DATA 622 - Test 1

OMER OZEREN

Table of Contents

## A)

Run Bagging (ipred package)

* sample with replacement
* estimate metrics for a model
* repeat as many times as specied and report the average

### Load the Data

df <- read.table("~/GitHub/DATA622/data.txt",header = T,sep=',')  
df$label <- ifelse(df$label =="BLACK",1,0)  
df$y <- as.numeric(df$y)  
df$X <- as.factor(df$X)

### Split Data into Train (70%) and Test data(30%)

set.seed(42)  
split\_df <- createDataPartition(df$label, p = .70, list = FALSE)  
df\_train <- df[split\_df,]  
df\_test <- df[-split\_df,]

### Model Performance Estimator

estimate\_model\_performance <- function(y\_true, y\_pred, model\_name){  
 cm <- confusionMatrix(table(y\_true, y\_pred))  
 cm\_table <- cm$table  
 tpr <- cm\_table[[1]] / (cm\_table[[1]] + cm\_table[[4]])  
 fnr <- 1 - tpr  
 fpr <- cm\_table[[3]] / (cm\_table[[3]] + cm\_table[[4]])  
 tnr <- 1 - fpr  
 accuracy <- cm$overall[[1]]  
 for\_auc <- prediction(c(y\_pred), y\_true)  
 auc <- performance(for\_auc, "auc")  
 auc <- auc@y.values[[1]]  
 return(data.frame(Algo = model\_name, AUC = auc, ACCURACY = accuracy, TPR = tpr, FPR = fpr, TNR = tnr, FNR = fnr))  
}

### NB Model Building - Standalone

nb\_model<-naiveBayes(df\_train$label~.,data=df\_train)  
nb\_testpred<-predict(nb\_model,df\_test,type='raw')  
nb\_testclass<-unlist(apply(round(nb\_testpred),1,which.max))-1  
nb\_table<-table(df\_test$label, nb\_testclass)  
nb\_cm<-caret::confusionMatrix(nb\_table)  
nb\_cm

## Confusion Matrix and Statistics  
##   
## nb\_testclass  
## 0 1  
## 0 2 0  
## 1 2 6  
##   
## Accuracy : 0.8   
## 95% CI : (0.4439, 0.9748)  
## No Information Rate : 0.6   
## P-Value [Acc > NIR] : 0.1673   
##   
## Kappa : 0.5455   
##   
## Mcnemar's Test P-Value : 0.4795   
##   
## Sensitivity : 0.50   
## Specificity : 1.00   
## Pos Pred Value : 1.00   
## Neg Pred Value : 0.75   
## Prevalence : 0.40   
## Detection Rate : 0.20   
## Detection Prevalence : 0.20   
## Balanced Accuracy : 0.75   
##   
## 'Positive' Class : 0   
##

### Estimate NB model test data () performance

rst\_nb<-estimate\_model\_performance(df\_test$label,nb\_testclass,'NB')  
rst\_nb

## Algo AUC ACCURACY TPR FPR TNR FNR  
## 1 NB 0.875 0.8 0.25 0 1 0.75

### Bagging Methodology - NB Model

I’m going to create a function for boostrap purposes first.I’m going to run NB model 50 times and store the performance metrics for each data boostrap.

apply\_bootstrap\_data <- function(data, proportion = 0.7, sample\_with\_replacement = TRUE){  
 observation <- round(nrow(data) \* proportion, 0)  
 return(data[sample(nrow(data), observation, replace = sample\_with\_replacement),])  
}

for (i in 1:50){  
 sample <- apply\_bootstrap\_data(df\_train)  
 nb\_model <- naiveBayes(sample$label ~ ., data = sample)  
 y\_pred <- predict(nb\_model, df\_test,type='raw') # probability  
 y\_pred\_class<-unlist(apply(round(y\_pred),1,which.max))-1 # class  
 performance <- estimate\_model\_performance(df\_test$label, y\_pred\_class, paste("NB Bootstrap ", i))  
 if(exists("performance\_table\_nb")){  
 performance\_table\_nb <- rbind(performance\_table\_nb, performance)  
 } else {  
 performance\_table\_nb <- performance  
 }  
}

### NB Boostrap Results Table

performance\_table\_nb

## Algo AUC ACCURACY TPR FPR TNR FNR  
## 1 NB Bootstrap 1 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 2 NB Bootstrap 2 0.3125 0.5 0.0000000 0.2857143 0.7142857 1.0000000  
## 3 NB Bootstrap 3 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 4 NB Bootstrap 4 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 5 NB Bootstrap 5 0.7500 0.6 0.3333333 0.0000000 1.0000000 0.6666667  
## 6 NB Bootstrap 6 0.1875 0.3 0.0000000 0.4000000 0.6000000 1.0000000  
## 7 NB Bootstrap 7 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 8 NB Bootstrap 8 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 9 NB Bootstrap 9 0.3750 0.6 0.0000000 0.2500000 0.7500000 1.0000000  
## 10 NB Bootstrap 10 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 11 NB Bootstrap 11 0.3125 0.5 0.0000000 0.2857143 0.7142857 1.0000000  
## 12 NB Bootstrap 12 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 13 NB Bootstrap 13 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 14 NB Bootstrap 14 0.6875 0.8 0.1250000 0.1250000 0.8750000 0.8750000  
## 15 NB Bootstrap 15 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 16 NB Bootstrap 16 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 17 NB Bootstrap 17 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 18 NB Bootstrap 18 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 19 NB Bootstrap 19 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 20 NB Bootstrap 20 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 21 NB Bootstrap 21 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 22 NB Bootstrap 22 0.3125 0.5 0.0000000 0.2857143 0.7142857 1.0000000  
## 23 NB Bootstrap 23 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 24 NB Bootstrap 24 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 25 NB Bootstrap 25 0.7500 0.6 0.3333333 0.0000000 1.0000000 0.6666667  
## 26 NB Bootstrap 26 0.6250 0.4 0.5000000 0.0000000 1.0000000 0.5000000  
## 27 NB Bootstrap 27 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 28 NB Bootstrap 28 0.7500 0.6 0.3333333 0.0000000 1.0000000 0.6666667  
## 29 NB Bootstrap 29 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 30 NB Bootstrap 30 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 31 NB Bootstrap 31 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 32 NB Bootstrap 32 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 33 NB Bootstrap 33 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 34 NB Bootstrap 34 0.4375 0.7 0.0000000 0.2222222 0.7777778 1.0000000  
## 35 NB Bootstrap 35 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 36 NB Bootstrap 36 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 37 NB Bootstrap 37 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 38 NB Bootstrap 38 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 39 NB Bootstrap 39 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 40 NB Bootstrap 40 0.3125 0.5 0.0000000 0.2857143 0.7142857 1.0000000  
## 41 NB Bootstrap 41 0.6875 0.8 0.1250000 0.1250000 0.8750000 0.8750000  
## 42 NB Bootstrap 42 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 43 NB Bootstrap 43 0.6875 0.5 0.4000000 0.0000000 1.0000000 0.6000000  
## 44 NB Bootstrap 44 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 45 NB Bootstrap 45 0.6250 0.4 0.5000000 0.0000000 1.0000000 0.5000000  
## 46 NB Bootstrap 46 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 47 NB Bootstrap 47 0.6250 0.4 0.5000000 0.0000000 1.0000000 0.5000000  
## 48 NB Bootstrap 48 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 49 NB Bootstrap 49 0.5625 0.6 0.1666667 0.1666667 0.8333333 0.8333333  
## 50 NB Bootstrap 50 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778

### The Mean of Boostrap NB model

mean(performance\_table\_nb$ACCURACY)

## [1] 0.7

### The Variance of Boostrap NB model

var(performance\_table\_nb$ACCURACY)

## [1] 0.02285714

Now, I’m going to try KNN stand alone and boostrap methodology.For the KNN model, I will use K =3.

### KNN Model Building - Standalone

knn\_y\_true<- knn(df\_train[1:2],df\_test[1:2], cl = df\_train$label, k = 5)  
knn\_testclass<-knn\_y\_true  
knn\_table<-table(df\_test$label, knn\_testclass)  
knn\_cm<-caret::confusionMatrix(knn\_table)  
knn\_cm

## Confusion Matrix and Statistics  
##   
## knn\_testclass  
## 0 1  
## 0 2 0  
## 1 2 6  
##   
## Accuracy : 0.8   
## 95% CI : (0.4439, 0.9748)  
## No Information Rate : 0.6   
## P-Value [Acc > NIR] : 0.1673   
##   
## Kappa : 0.5455   
##   
## Mcnemar's Test P-Value : 0.4795   
##   
## Sensitivity : 0.50   
## Specificity : 1.00   
## Pos Pred Value : 1.00   
## Neg Pred Value : 0.75   
## Prevalence : 0.40   
## Detection Rate : 0.20   
## Detection Prevalence : 0.20   
## Balanced Accuracy : 0.75   
##   
## 'Positive' Class : 0   
##

### Estimate KNN model test data () performance

rst\_knn<-estimate\_model\_performance(df\_test$label,knn\_testclass,'KNN')  
rst\_knn

## Algo AUC ACCURACY TPR FPR TNR FNR  
## 1 KNN 0.875 0.8 0.25 0 1 0.75

### Bagging Methodology - KNN Model

I’m going to create a function for boostrap purposes first.I’m going to run KNN model 50 times and store the performance metrics for each data boostrap.

apply\_bootstrap\_data <- function(data, proportion = 0.7, sample\_with\_replacement = TRUE){  
 observation <- round(nrow(data) \* proportion, 0)  
 return(data[sample(nrow(data), observation, replace = sample\_with\_replacement),])  
}

for (i in 1:50){  
 sample <- apply\_bootstrap\_data(df\_train)  
 y\_pred <- knn(sample[1:2],df\_test[1:2], cl = sample$label, k = 3)  
 y\_pred\_class<-y\_pred  
 performance <- estimate\_model\_performance(df\_test$label, y\_pred\_class, paste("KNN Bootstrap ", i))  
 if(exists("performance\_table\_knn")){  
 performance\_table\_knn <- rbind(performance\_table\_knn, performance)  
 } else {  
 performance\_table\_knn <- performance  
 }  
}

### KNN Boostrap Results Table

performance\_table\_knn

## Algo AUC ACCURACY TPR FPR TNR FNR  
## 1 KNN Bootstrap 1 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 2 KNN Bootstrap 2 0.7500 0.6 0.3333333 0.0000000 1.0000000 0.6666667  
## 3 KNN Bootstrap 3 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 4 KNN Bootstrap 4 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 5 KNN Bootstrap 5 0.6250 0.4 0.5000000 0.0000000 1.0000000 0.5000000  
## 6 KNN Bootstrap 6 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 7 KNN Bootstrap 7 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 8 KNN Bootstrap 8 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 9 KNN Bootstrap 9 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 10 KNN Bootstrap 10 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 11 KNN Bootstrap 11 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 12 KNN Bootstrap 12 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 13 KNN Bootstrap 13 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 14 KNN Bootstrap 14 0.6875 0.5 0.4000000 0.0000000 1.0000000 0.6000000  
## 15 KNN Bootstrap 15 0.4375 0.7 0.0000000 0.2222222 0.7777778 1.0000000  
## 16 KNN Bootstrap 16 0.7500 0.6 0.3333333 0.0000000 1.0000000 0.6666667  
## 17 KNN Bootstrap 17 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 18 KNN Bootstrap 18 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 19 KNN Bootstrap 19 0.2500 0.4 0.0000000 0.3333333 0.6666667 1.0000000  
## 20 KNN Bootstrap 20 1.0000 1.0 0.2000000 0.0000000 1.0000000 0.8000000  
## 21 KNN Bootstrap 21 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 22 KNN Bootstrap 22 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 23 KNN Bootstrap 23 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 24 KNN Bootstrap 24 0.2500 0.4 0.0000000 0.3333333 0.6666667 1.0000000  
## 25 KNN Bootstrap 25 0.5000 0.8 0.0000000 0.2000000 0.8000000 1.0000000  
## 26 KNN Bootstrap 26 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 27 KNN Bootstrap 27 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 28 KNN Bootstrap 28 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 29 KNN Bootstrap 29 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 30 KNN Bootstrap 30 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 31 KNN Bootstrap 31 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 32 KNN Bootstrap 32 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 33 KNN Bootstrap 33 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 34 KNN Bootstrap 34 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 35 KNN Bootstrap 35 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 36 KNN Bootstrap 36 0.3750 0.6 0.0000000 0.2500000 0.7500000 1.0000000  
## 37 KNN Bootstrap 37 0.6875 0.5 0.4000000 0.0000000 1.0000000 0.6000000  
## 38 KNN Bootstrap 38 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 39 KNN Bootstrap 39 0.6875 0.8 0.1250000 0.1250000 0.8750000 0.8750000  
## 40 KNN Bootstrap 40 0.3750 0.6 0.0000000 0.2500000 0.7500000 1.0000000  
## 41 KNN Bootstrap 41 0.6875 0.5 0.4000000 0.0000000 1.0000000 0.6000000  
## 42 KNN Bootstrap 42 0.4375 0.7 0.0000000 0.2222222 0.7777778 1.0000000  
## 43 KNN Bootstrap 43 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 44 KNN Bootstrap 44 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 45 KNN Bootstrap 45 0.9375 0.9 0.2222222 0.0000000 1.0000000 0.7777778  
## 46 KNN Bootstrap 46 0.7500 0.6 0.3333333 0.0000000 1.0000000 0.6666667  
## 47 KNN Bootstrap 47 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 48 KNN Bootstrap 48 0.8125 0.7 0.2857143 0.0000000 1.0000000 0.7142857  
## 49 KNN Bootstrap 49 0.8750 0.8 0.2500000 0.0000000 1.0000000 0.7500000  
## 50 KNN Bootstrap 50 1.0000 1.0 0.2000000 0.0000000 1.0000000 0.8000000

### The Mean of Boostrap KNN model

mean(performance\_table\_knn$ACCURACY)

## [1] 0.726

### The Variance of Boostrap KNN model

var(performance\_table\_knn$ACCURACY)

## [1] 0.01992245

## B)

Run LOOCV (jacknife) for the same dataset

* iterate over all points
* keep one observation as test
* train using the rest of the observations
* determine test metrics
* aggregate the test metrics

end of loop

find the average of the test metric(s)

Compare (A), (B) above with the results you obtained in HW-1 and write 3 sentences explaining the

observed difference.

### Jacknife: Leave One Out (LOO) Cross Validation -KNN Model

For each observation train with aLL other observations predict that one observation.

y\_pred\_train\_loocv\_knn <- c()  
for (i in 1:nrow(df\_train)){  
 loocv\_test <- df\_train[i,]  
 loocv\_train\_df <- df\_train[-c(i),]  
 y\_pred\_train\_loocv\_knn <- c(y\_pred\_train\_loocv\_knn, knn(loocv\_train\_df[1:2], loocv\_test[1:2], loocv\_train\_df$label, k = 3))  
 y\_pred\_test\_loocv\_knn <- knn(loocv\_train\_df[1:2], df\_test[1:2], loocv\_train\_df$label, k = 3)  
 performance <- estimate\_model\_performance(df\_test$label, y\_pred\_test\_loocv\_knn, paste("KNN - LOOCV", i))  
 if(exists("performance\_table\_knn\_loocv")){  
 performance\_table\_knn\_loocv <- rbind(performance\_table\_knn\_loocv, performance)  
 } else {  
 performance\_table\_knn\_loocv <- performance  
 }  
}

### KNN LOOCV Results

performance\_table\_knn\_loocv

## Algo AUC ACCURACY TPR FPR TNR FNR  
## 1 KNN - LOOCV 1 0.9375 0.9 0.2222222 0 1 0.7777778  
## 2 KNN - LOOCV 2 0.9375 0.9 0.2222222 0 1 0.7777778  
## 3 KNN - LOOCV 3 0.8125 0.7 0.2857143 0 1 0.7142857  
## 4 KNN - LOOCV 4 0.8125 0.7 0.2857143 0 1 0.7142857  
## 5 KNN - LOOCV 5 0.8750 0.8 0.2500000 0 1 0.7500000  
## 6 KNN - LOOCV 6 0.8750 0.8 0.2500000 0 1 0.7500000  
## 7 KNN - LOOCV 7 0.8750 0.8 0.2500000 0 1 0.7500000  
## 8 KNN - LOOCV 8 0.8750 0.8 0.2500000 0 1 0.7500000  
## 9 KNN - LOOCV 9 0.8750 0.8 0.2500000 0 1 0.7500000  
## 10 KNN - LOOCV 10 0.8750 0.8 0.2500000 0 1 0.7500000  
## 11 KNN - LOOCV 11 0.8750 0.8 0.2500000 0 1 0.7500000  
## 12 KNN - LOOCV 12 0.8750 0.8 0.2500000 0 1 0.7500000  
## 13 KNN - LOOCV 13 0.8750 0.8 0.2500000 0 1 0.7500000  
## 14 KNN - LOOCV 14 0.8750 0.8 0.2500000 0 1 0.7500000  
## 15 KNN - LOOCV 15 0.8750 0.8 0.2500000 0 1 0.7500000  
## 16 KNN - LOOCV 16 0.8750 0.8 0.2500000 0 1 0.7500000  
## 17 KNN - LOOCV 17 0.8750 0.8 0.2500000 0 1 0.7500000  
## 18 KNN - LOOCV 18 0.8750 0.8 0.2500000 0 1 0.7500000  
## 19 KNN - LOOCV 19 0.8750 0.8 0.2500000 0 1 0.7500000  
## 20 KNN - LOOCV 20 0.8750 0.8 0.2500000 0 1 0.7500000  
## 21 KNN - LOOCV 21 0.8750 0.8 0.2500000 0 1 0.7500000  
## 22 KNN - LOOCV 22 0.8750 0.8 0.2500000 0 1 0.7500000  
## 23 KNN - LOOCV 23 0.8750 0.8 0.2500000 0 1 0.7500000  
## 24 KNN - LOOCV 24 0.8750 0.8 0.2500000 0 1 0.7500000  
## 25 KNN - LOOCV 25 0.8750 0.8 0.2500000 0 1 0.7500000  
## 26 KNN - LOOCV 26 0.8750 0.8 0.2500000 0 1 0.7500000

### The Mean of LOOCV KNN model

mean(performance\_table\_knn\_loocv$ACCURACY)

## [1] 0.8

### The Variance of LOOCV KNN mode

var(performance\_table\_knn\_loocv$ACCURACY)

## [1] 0.0016

### Jacknife: Leave One Out (LOO) Cross Validation -NB Model

y\_pred\_train\_loocv\_nb <- c()  
for (i in 1:nrow(df\_train)){  
 loocv\_test <- df\_train[i,]  
 loocv\_train\_df <- df\_train[-c(i),]  
 nb\_model <- naiveBayes(loocv\_train\_df$label ~ ., data = loocv\_train\_df)  
 y\_pred\_train\_loocv\_nb <- predict(nb\_model, loocv\_test[1:2],type='raw') # probability  
 y\_pred\_train\_loocv\_class\_nb<-unlist(apply(round(y\_pred\_train\_loocv\_nb),1,which.max))-1 # class  
 y\_pred\_train\_loocv\_nb <- c(y\_pred\_train\_loocv\_nb,y\_pred\_train\_loocv\_class\_nb)   
 y\_pred\_test\_loocv\_nb <- predict(nb\_model, df\_test[1:2],type='raw') # probability  
 y\_pred\_test\_loocv\_class\_nb<-unlist(apply(round(y\_pred\_test\_loocv\_nb),1,which.max))-1 # class  
 performance <- estimate\_model\_performance(df\_test$label, y\_pred\_test\_loocv\_class\_nb, paste("NB - LOOCV", i))   
 if(exists("performance\_table\_nb\_loocv")){  
 performance\_table\_nb\_loocv <- rbind(performance\_table\_nb\_loocv, performance)  
 } else {  
 performance\_table\_nb\_loocv <- performance  
 }  
}

### NB LOOCV Results

performance\_table\_nb\_loocv

## Algo AUC ACCURACY TPR FPR TNR FNR  
## 1 NB - LOOCV 1 0.9375 0.9 0.2222222 0 1 0.7777778  
## 2 NB - LOOCV 2 0.9375 0.9 0.2222222 0 1 0.7777778  
## 3 NB - LOOCV 3 0.8125 0.7 0.2857143 0 1 0.7142857  
## 4 NB - LOOCV 4 0.8125 0.7 0.2857143 0 1 0.7142857  
## 5 NB - LOOCV 5 0.8750 0.8 0.2500000 0 1 0.7500000  
## 6 NB - LOOCV 6 0.8750 0.8 0.2500000 0 1 0.7500000  
## 7 NB - LOOCV 7 0.8750 0.8 0.2500000 0 1 0.7500000  
## 8 NB - LOOCV 8 0.8750 0.8 0.2500000 0 1 0.7500000  
## 9 NB - LOOCV 9 0.8750 0.8 0.2500000 0 1 0.7500000  
## 10 NB - LOOCV 10 0.8750 0.8 0.2500000 0 1 0.7500000  
## 11 NB - LOOCV 11 0.8750 0.8 0.2500000 0 1 0.7500000  
## 12 NB - LOOCV 12 0.8750 0.8 0.2500000 0 1 0.7500000  
## 13 NB - LOOCV 13 0.8750 0.8 0.2500000 0 1 0.7500000  
## 14 NB - LOOCV 14 0.8750 0.8 0.2500000 0 1 0.7500000  
## 15 NB - LOOCV 15 0.8750 0.8 0.2500000 0 1 0.7500000  
## 16 NB - LOOCV 16 0.8750 0.8 0.2500000 0 1 0.7500000  
## 17 NB - LOOCV 17 0.8750 0.8 0.2500000 0 1 0.7500000  
## 18 NB - LOOCV 18 0.8750 0.8 0.2500000 0 1 0.7500000  
## 19 NB - LOOCV 19 0.8750 0.8 0.2500000 0 1 0.7500000  
## 20 NB - LOOCV 20 0.8750 0.8 0.2500000 0 1 0.7500000  
## 21 NB - LOOCV 21 0.8750 0.8 0.2500000 0 1 0.7500000  
## 22 NB - LOOCV 22 0.8125 0.7 0.2857143 0 1 0.7142857  
## 23 NB - LOOCV 23 0.8750 0.8 0.2500000 0 1 0.7500000  
## 24 NB - LOOCV 24 0.8750 0.8 0.2500000 0 1 0.7500000  
## 25 NB - LOOCV 25 0.8750 0.8 0.2500000 0 1 0.7500000  
## 26 NB - LOOCV 26 0.8750 0.8 0.2500000 0 1 0.7500000

### The Mean of LOOCV NB model

mean(performance\_table\_nb\_loocv$ACCURACY)

## [1] 0.7961538

### The Variance of LOOCV NB model

var(performance\_table\_nb\_loocv$ACCURACY)

## [1] 0.001984615

### Compate Metrics

print(paste('NB:',rst\_nb$ACCURACY))

## [1] "NB: 0.8"

print(paste('Bagging -NB :',mean(performance\_table\_nb$ACCURACY)))

## [1] "Bagging -NB : 0.7"

print(paste('KNN:',rst\_knn$ACCURACY))

## [1] "KNN: 0.8"

print(paste('Bagging -KNN :',mean(performance\_table\_knn$ACCURACY)))

## [1] "Bagging -KNN : 0.726"

print(paste('LOO-CV/KNN:',mean(performance\_table\_knn\_loocv$ACCURACY)))

## [1] "LOO-CV/KNN: 0.8"

print(paste('LOO-CV/NB:',mean(performance\_table\_nb\_loocv$ACCURACY)))

## [1] "LOO-CV/NB: 0.796153846153846"

## Summary

The above display shows results of each methods model performance.Initially, HMW1 KNN stand alone model(0.8) results somewhere better than both bagging methodologies for KNN(0.72) and NB(0.7).The KNN stand alone model performs better than the KNN with bagging method.In addition to that, stand alone NB model performs better than bagging methodology with NB. This is one of the drawnback boostrap methodology that differences due to randomly assign samples in boostrap method.In the boostrap methodoloy ,I used 50 iterations and for each iteration model perform vary due to sampling data set for training randomly.The stand alone KNN and NB models perform almost the the same results with LOOCV.