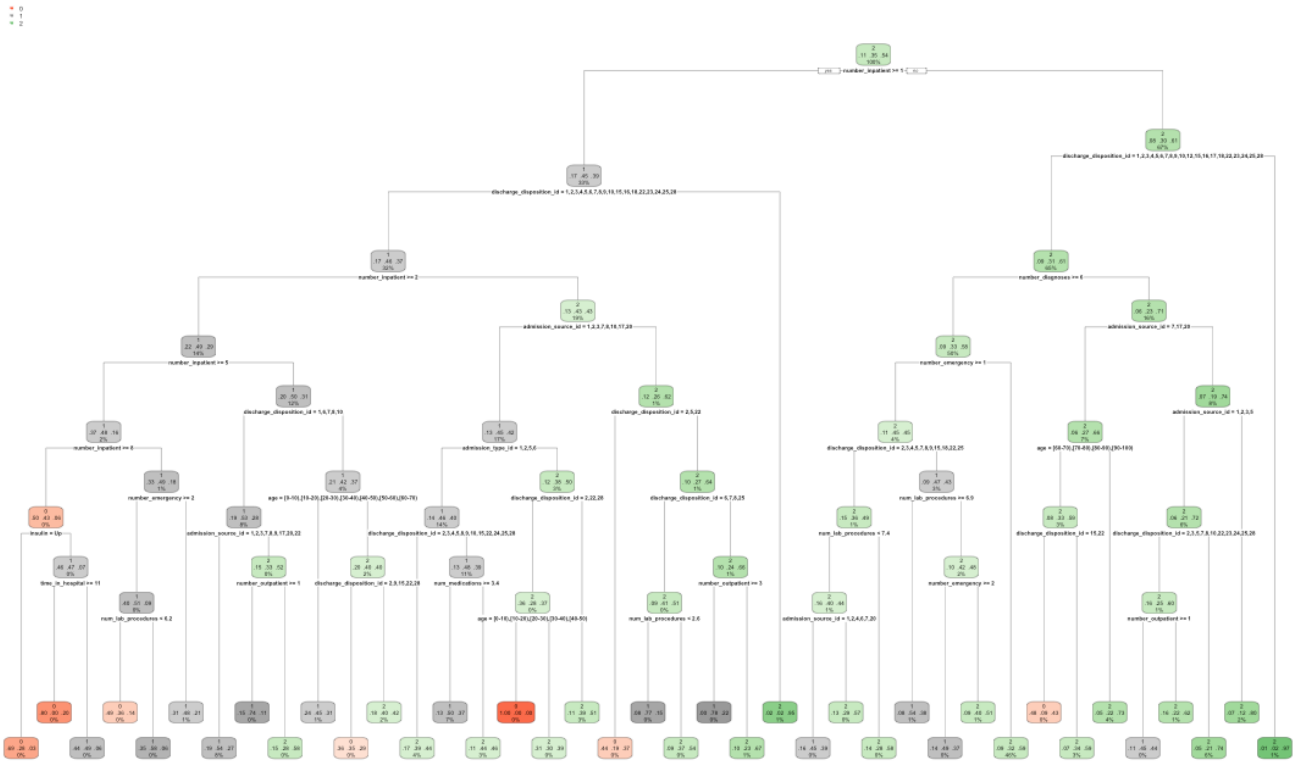
```
###############################################################################
### 3. Decision Tree

dtree.model <- rpart(readmitted ~ ., data = train,method = "class",
                     control = list(minsplit = 100,maxdepth = 7, xval = 10,cp = 0))
## xval = number of cross validations

## Decission Tree Plot
rpart.plot(dtree.model)

## Deciding on Max Depth
plotcp(dtree.model)
```

cp – plot: (complexity parameter)



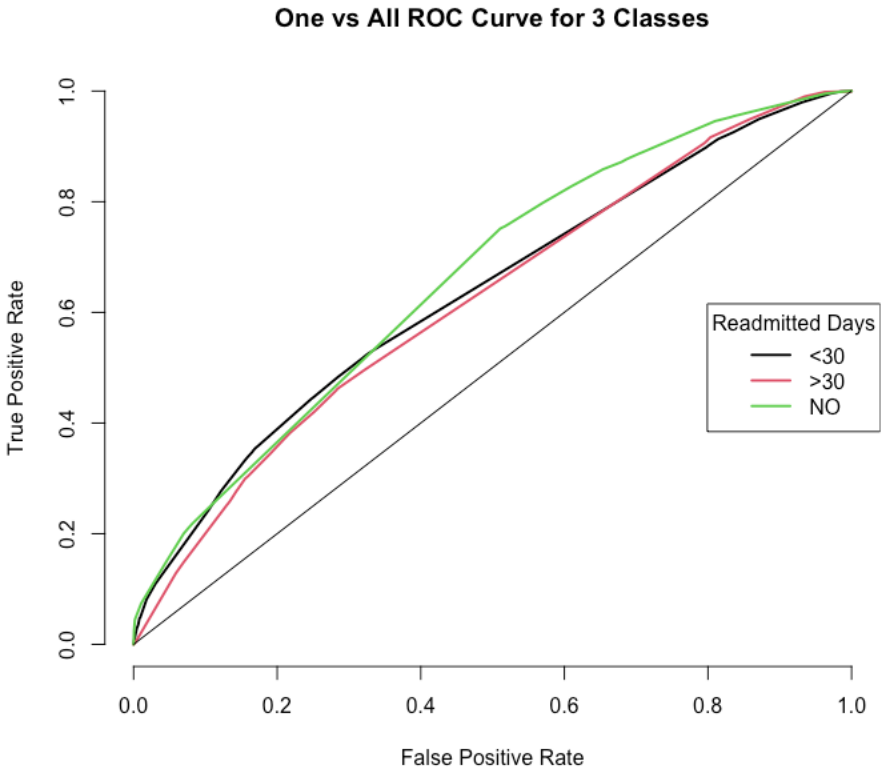
The plot shows that the error significantly reduced till the tree of size around 7, and then became nearly constant.
So we choose the maxdepth hyperparameter equal to 7 in our model.

Decision Tree:



```
### train Results
train.dtree.results <- predict(dtree.model,select(train,all_of(sig.attrs)), type="class")

confusionMatrix(train.dtree.results, train$readmitted)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0   224   138   121
##          1  2612  7581  4568
##          2  5113 17162 33715
##
## Overall Statistics
##
## Accuracy : 0.5829
## 95% CI : (0.5792, 0.5865)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16
##
## Kappa : 0.1709
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity          0.028180   0.3047   0.8779
## Specificity          0.995907   0.8451   0.3215
## Pos Pred Value       0.463768   0.5136   0.6022
## Neg Pred Value       0.890814   0.6937   0.6924
## Prevalence           0.111590   0.3493   0.5391
## Detection Rate       0.003145   0.1064   0.4733
## Detection Prevalence 0.006780   0.2072   0.7860
## Balanced Accuracy    0.512044   0.5749   0.5997
```

```
### test Results
test.dtree.results <- predict(dtree.model,select(test,all_of(sig.attrs)), type="class")

confusionMatrix(test.dtree.results, test$readmitted)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0    83    78    49
##          1  1104  3240  2061
##          2  2221  7346 14350
##
## Overall Statistics
##
## Accuracy : 0.5788
## 95% CI : (0.5733, 0.5844)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16
##
## Kappa : 0.1638
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity          0.024354   0.3038   0.8718
## Specificity          0.995318   0.8407   0.3201
## Pos Pred Value       0.395238   0.5059   0.6000
## Neg Pred Value       0.890344   0.6923   0.6810
## Prevalence           0.111621   0.3493   0.5391
## Detection Rate       0.002718   0.1061   0.4700
## Detection Prevalence 0.006878   0.2098   0.7833
## Balanced Accuracy    0.509836   0.5723   0.5960

### Calculating AUC
test.dtree.probs <- predict(dtree.model,select(test,all_of(sig.attrs)), type="prob")

roc.multi <- multiclass.roc(test$readmitted, test.dtree.probs)
auc(roc.multi)
## Multi-class area under the curve: 0.6189

## Plotting ROC Curve
multiclass_roc_plot(test, test.dtree.probs)
```



Observations Decision Tree:

Accuracy	0.5844
AUC	0.6189

	Class : <30	Class: >30	Class : NO
Sensitivity (Recall)	0.02435	0.3038	0.8718
Precision	0.395238	0.50585	0.5999
Specificity	0.995318	0.8407	0.3201

Some of the reasons why decision tree algorithm is not performing well can be:

- A small change in the data can cause a large change in the structure of the decision tree causing instability.
- For a Decision tree sometimes calculation can go far more complex compared to other algorithms.
- Decision tree often involves higher time to train the model.

The decision tree plot generated is so complex, that it becomes impossible to interpret why the model is performing like this. However we can see that sensitivity of Decision Tree is 2% which is quite better as compared to SVM which had 0.5% sensitivity for "<30" class.

The ROC plot clearly justifies the low average AUC score of 0.6189, which the average of One-vs-All AUC for the 3 classes.

Calculating Variance of algorithm:

```
### Calculating the Variance of Decision Tree Algorithm
## Training with Train Set 1
dtree.model.v1 <- rpart(readmitted ~ ., data = train_v1,method = "class",
                        control = list(minsplit = 100,maxdepth = 7, xval = 10,cp = 0))
test.results.v1 <- predict(dtree.model.v1,select(test_v,all_of(sig.attrs)), type="class")

## Training with Train Set 2
dtree.model.v2 <- rpart(readmitted ~ ., data = train_v2,method = "class",
                        control = list(minsplit = 100,maxdepth = 7, xval = 10,cp = 0))
test.results.v2 <- predict(dtree.model.v2,select(test_v,all_of(sig.attrs)), type="class")

cal_variance(test.results.v1,test.results.v2)
## [1] 0.1804873
```

The variance for decision tree algorithm is found out to be 0.18 , which is high as compared to logistic regression algorithm (0.04). This shows that Decision Tree algorithm gives different results when the training population is different.

The Bias calculation is not possible.

10) Nearest Neighbour Classifier:

Transforming the factors to numeric value before using into KNN.

```
################################################################################
### 4. Nearest Neighbour (knn)

library(class)

train_knn <- train
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)
```

Applying KNN with k = 3 (clusters)

```
## KNN with k=3
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  1344   472    30
##          1  1913  7408  1575
##          2   151  2784 14855
##
## Overall Statistics
##
## Accuracy : 0.7732
## 95% CI : (0.7684, 0.7779)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16
##
## Kappa : 0.591
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##                      Class: 0 Class: 1 Class: 2
## Sensitivity           0.39437   0.6947   0.9025
## Specificity           0.98149   0.8244   0.7914
## Pos Pred Value        0.72806   0.6799   0.8350
## Neg Pred Value        0.92805   0.8342   0.8740
## Prevalence            0.11162   0.3493   0.5391
## Detection Rate        0.04402   0.2426   0.4865
## Detection Prevalence  0.06046   0.3569   0.5827
## Balanced Accuracy     0.68793   0.7596   0.8470
```

Unfortunately, this algorithm does not return us the probabilities for individual class. Hence we will not be able to calculate the AUC score and plot the ROC curve.

Observations KNN (k=3):

Accuracy	0.7732
AUC	Probabilities not available

	Class : <30	**Class: >30**	**Class : NO**
Sensitivity (Recall)	0.39437	0.6947	0.9025
Precision	0.728	0.6798	0.835
Specificity	0.98149	0.8244	0.7914

Applying KNN with k = 5 (clusters)

```
## KNN with k=5
knn.model.2 <- knn(train_knn,test_knn,cl = train_knn$readmitted, k = 5)

confusionMatrix(knn.model.2, test$readmitted)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2
##          0  1172   304    19
##          1  2072  7570  1285
##          2   164  2790 15156
##
## Overall Statistics
##
## Accuracy : 0.7827
## 95% CI : (0.7781, 0.7873)
## No Information Rate : 0.5391
## P-Value [Acc > NIR] : < 2.2e-16
##
## Kappa : 0.6048
##
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##                      Class: 0 Class: 1 Class: 2
## Sensitivity           0.34390   0.7099   0.9208
## Specificity           0.98809   0.8310   0.7901
## Pos Pred Value        0.78395   0.6928   0.8369
## Neg Pred Value        0.92299   0.8422   0.8950
## Prevalence            0.11162   0.3493   0.5391
## Detection Rate        0.03839   0.2479   0.4964
## Detection Prevalence  0.04897   0.3579   0.5931
## Balanced Accuracy     0.66599   0.7704   0.8554
```

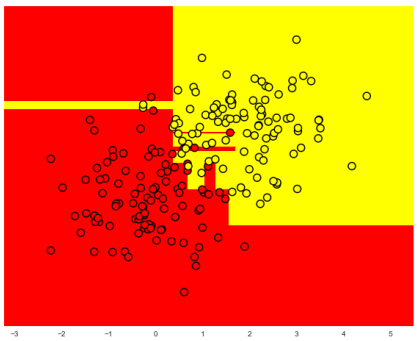
Observations KNN (k=5):

Accuracy	0.7827
AUC	Probabilities not available

	Class : <30	**Class: >30**	**Class : NO**
Sensitivity (Recall)	0.3439	0.7099	0.9208
Precision	0.7839	0.6927	0.8368
Specificity	0.98809	0.8310	0.7901

The KNN-based classifier, however, does not build any classification model. It directly learns from the training instances (observations). It starts processing data only after it is given a test observation to classify. Thus, KNN comes under the category of **"Lazy Learner"** approaches.

Decision trees try to separate the classes using rectangles in a 2D space as shown below.



Because not every time we can split the data points into rectangular sections, decision trees will fail in those cases.

KNN however tries to create clusters and then assign them the classes, which gives us better results.

When comparing the KNN for k=3 vs k=5 , we can see that recall is getting reduced for "<30" class with k=5, which is not a good sign. Since the class "<30" represents the people who got **readmitted** to the hospital within 30 days. Thus we cannot risk these people in the sake of correctly classifying "NO" class, which are already a safe group of people.
Hence **k=3**, should be a better choice.

Calculating Variance of algorithm:

```
### Calculating the Variance of Decision Tree Algorithm

# Preparing Data
train_knn_v1 <- train_v1
train_knn_v2 <- train_v2
test_knn_v <- test_v
train_knn_v1[sig10.cat.attrs] <- lapply(train_knn_v1[sig10.cat.attrs], as.numeric)
train_knn_v2[sig10.cat.attrs] <- lapply(train_knn_v2[sig10.cat.attrs], as.numeric)
test_knn_v[sig10.cat.attrs] <- lapply(test_knn_v[sig10.cat.attrs], as.numeric)

## Training with Train Set 1
predictions.knn.model.v1 <- knn(train_knn_v1,test_knn_v,cl = train_knn_v1$readmitted, k = 3)

## Training with Train Set 2
predictions.knn.model.v2 <- knn(train_knn_v2,test_knn_v,cl = train_knn_v2$readmitted, k = 3)

cal_variance(predictions.knn.model.v1,predictions.knn.model.v2)
## [1] 0.282472
```

The variance for nearest neighbour algorithm is found out to be 0.28 , which is very high as compared to logistic regression algorithm (0.04). This shows that Nearest Neighbour algorithm is very sensitive to population with which it is trained and can give very different results when trained with two different samples.

The Bias calculation is not possible.

11) Summary:

Algo	Accuracy	AUC
Logistic Reg	0.5815	0.6468
SVM	0.5814	0.6295
Decision Tree	0.5844	0.6189
KNN (k=3)	0.7732	Probabilities not available
KNN (k=5)	0.7827	Probabilities not available

Clearly the Nearest Neighbour algorithms have performed the best for this dataset. The Decision tree algorithms should also have performed better since we have a large number of categorical attributes.

Algorithm	Measure	Class : <30	Class: >30	Class : NO
Logistic Regression	Sensitivity (Recall)	0.024061	0.2992	0.8798
	Precision	0.41	0.5117	0.601
	Specificity	0.99565	0.8467	0.3168
SVM	Sensitivity (Recall)	0.0052817	0.2621	0.9076
	Precision	0.38297	0.51788	0.59546
	Specificity	0.9989308	0.86904	0.2788
Decision Tree	Sensitivity (Recall)	0.02435	0.3038	0.8718
	Precision	0.395238	0.50585	0.5999
	Specificity	0.995318	0.8407	0.3201
KNN (k=3)	Sensitivity (Recall)	0.39437	0.6947	0.9025
	Precision	0.728	0.6798	0.835
	Specificity	0.98149	0.8244	0.7914
KNN (k=5)	Sensitivity (Recall)	0.3439	0.7099	0.9208
	Precision	0.7839	0.6927	0.8368
	Specificity	0.98809	0.831	0.7901

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)

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. 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, since they is classifying most of the observations as "NO", which learning much from the data.

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).