ISYE 7406 Final Project: Water Potability Prediction

Group 165: Zakaria Elahmadi
April 2023

1 Introduction

Access to safe drinking water is critical to the health of individuals and broader communities. Water contamination is a prevalent challenge in this industrial era with the increased use of heavy metals, chemicals (i.e., pesticides) and germs. All these components easily contaminate our water sources leading to additional challenges (such as public health problems).

Testing for water potability (i.e., whether water quality is safe for human consumption) is an important and critical element for the health of communities. Regions with water contamination have resulted in decreased human health. This is exemplified in diseases such as malaria, cholera, and in higher heavy metal concentration in blood which has shown to lead to nerve diseases as well as cancer. Investments in water supply and sanitation process can yield a net economic benefit, such reduction in health care cost and productive healthy individuals in the society. For the above reasons we need a machine learning model to help us classify and predict water potability to save time and money.

2 Dataset

The data-set used in this project is a data collection of different water bodies downloaded from Kaggle [1]. Each data point in in the data-set has nine variables. These variables are different properties of the water including pH, hardness, total dissolved solids, amount of Chloramines, amount of sulfates, electrical conductivity, amount of organic carbon, amount of Trihalomethanes, turbidity, and potability. The data contains also a test result (response variable) of the sample. The response variable is a a binary variable indicating the test result: "1" indicating the water is potable and "0" indicating the water is not potable.

3 Exploratory Data Analysis

Before running any machine learning algorithm, we clean the data from any missing data and visualize some dependant variables distribution in the potable and not potable water. we Also show correlation plot between the variables. Lastely we run PCA representation and data distribution to Explore the linearity of the data and also see whether the data is balanced or imbalanced.

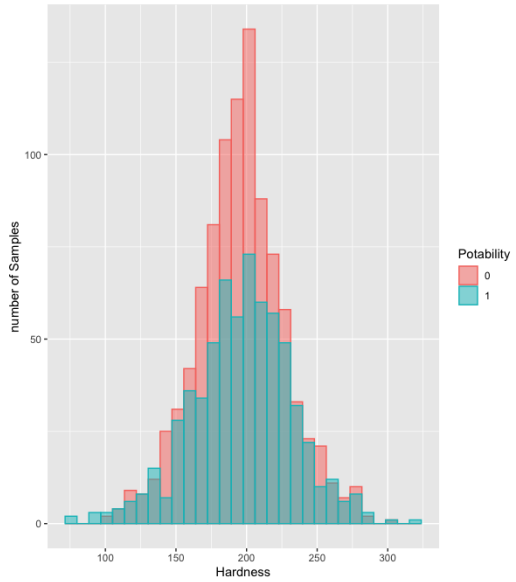


Figure 1: Hardness distribution in both potable and non-potable water.

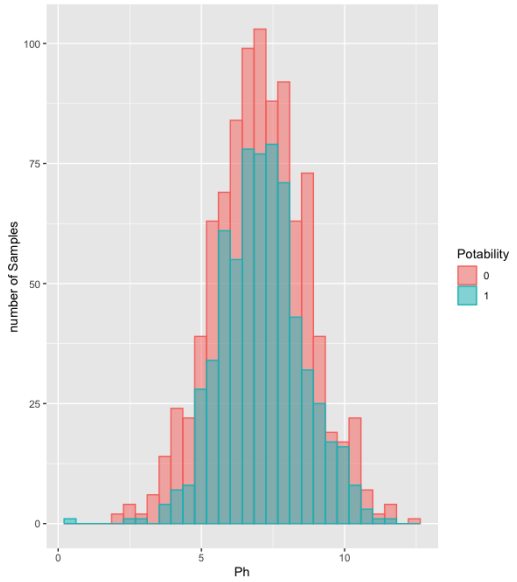


Figure 2: pH distribution in both potable and non-potable water.

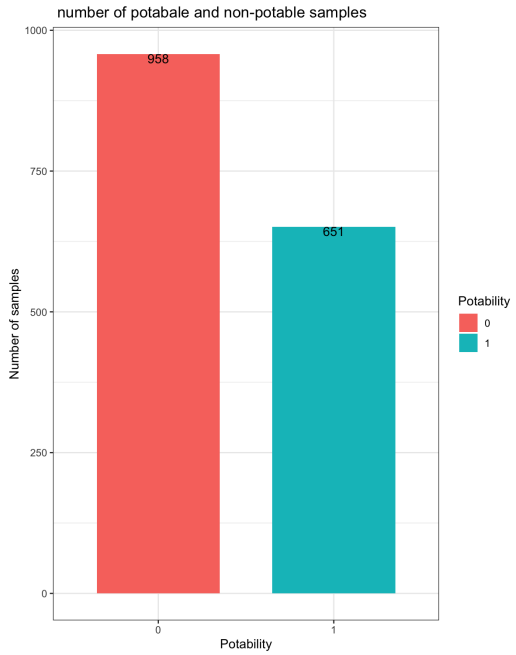


Figure 3: Hardness distribution in both potable and non-potable water.

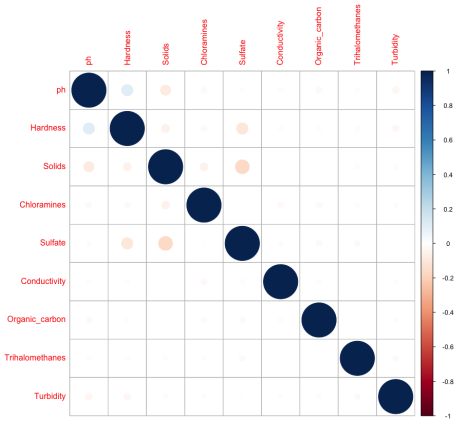
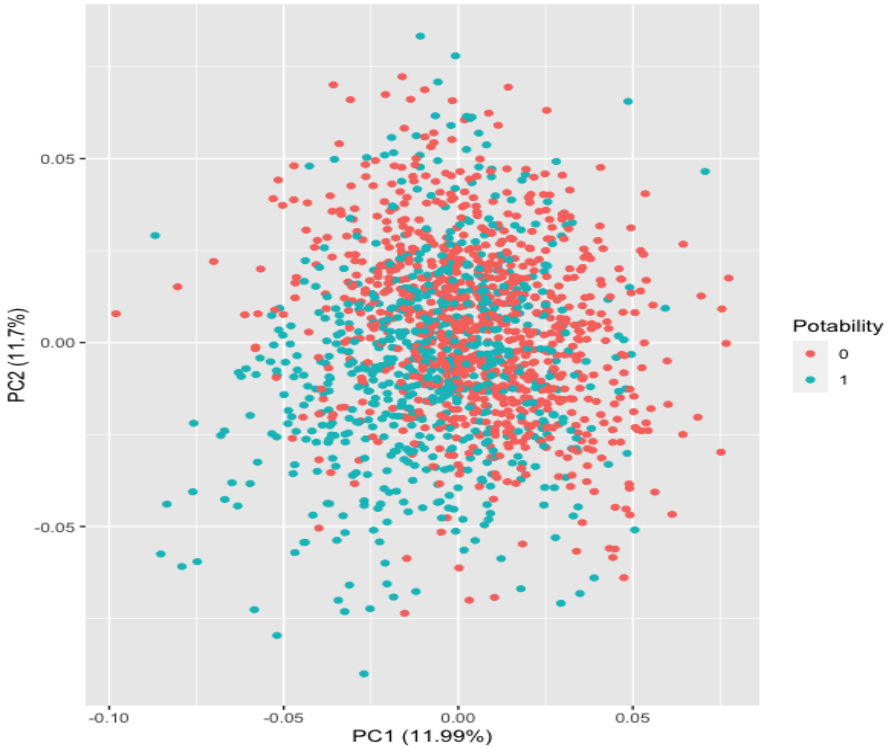


Figure 4: pH distribution in both potable and non-potable water.



As we can see from the above plots there is no correlation between the dependant variables, the variables are independent from each other. Another observation is that the data is imbalanced and the PCA representation shows that the classes are not linearly separable. The data visualization tells us that we need to balance the data and Also consider a non-linear classfier for our data classfication task.

4 Methods/Methodology

- Removing missing data, split the data to %80 training and %20 for testing and scaling the data.
- Using different classification (linear & non-linear) algorithms to classify the water sample whether is potable or not potable based on the water properties (pH, hardness, amount of sulfate, organic carbon etc...).
- The different classification algorithms used are LDA, Naive Bayes, Logistic Regression, Logistic regression with Stepwise varaible selection, KNN, RandomForest, Adaboost and SVM with non-linear kenel .
- Pick the best classifier based on accuracy and precision of test data. Running the best model of the imbalanced and Balanced data.
- Improve the classifier accuracy by: Variable selection for logistic regression. Balancing the imbalanced data using ROSE method. Tuning the Hyperparameters using CV.

Note: The PCA representation suggests using Non-linear models, however we will also run some linear models (e.g LDA, Logistic regression, QDA) for comparison and to confirm.

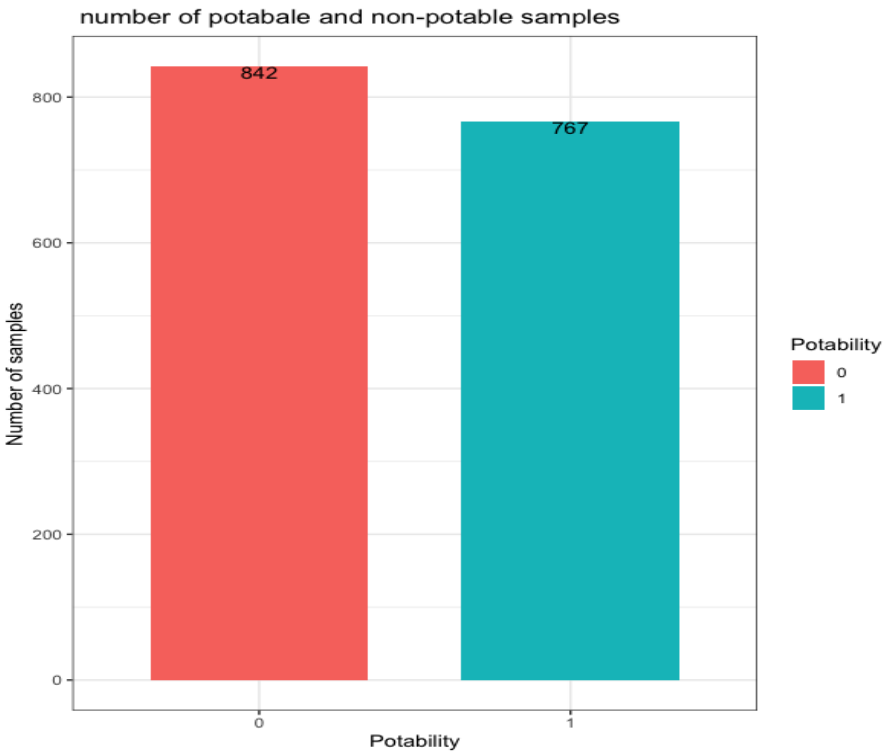
5 Results

The table below shows the test accuracy and precision of imbalanced data for each model.

	Random Forest	Ada-Boost	SVM (non-linear, Gaussian Kernel)	NaiveBayes	LDA	Logistic regression with All variables	Lgistic regreesion with StepWise AIC selection	KNN (K=4)
Test accuracy	0.675	0.637	0.674	0.617	0.602	0.510	0.547	0.5201
Test Precision	0.7137	0.621	0.662	0.629	0.605	0.60	0.5928	0.586

Running cross validation for KNN, gives best k=4 and resulted in test accuracy of Running stepwise AIC variable selection for logistic regression leads to one variable selection "Solids" and resulted in test accuracy of Among the linear models LDA performs better with accuracy of The non-linear models outperformed all the linear models (as expected from PCA insights).
Running Cross validation for RandomForest lead to number of trees of 200 and number of variable of 5. (see cross validation charts in the appendix section) with accuracy of % 67.5 and precision of % 71.37
Running SVM with kernel lead to accuracy of % 67.4 and precision of % 66.2.
The two Best performing models overall are RandomForest and SVM, based on accuracy and precision.

Balancing the data using ROSE method.



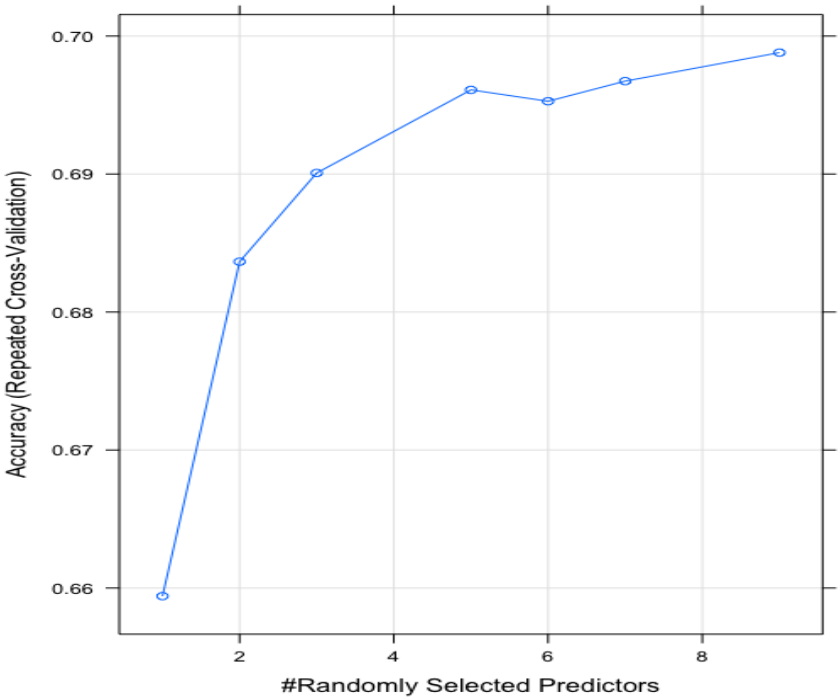
The table below shows the test accuracy and precision of balanced data for each model.

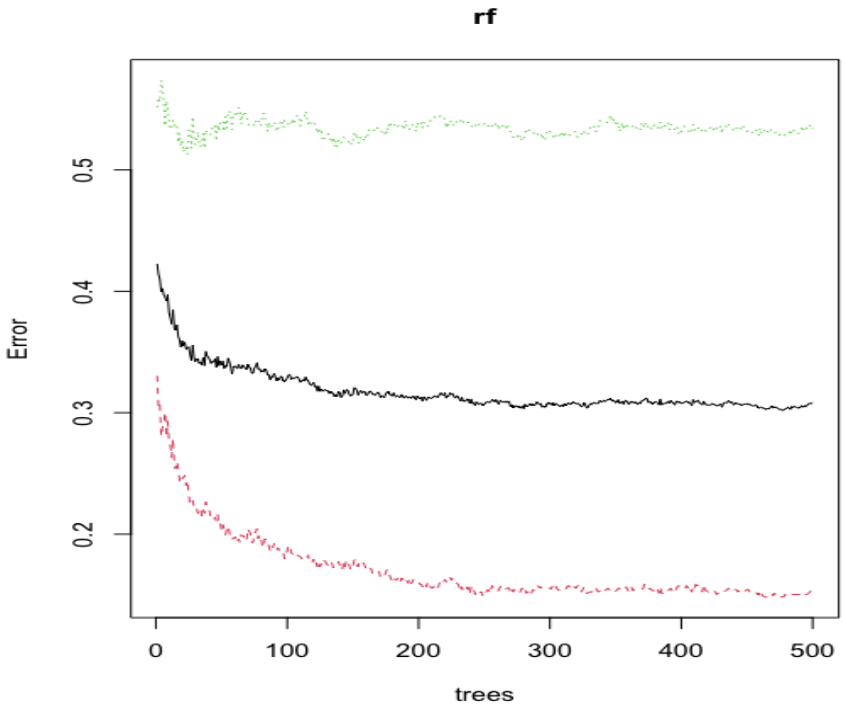
	Random Forest	Ada-Boost	SVM (non-linear, Gaussian Kernel)	NaiveBayes	LDA	Logistic regression with All variables	Lgistic regreesion with StepWise AIC selection	KNN (K=3)
Test accuracy	0.552	0.577	0.639	0.60	0.565	0.355	0.385	0.53
Test Precision	0.58609	0.542	0.666	0.626	0.611	0.13	0.135	0.587

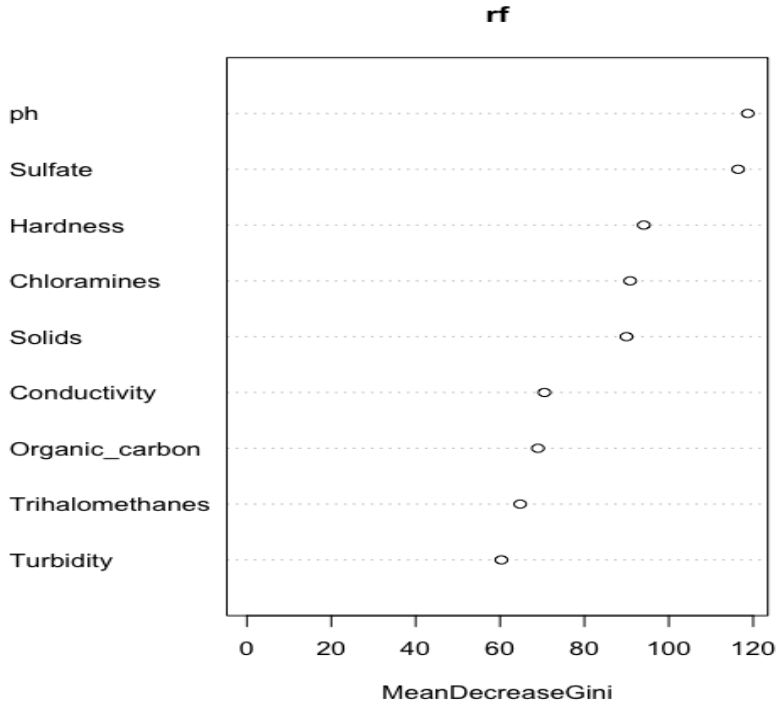
Running cross validation for KNN, gives best $k = 4$ and resulted in test accuracy of % 53 and precision of % 58.
Running stepwise AIC variable selection for logistic regression leads to two variable selection "Sulfate" and "Trihalomehanes" resulted in test accuracy of % 38.5 precision % 13.5.
Among the linear models LDA performs better with accuracy of % 56.5.
The non-linear models outperformed all the linear models (as expected from PCA insights).
The two Best performing models overall are SVM and NaiveBayes, with accuracy of % 63.9 and % 60 and precision of %66.6 and %62.6 respectively.
Running Cross validation for RandomForest lead to number of trees of 200 and number of variable of 3. (see cross validation charts in the appendix section) with accuracy of % 55.2, precision % 58.6.

The plot 1 and 2 below shows the curves for hyper-parameter tuning for random forest, the best

number of variable split in random-forest is mtry=9 (best accuracy) and best number of trees is 200. The plot 3 below shows the importance of each variable based on the Gini metric.




rf
Error
0.2
0.3
0.4
0.5
0
100
200
300
400
500
trees


rf
ph
Sulfate
Hardness
Chloramines
Solids
Conductivity
Organic_carbon
Trihalomethanes
Turbidity
0
20
40
60
80
100
120
MeanDecreaseGini

6 Findings/Conclusion

Evaluating Models performance on imbalance and balanced data based on accuracy is not always the right approach (know as the accuracy paradox). Evaluating the models based on precision, recall and specify is more proper (in our case we evaluated performance based on precision).

The best performing classification model is SVM based on accuracy and precision for balanced data and Random Forest for imbalanced data.
The reason Random forest and SVM perform better compared to other models is because the Random forest and SVM can handle non- linearly separable data better.
The best accuracy and precision we achieved is %67.5 and %70 (Random Forest), we would like to see higher accuracy especially when it comes to public safety.

We were expecting that balancing the data will increase the accuracy and precision however it did not, this might be due to the nature of the data and size of the data-set.Also another reason is that there is some information loss when balancing the data for relatively small data set.
Relatively small data-set limits the performance, larger data is needed to investigate and improve classification models.

7 Lessons Learned

An important lesson we learned from this project is that in order to build a more accurate model especially when it comes to public safety, more data is needed to draw a conclusion and make an accurate prediction, an accuracy of %70 is not enough in this specific classification problem (potable water classification). Another learned lesson is that depending of the nature of the problem, a careful metric selection for evaluating the model is needed (e.g accuracy vs precision).

Finally we learned in this course is that a good data exploration techniques and understanding of the data can help selecting the right model, and also help understand why some models perform better than others, pre-processing data and choosing the right metric and cross validation for evaluation are critical tasks in machine learning.

8 References

1. https://www.kaggle.com/datasets/adityakadiwal/water-potability

9 Appendix

```
library(dplyr)
library(plotly)
library(GGally)
library("ggplot2")
```

```
library(leaps)
library(MASS)
library(glmnet)
library(pls)
library(randomForest)
library(adabag)
library(caret)
library(ada)
library(corrplot)
library(nnet)
library(caTools)
library(ROCR)
library(pROC)
library(randomForest)
library('smotefamily')
library("ROSE")
### Data Preparation
## Read Data
rm(list = ls())
water_potability <- read.csv("~/Downloads/water_potability.csv")
## Split to training and testing subset
set.seed(123)
clean_data<- na.omit(water_potability)
clean_data$Potability=as.factor(clean_data$Potability);
n<-dim(clean_data)[1]
flag <- sort(sample(n,floor(0.2*n), replace = FALSE))
datatrain <- clean_data[-flag,]
datatest <- clean_data[flag,]

train_std_deviation= apply(datatrain[,-10],2,sd)
train_mean= apply(datatrain[,-10],2,mean)

scaled_train =  (datatrain[,-10] - train_mean) / train_std_deviation
scaled_test = (datatest[,-10] - train_mean) / train_std_deviation
scaled_train$Potability=datatrain[,10]
scaled_test$Potability=datatest[,10]
## Extra the true response value for training and testing data
y1  <- as.factor(datatrain$Potability);
y2   <- as.factor(datatest$Potability);



## Data Exploratory Analysis (EDA)
ggpairs(clean_data, columns = 1:9)#+theme(strip.text = element_text(size = 1))

#ggplotly(p1)
```

```
#correlation Matrix
M=cor(clean_data[,-10])
corrplot(M)
# PCA representation

library(ggfortify)
# PCA visualization
df <- datatrain
df$Potability = as.numeric(df$Potability)

#df$Patability=as.numeric(datatrain$Potability)
pca_im <- prcomp(df, scale. = TRUE)

autoplot(pca_im)

a<-autoplot(pca_im, data = datatrain, colour = 'Potability')
a

library(ggplot2)
# Bar chart
ggplot(datatrain, aes(x=Potability,fill=Potability))+geom_bar(stat="count",
                                                              width=0.7)+
geom_text(stat='count', aes(label=..count..), vjust=1)+
labs(title = "number of potabale and non-potable samples  ",x="Potability",
     y="Number of samples")+theme_bw()

# Use semi-transparent fill
ph<-ggplot(datatrain, aes(x=ph, fill=Potability, color=Potability)) +
  geom_histogram(position="identity", alpha=0.5)+labs(y= "number of Samples",
                                                      x = "Ph")

ph

Hardness<-ggplot(datatrain, aes(x=Hardness, fill=Potability, color=Potability)) +
  geom_histogram(position="identity", alpha=0.5)+labs(y= "number of Samples",
                                                      x = "Hardness")

Hardness

Org<-ggplot(datatrain, aes(x=Organic_carbon, fill=Potability, color=Potability)) +
  geom_histogram(position="identity", alpha=0.5)+labs(y= "number of Samples", x
                                                      = "Organic Carbon")

Org


S<-ggplot(datatrain, aes(x=Turbidity, fill=Potability, color=Potability)) +
  geom_histogram(position="identity", alpha=0.5)+labs(y= "number of Samples",
```

```
                                                    x = "Solids")

S
#######################################################
# Using Balanced data
datatrain<- ROSE(Potability ~ ., data = datatrain,seed = 1)$data
ggplot(datatrain, aes(x=Potability, fill=Potability))+geom_bar(stat="count",
                                                               width=0.7)+
  geom_text(stat='count', aes(label=..count..), vjust=1)+
  labs(title = "number of potabale and non-potable samples  ",x="Potability",
       y="Number of samples")+theme_bw()
#####################################################

# Cross Validation for Random Forest
control <- trainControl(method="repeatedcv", number=10, repeats=3,
                        search="random")
set.seed(1)
mtry <- sqrt(ncol(datatrain))
#rf1 <- randomForest(y1 ~., data=datatrain[,-10],
#                    ntree= 600,
 #                   mtry=mtry, nodesize =2, importance=TRUE)
rf1 <- train(Potability~., data=datatrain, method="rf", metric="Accuracy",
             tuneLength=15, trControl=control)
print(rf1)
plot(rf1)

## Check Important variables
importance(rf1)
rf <- randomForest(y1~., data=datatrain[,-10], mtry=6)
## There are two types of importance measure
##  (1=mean decrease in accuracy,
##   2= mean decrease in node impurity)
importance(rf, type=2)
varImpPlot(rf)

## The plots show that V52, V53, V7, V55 are among the most
##      important features when predicting V58.

## Prediction on the testing data set
rf.pred = predict(rf, datatest[,-10], type='class')
t1= table(rf.pred, y2)
sum(diag(table(rf.pred, y2)))/sum(table(rf.pred,y2))

rc1= confusionMatrix(t1)
precision <- rc1$byClass['Pos Pred Value']
precision

##In practice, You can fine-tune parameters in Random Forest such as
```

```
#ntree = number of tress to grow, and the default is 500.
#mtry = number of variables randomly sampled as candidates at each split.
#          The default is sqrt(p) for classfication and p/3 for regression
#nodesize = minimum size of terminal nodes.
#           The default value is 1 for classification and 5 for regression
plot(rf)

### ababoost
datatrain$Potability <- as.factor(datatrain$Potability)
model_adaboost <- boosting(Potability~., data=datatrain,boos=TRUE)
summary(model_adaboost)
pred_test1 = predict(model_adaboost, datatest)
pred_test1$error
sum(diag(pred_test$confusion))/sum(pred_test$confusion)

t6= table(as.factor(pred_test1), datatest$Potability)
rc6= confusionMatrix(t6)
precision <- rc6$byClass['Pos_Pred_Value']
precision



### KNN, Linear , LDA, logistic regression
# Using KNN
library(class);
xnew <- scaled_train[,1:9];
training_err=c()
for (kk in c(1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31)){

  ypred2.train <- knn(scaled_train[,1:9], xnew, scaled_train[,10], k=kk);
  training_err= c(training_err,mean( ypred2.train != scaled_train[,10]))
}
# KNN training error for each K value
training_err
plot(c(1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31),
     training_err,main ="K-nearest Vs Training Error", type = "b", pch = 19,
     col = "red", xlab = "K-nearest", ylab = "Training Error")
### 3. Testing Error

## Testing error of KNN, and you can change the k values.
xnew2 <- scaled_test[,1:9]; ## xnew2 is the X variables of the "testing" data

ypred2.test <- knn(scaled_train[,1:9], xnew2, scaled_train[,10], k=11);
test_err= mean( ypred2.test!= scaled_test[,10])
t1= table(ypred2.test, datatest$Potability)
rc1= confusionMatrix(t1)
precision <- rc1$byClass['Pos_Pred_Value']
# Test error for different k values
```

```
test_err
precision


TrainErr <- NULL;
TestErr  <- NULL;
#selecting only the variables that are useful in predicting mpg
mod1 <- lda(datatrain[,1:9], datatrain[,10]);

## training error
## we provide a detailed code here
pred1 <- predict(mod1,datatrain[,1:9])$class;
lda_TrainErr <- mean( pred1 != datatrain$Potability);
lda_TrainErr;
## testing error
pred1test <- predict(mod1,datatest[,1:9])$class;
lda_testerr=mean(pred1test != datatest$Potability)
lda_testerr
t2= table(pred1test,  datatest$Potability)
rc2= confusionMatrix(t2)
precision <- rc2$byClass['Pos_Pred_Value']
precision
## Method 2: QDA
mod2 <- qda(datatrain[,1:9], datatrain[,10])
## Training Error
pred2 <- predict(mod2,datatrain[,1:9])$class
qda_trainerr=mean( pred2!= datatrain$Potability)
qda_trainerr

##  Testing Error
qda_testerr=mean( predict(mod2,datatest[,1:9])$class != datatest$Potability)
qda_testerr

##Logistic Regression
# Training model
logistic_model <- glm(Potability ~ .,
                      data = scaled_train,
                      family = "binomial")


# Summary
summary(logistic_model)

# Predict test data based on model
predtrain <- predict(logistic_model,
                     scaled_train[,1:9], type = "response")
```

```
# figuring out the best probability treshhold to categorize the data
prediction(predtrain, scaled_train$Potability) %>%
  performance(measure = "tpr", x.measure = "fpr") -> result

plotdata <- data.frame(x = result@x.values[[1]],
                       y = result@y.values[[1]],
                       p = result@alpha.values[[1]])

p <- ggplot(data = plotdata) +
  geom_path(aes(x = x, y = y)) +
  xlab(result@x.name) +
  ylab(result@y.name) +
  theme_bw()

dist_vec <- plotdata$x^2 + (1 - plotdata$y)^2
opt_pos <- which.min(dist_vec)

p +
  geom_point(data = plotdata[opt_pos, ],
             aes(x = x, y = y), col = "red") +
  annotate("text",
           x = plotdata[opt_pos, ]$x + 0.1,
           y = plotdata[opt_pos, ]$y,
           label = paste("p=", round(plotdata[opt_pos, ]$p, 3)))
# Changing probabilities
predtr<- ifelse(predtrain >0.411, 1, 0)
# training error
logit_trainerr=mean(predtr != datatrain$Potability)
logit_trainerr
# Evaluating model accuracy
# using confusion matrix


table(datatrain$Potability, predtr)

predtest<- predict(logistic_model,
                   datatest[,1:9], type = "response")

# Changing probabilities
predts<- ifelse(predtest >0.411, 1, 0)
# training error
logit_testerr= mean(predts!= scaled_test$Potability)
logit_testerr
table(predts,  scaled_test$Potability)

t3= table(predts,  datatest$Potability)
rc3= confusionMatrix(t3)
precision <- rc3$byClass['Pos_Pred_Value']
```

```
precision
# Naive bayes

library(e1071)
mod3 <- naiveBayes( datatrain[,1:9], datatrain[,10])
#mod3 <- naive_bayes(Potability~ ., data = datatrain,type = 'prob' )
## Training Error
prednb3 <- predict(mod3, datatrain)
(tab2 <- table(prednb3 , datatrain$Potability))
NBtrainerr= mean( prednb3 != datatrain$Potability)
NBtrainerr
##  0.2765152 for miss.class.train.error of Naive Bayes
## Testing Error
predt3 <- predict(mod3, datatest)
NBtesterr= mean( predict(mod3,datatest) != datatest$Potability)
NBtesterr
t4= table(predt3,  datatest$Potability)
rc4= confusionMatrix(t4)
precision <- rc4$byClass['Pos_Pred_Value']
precision
# SVM
library(kernlab)
## Model #4: Gaussian-kernel SVM
fit4 <- ksvm(Potability ~ ., data = datatrain,C=1, kernel="rbfdot")
y_pred= predict(fit4, datatest,type ="response")
ytest=datatest$Potability
table(y_pred,ytest)
sum(diag(table(y_pred,ytest)))/sum(table(y_pred,ytest))
##   This should be the same model #1 from the "e1701" package



t5= table(y_pred,  datatest$Potability)
rc5= confusionMatrix(t5)
precision <- rc5$byClass['Pos_Pred_Value']
precision

#pca_res <- prcomp(clean_data[,-10], scale. = TRUE)
#autoplot(pca_res, data=clean_data[,-10], colour=clean_data[,1])

library(MASS)
full.model <- glm(Potability ~., data = datatrain, family = binomial)
coef(full.model)
step.model <- full.model %>% stepAIC(trace = FALSE)
coef(step.model)
```

```
# Make predictions
probabilities <- full.model %>% predict(datatest[,-10], type = "response")
predicted.classes <- ifelse(probabilities > 0.411, 1, 0)

# Balancing the data using Rose Package
t5= table(predicted.classes,  datatest$Potability)
rc5= confusionMatrix(t5)
precision <- rc5$byClass['Pos Pred Value']
precision
# Prediction accuracy
mean(predicted.classes == scaled_test$Potability)
# Make predictions
probabilities <- predict(step.model, datatest[,-10], type = "response")
predicted.classes <- ifelse(probabilities > 0.411, 1, 0)
# Prediction accuracy
t5= table(predicted.classes,  datatest$Potability)
rc5= confusionMatrix(t5)
precision <- rc5$byClass['Pos Pred Value']
precision
mean(predicted.classes == scaled_test$Potability)
```