Wine Quality Prediction

Prathamesh Desai

General imports/library calls and other definitions

```
library(ggplot2)
library(plyr)
library(knitr)
library(e1071)
library(caret)
```

```
## Loading required package: lattice
```

```
library(class)

set.seed(100)

cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC
  79A7")

options(scipen = 4)
```

Introduction: White wine data

In this project, I performed model selection with k-Nearest-Neighbors, Logistic Regression and Naïve Bayes' classifiers.

The data for this exercise came from wine industry. Each record represented a sample of a specific wine product, the inputs were a list of its organoleptic characteristics, and the output were the quality class of each wine: {high, low}. The labels were assigned by human wine tasting experts.

*** Overall Goal ***

The goal was to build the best model to predict wine quality from its characteristics, so that the winery could replace costly services of professional sommeliers with the automated alternative, to enable quick and effective quality tracking of their wines at production facilities. Thus, they could know whether such change is feasible, and what kinds of inaccuracies may be involved.

**** Setting up the classifiers ****

- Multiple functions were implemented that would accept training set data frame and testing set data frame.
- The functions would use the corresponding algorithm and train using the training set data frame and run predictions on the testing set data frame.
- The prediction results obtained on testing set data frame were then returned by the functions. Specifically, each of the function below returned dataframes with two columns: prediction (probability where applicable or class label), and true output.
- The last column of the data in each of the data frame was considered as the target/output variable, and the output is binary {0,1}.

** Implemented R function that runs logistic regression model

implementation: glm with binomial (logit) model

```
get_pred_logreg <- function(train, test) {
    class_vector <- train[,length(train)]
    levels(class_vector) = c("low", "high")

    factor_class_vector <- as.factor(class_vector)

    colnames(train)[length(train)] <- c("out")

    train["out"] <- factor_class_vector

    glm.fit <- glm(out~., family=binomial(link='logit'), data=train)

  result_predict <- predict(glm.fit, test, type= "response")

  #Dataframe of predicted and true values of Test
  df <- data.frame(result_predict,test[,length(test)])

  return(df)

}
```

** Implemented R function that runs naïve bayes' model

implementation: e1071:naiveBayes

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

    class_vector <- train[,length(train)]

    factor_class_vector <- as.factor(class_vector)
    levels(factor_class_vector) <- c("low", "high")

    colnames(train)[length(train)] <- c("out")

    train["out"] <- factor_class_vector

    nb <- naiveBayes(out~., data=train)

    result_predict <- predict(nb, test, type="class")

    #Dataframe of predicted and true values of Test
    df <- data.frame(result_predict,test[,length(test)])

    return(df)

}
```

** Implemented R function that runs knn model

implementation: class:knn

```
get_pred_knn <- function(train, test, k) {

  df_trn <- data.frame(train[,1:length(train)-1])
  df_test <- data.frame(test[,1:length(train)-1])
  class_label <- train[,length(train)]

  result <- knn(train = df_trn,
      test = df_test,
      cl = class_label,
      k = k)

  #Dataframe of predicted and true values of Test
  df <- data.frame(result,test[,length(test)])

  return(df)
}
```

**** Setting up the Cross Validation Function ****

A generic function for k-fold cross-validation was implemented for selected classification models model_name= {‘logreg’,‘nb’,‘knn’}

Note: For the use with knn model, parsing of ‘model_name’ argument was done to identify the number of neighbors assuming that ‘k’ in ‘knn’ encodes that value (so `do_cv_class(df,10,"5nn"")` would execute 10-fold cross-validation on frame ‘df’ using 5-Nearest-Neighbor classifier).

```
set.seed(100)

do_cv_class <- function(df, num_folds, model_name) {

  class_vector <- df[,length(df)]
  levels(class_vector) = c("low", "high")

  factor_class_vector <- as.factor(class_vector)
  colnames(df)[5] <- c("out")
  df["out"] <- factor_class_vector

  K <- num_folds

  Final_Data_Frame <-  NULL

  #Split the rows into into K parts
  splits  <- split(df, f = rep_len(sample(seq(1,K), K), nrow(df)))

  #Iterate over K and test on 1 to K chunks of data
  for(i in 1:K){

  test <- splits[[i]]

  test_vector <- NULL

  test_vector <-  c(strtoi(row.names(test)))

  if(model_name == 'logreg'){

    trainingdata <- df[-test_vector,]
    testingdata <-  df[test_vector,]

    temp_Frame <- get_pred_logreg(trainingdata,testingdata)
    Final_Data_Frame <- rbind(Final_Data_Frame, temp_Frame)

  }else if(model_name == 'nb'){

    trainingdata <- df[-test_vector,]
    testingdata <-  df[test_vector,]

    temp_Frame <- get_pred_nb(trainingdata,testingdata)
    Final_Data_Frame <- rbind(Final_Data_Frame, temp_Frame)

  }else{

   k_in_knn <-  strsplit(model_name,"nn")[[1]]
   k_in_knn <- strtoi(k_in_knn[1])

   trainingdata <- df[-test_vector,]
   testingdata <-  df[test_vector,]

   temp_Frame <- get_pred_knn(trainingdata,testingdata,k_in_knn)
   Final_Data_Frame <- rbind(Final_Data_Frame, temp_Frame)
```

```
  }
  }


Final_Data_Frame <- setNames(Final_Data_Frame, c('Predicted_Values', 'True_Values'))
return (Final_Data_Frame)
}
```

**** Model Evaluation - Metrics ****

** R function called `get_metrics` was implemented which had:

- Input:
  - Prediction data frame: The first column of it contained predicted values (probability or label depending on the type of model), the second column represented true lables/target(0/1)
  - Cutoff: A numeric parameter with the default value of 0.5 that specified the probability threshold value when prediction of the binary classifier is a probability.
- Output:
  - A data frame with elements (tpr, fpr, acc, precision, recall) tpr -> True Positive rate fpr -> False Positive rate acc -> Accuracy

```
get_metrics <- function(results, cutoff = 0.5) {

if(is.numeric(results$Predicted_Values)){
  results$Predicted_Values[results["Predicted_Values"] <= cutoff] = "Low"
  results$Predicted_Values[results["Predicted_Values"] != "Low"] = "High"

  tbl <- table(results$Predicted_Values,results$True_Values)

  TP <- tbl[1,2]
  FP <- tbl[1,1]
  FN <- tbl[2,2]
  TN <- tbl[2,1]

  accuracy <- (TP + TN)/(TP + FP + FN + TN)
  precision <- (TP)/(TP + FP)
  recall <- (TP)/(TP + FN)
  FPR <- (FP)/(FP + TN)

  performance <- data.frame(accuracy,precision,recall,FPR)
  return(performance)

 }else{
  tbl <- table(results$Predicted_Values,results$True_Values)

  TP <- tbl[2,2]
  FP <- tbl[2,1]
  FN <- tbl[1,2]
  TN <- tbl[1,1]

  accuracy <- (TP + TN)/(TP + FP + FN + TN)
  precision <- (TP)/(TP + FP)
  recall <- (TP)/(TP + FN)
  FPR <- (FP)/(FP + TN)

  performance <- data.frame(accuracy,precision,recall,FPR)
  return(performance)

 }

}
```

**** Application on Wines dataset ****

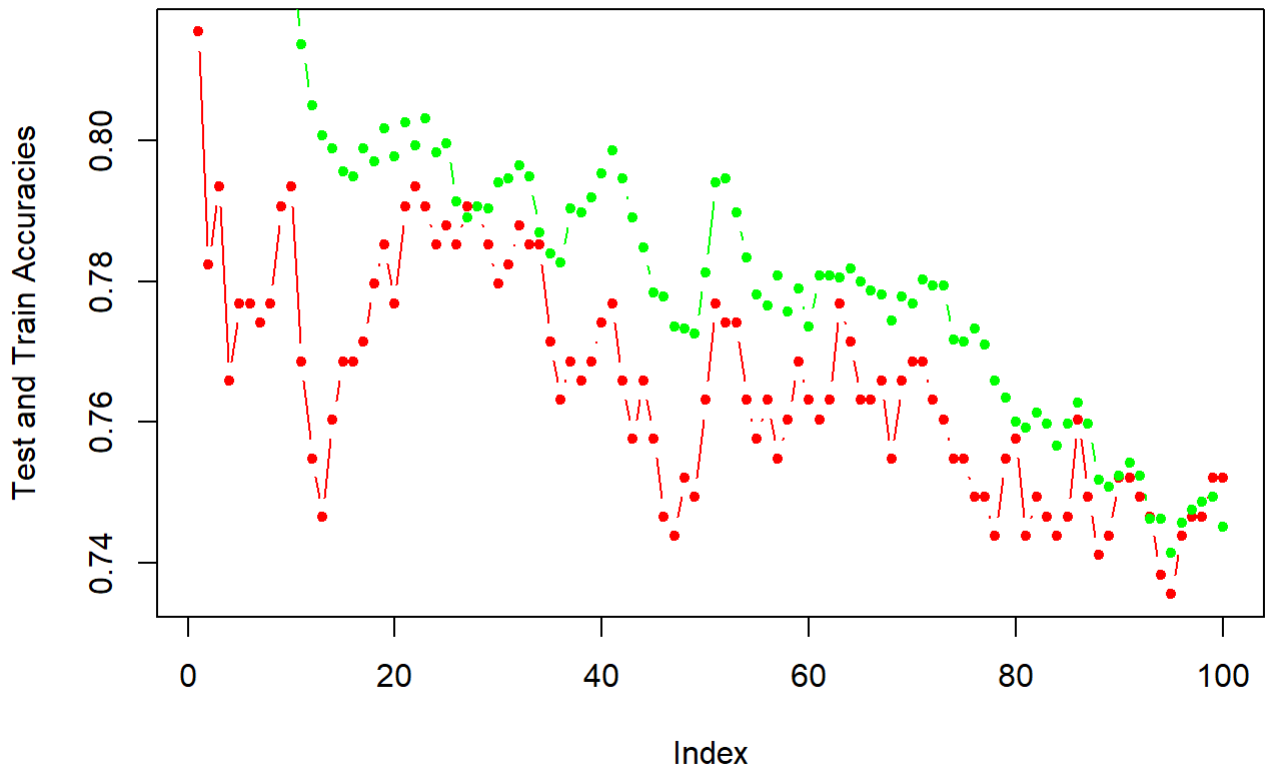
A model was built to predict the quality wine: {high, low}, using all other attributes

```
wines <- read.csv("./white_wine.csv", header = TRUE)
```

** Using `do_cv_class` , `get_metrics` and corresponding model function(s) implemented above,the best k in kNN model was found that generalizes best. As the number of neighbors varies, the ranges of k when the models overfit and underfit data were found.

```
train_accuracy <- NULL
test_accuracy <- NULL

for(i in seq(1,100)){

  set.seed(100)

  do_cv_train <- function(df, num_folds, model_name){
    class_vector <- df[,5]
    levels(class_vector) = c("low", "high")
    factor_class_vector <- as.factor(class_vector)

    colnames(df)[5] <- c("out")
    df["out"] <- factor_class_vector
    K <- num_folds
    Final_Data_Frame <-  NULL

  #Split the rows into into K parts
  splits  <- split(df, f = rep_len(sample(seq(1,K), K), nrow(df)))

  #Iterate over K and test on 1 to K chunks of data
  for(i in 1:K){

    test <- splits[[i]]

    test_vector <- NULL

    test_vector <-  c(strtoi(row.names(test)))

    k_in_knn <-  strsplit(model_name,"nn")[[1]]
    k_in_knn <- strtoi(k_in_knn[1])

    trainingdata <- df[-test_vector,]
    testingdata <-  df[-test_vector,]

    temp_Frame <- get_pred_knn(trainingdata,testingdata,k_in_knn)

    Final_Data_Frame <- rbind(Final_Data_Frame, temp_Frame)
  }

  Final_Data_Frame <- setNames(Final_Data_Frame, c('Predicted_Values', 'True_Values'))
  return (Final_Data_Frame)
  }

  k <- as.character(i)
  k <- paste(k,"nn",sep="")

  results <- do_cv_class(wines, 10, k)
  results_train <- do_cv_train(wines, 10, k)

  performance_test <- get_metrics(results, cutoff = 0.5)
  performance_train <- get_metrics(results_train, cutoff = 0.5)
```

```
  test_accuracy <- c(test_accuracy,c(performance_test$accuracy))
  train_accuracy <-  c(train_accuracy, c(performance_train$accuracy))

}

dataframe <- data.frame(train_accuracy, test_accuracy)

plot(test_accuracy, type = "b", col = "red", cex = 1, pch=20, ylab = "Test and Train Accuracies"
  )
lines(train_accuracy, type = "b", col= "green", cex = 1, pch = 20)
```




```
print(test_accuracy[20:30])
```

```
##  [1] 0.7768595 0.7906336 0.7933884 0.7906336 0.7851240 0.7878788 0.7851240
##  [8] 0.7906336 0.7906336 0.7851240 0.7796143
```

```
print(paste("At k=22, the train accuracy is ",train_accuracy[22]))
```

```
## [1] "At k=22, the train accuracy is  0.799204162840526"
```

```
print(paste("At k=22, the test accuracy is ",test_accuracy[22]))
```

```
## [1] "At k=22, the test accuracy is  0.793388429752066"
```

The above plot describes the accuracies for training and testing data for various values of k (in this case -> 1 to 100). The red dots indicate the accuracy for testing data, whereas the green dots indicate the accuracy for training data. The best k in knn model is the one which provides good accuracy and the model does not overfit or underfit.

Overfitting is when the training data performs well, but the testing data does not perform as better as the training data. In the above figure, it can be noticed that the model overfits in the initial ranges of k. The accuracy for training data is pretty high, while the accuracy for testing data is comparatively lower in the range of k from 1 to 15. Hence, as per me, the ideal k shall not lie between this range as the data overfits in this range.

Underfitting is when both the training and testing data do not perform well. In this case, there is no prominent underfitting at any range as the accuracy from 1 to 100 varies between 0.74 to 0.80.

The range from 20 to 30 seems to be the ideal range for k as the data does not overfit, neither does it underfit. In this range, k=22 provides the highest accuracy for test which is 79.338 %, hence as per me, the k that best generalizes in thsi knn model is k = 22.

** Using `do_cv_class` , `get_metrics` and corresponding model function(s) that were implemented above, the two types of parametric models for white wine data were fitted and the type of model yielding the highest accuracy was found.

```
#10-Fold Cross Validation
result_logreg <- do_cv_class(wines, 10,'logreg')
logreg_performance <- get_metrics(result_logreg, cutoff = 0.5)
logreg_accuracy <- logreg_performance$accuracy

result_nb <- do_cv_class(wines, 10, 'nb')
nb_performance <- get_metrics(result_nb, cutoff = 0.5)
nb_accuracy <-  nb_performance$accuracy



if(nb_accuracy > logreg_accuracy){print(paste("For 10-Fold Cross Validation, Naive Bayes model p
  rovides higher accuracy as ", nb_accuracy*100,"%"," > ",logreg_accuracy*100,"%"))}else{print(p
  aste("For 10-Fold Cross Validation, Logistic Regression provides higher accuracy as ",logreg_a
  ccuracy*100,"%"," > ", nb_accuracy*100,"%"))}
```

```
## [1] "For 10-Fold Cross Validation, Logistic Regression provides higher accuracy as  77.410468
3195592 %  >  77.1349862258953 %"
```

```
#5-Fold Cross Validation

result_logreg <- do_cv_class(wines, 5,'logreg')
logreg_performance <- get_metrics(result_logreg, cutoff = 0.5)
logreg_accuracy <- logreg_performance$accuracy

result_nb <- do_cv_class(wines, 5, 'nb')
nb_performance <- get_metrics(result_nb, cutoff = 0.5)
nb_accuracy <-  nb_performance$accuracy



if(nb_accuracy > logreg_accuracy){print(paste("For 5-Fold Cross Validation, Naive Bayes model pr
  ovides higher accuracy as ", nb_accuracy*100,"%"," > ",logreg_accuracy*100,"%"))}else{print(pa
  ste("For 5-Fold Cross Validation, Logistic Regression provides higher accuracy as ",logreg_acc
  uracy*100,"%"," > ", nb_accuracy*100,"%"))}
```

```
## [1] "For 5-Fold Cross Validation, Naive Bayes model provides higher accuracy as  77.410468319
5592 %  >  77.1349862258953 %"
```

```
default_classifier_accuracy <-  count((wines[5]=="high"))[2,2] / (count((wines[5]=="high"))[2,2]
  + count((wines[5]=="high"))[1,2])

print(paste("The accuracy for default classifier of this data is ", default_classifier_accuracy*
  100,"%"))
```

```
## [1] "The accuracy for default classifier of this data is  49.5867768595041 %"
```

As per the calculations above, the Naive bayes model provides an accuracy of 77.1349 %, whereas Logistic Regression model provides an accuracy of 77.4104% for 10-fold Cross Validation. So, it can be said that, for 10-fold cross validation, Logistic Regression model performs slightly better than the Naive Bayes Model as the difference between accuracies suggest that the their performance is quite close. Different results can be obtained when the vale of K in K-fold Cross Validation is different. For example, with a lower K, the accuracies are different for the two models as shown above. For 5-fold CV, Naive Bayes Model seems to perform slightly better than Logistic Regression Model and that could impact the decision to choose.

The default classifier for this dataset is ‘high’ as we want to find which wine is the best out of all, and we are classifying in order to find the good quality wine. So the accuracy for default classifier is 49.5867%

** Summary

As per the findings from above, it can be seen that the accuracies obtained in case of Naive Bayes and Logistic Regression model are about 77% for different folds between 5 to 10 for k-fold cross validation.

Whereas in case of Knn model, when the k is chosen as 22, it provides accuracy of about 79%, which is better than both Logistic Regression Model and Naive Bayes Model.

Though, this dataset is as not as large, it can be argued that with a larger dataset, the choice could be different and other models might provide better accuracy.

But for the current white-wine data, it can be said that the Knn Model with k=22 works best and it should be chosen over the other two.