Homework 6

Karan Ashar

3/30/2021

set.seed(608)  
library(e1071)

## Warning: package 'e1071' was built under R version 4.0.4

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 4.0.4

## Loading required package: rpart

library(rpart)  
library(splitTools)

## Warning: package 'splitTools' was built under R version 4.0.4

library(ranger)

## Warning: package 'ranger' was built under R version 4.0.4

library(caret)

## Warning: package 'caret' was built under R version 4.0.4

## Loading required package: lattice

## Loading required package: ggplot2

library(MLmetrics)

## Warning: package 'MLmetrics' was built under R version 4.0.4

##   
## Attaching package: 'MLmetrics'

## The following objects are masked from 'package:caret':  
##   
## MAE, RMSE

## The following object is masked from 'package:base':  
##   
## Recall

library(plyr)

## Warning: package 'plyr' was built under R version 4.0.4

## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.  
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:  
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##   
## Attaching package: 'plyr'

## The following objects are masked from 'package:dplyr':  
##   
## arrange, count, desc, failwith, id, mutate, rename, summarise,  
## summarize

library(klaR)

## Warning: package 'klaR' was built under R version 4.0.4

## Loading required package: MASS

##   
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':  
##   
## select

## 

## Introduction

We have been given a dataset where each row represets a digit (0-9). The columns represent each pixel and the value in the column represent the intensity of each pixel. Our goal is to build a model which could predict the digit based on the pixel values. This is a classificaton problem. To be specific it is a multiClass classification problem. We will build two models using Decision Tree and Naive Bayes. We will then compare the results to see which performs better. After loading the data I had to carry out one pre-processing step. I had to convert our output label to the words they correspond to. For example, “1” would convert to “One”. We had to do this because of package dependencies.

#Loading the data and baisc pre-processing  
data=read.csv("digit-train.csv")  
data$label <- mapvalues(data$label, from=c("1","2","3","4","5","6","7","8","9","0"), to=c("One","Two","Three","Four", "Five","Six","Seven","Eight","Nine","Zero"))  
  
test=read.csv("digit-test.csv")  
test$label <- mapvalues(test$label, from=c("1","2","3","4","5","6","7","8","9","0"), to=c("One","Two","Three","Four", "Five","Six","Seven","Eight","Nine","Zero"))

## Decision Tree

Here we will train a decision tree model. We will use the k-fold cross validation approach to do so. We select K to be equal to 3. We will select the summaryFunction as multiClassSummary as our output is a multiClass output.

train\_control <- trainControl(method = "cv",   
 number = 3,  
 classProbs=TRUE,  
 summaryFunction=multiClassSummary  
 )  
start\_time <- Sys.time()  
decision\_tree <- train(label~.,data=data,   
 method = "rpart",   
 trControl = train\_control,  
 tuneLength=30  
 )

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :  
## There were missing values in resampled performance measures.

end\_time <- Sys.time()  
end\_time - start\_time

## Time difference of 31.90751 secs

print(decision\_tree)

## CART   
##   
## 4198 samples  
## 784 predictor  
## 10 classes: 'Eight', 'Five', 'Four', 'Nine', 'One', 'Seven', 'Six', 'Three', 'Two', 'Zero'   
##   
## No pre-processing  
## Resampling: Cross-Validated (3 fold)   
## Summary of sample sizes: 2797, 2799, 2800   
## Resampling results across tuning parameters:  
##   
## cp logLoss AUC prAUC Accuracy Kappa   
## 0.001878186 2.873818 0.9030057 0.49750502 0.7291591 0.69898714  
## 0.002146499 2.561414 0.9055166 0.46452141 0.7220130 0.69109037  
## 0.002414811 2.471893 0.9029559 0.41603078 0.7165320 0.68499842  
## 0.002548967 2.404295 0.9027711 0.42338475 0.7151045 0.68339664  
## 0.002683123 2.404295 0.9027711 0.42338475 0.7151045 0.68339664  
## 0.002951435 2.292086 0.9021286 0.36111187 0.7082009 0.67571281  
## 0.003219748 2.176676 0.8996438 0.32892062 0.7015324 0.66828591  
## 0.004024685 1.944294 0.8981158 0.29243333 0.6912915 0.65691298  
## 0.004561309 1.827771 0.8881190 0.29864832 0.6769895 0.64102807  
## 0.004829622 1.800029 0.8885284 0.26309737 0.6729379 0.63652423  
## 0.005097934 1.782496 0.8869490 0.24619725 0.6710345 0.63439748  
## 0.005634559 1.754417 0.8846474 0.25284059 0.6676964 0.63070704  
## 0.006439496 1.680151 0.8828686 0.21968807 0.6531612 0.61454289  
## 0.006976120 1.666332 0.8813415 0.21799401 0.6512558 0.61243113  
## 0.007781057 1.600249 0.8787959 0.18587695 0.6436311 0.60389799  
## 0.008049369 1.589082 0.8783954 0.17297914 0.6429158 0.60310658  
## 0.009390931 1.543752 0.8741687 0.16853778 0.6310032 0.58975497  
## 0.009927556 1.528485 0.8701166 0.15880740 0.6250464 0.58317385  
## 0.011805742 1.529326 0.8608235 0.14394807 0.6114826 0.56805497  
## 0.012342367 1.543006 0.8573275 0.14042016 0.6033873 0.55901328  
## 0.013147303 1.559618 0.8510551 0.13445803 0.5971924 0.55216731  
## 0.022001610 1.569810 0.8440641 0.11341443 0.5774217 0.52997232  
## 0.035685538 1.588156 0.8276563 0.09451438 0.5326257 0.47998188  
## 0.036758787 1.588156 0.8276563 0.09451438 0.5326257 0.47998188  
## 0.050174403 1.644476 0.8087448 0.07843893 0.4856996 0.42772172  
## 0.066004830 1.691395 0.7869130 0.06459192 0.4232710 0.35700218  
## 0.075395761 1.856836 0.7255703 0.04898592 0.3377447 0.26177264  
## 0.080225382 1.978518 0.6780887 0.03810335 0.2687464 0.18348201  
## 0.080493695 1.978518 0.6780887 0.03810335 0.2687464 0.18348201  
## 0.099543869 2.174445 0.5874988 0.01827251 0.1693582 0.06806112  
## Mean\_F1 Mean\_Sensitivity Mean\_Specificity Mean\_Pos\_Pred\_Value  
## 0.7270520 0.7275547 0.9699317 0.7305057   
## 0.7200485 0.7203498 0.9691541 0.7240952   
## 0.7146661 0.7150436 0.9685428 0.7204827   
## 0.7129743 0.7134506 0.9683808 0.7191926   
## 0.7129743 0.7134506 0.9683808 0.7191926   
## 0.7053749 0.7064538 0.9676108 0.7116230   
## 0.6991573 0.6997239 0.9668656 0.7067491   
## 0.6876999 0.6895626 0.9657282 0.6969957   
## 0.6730937 0.6748222 0.9641457 0.6796486   
## 0.6690728 0.6705753 0.9636974 0.6749425   
## 0.6674600 0.6686339 0.9634829 0.6731467   
## 0.6631161 0.6652025 0.9631183 0.6679302   
## 0.6485406 0.6501409 0.9615058 0.6555066   
## 0.6465808 0.6482463 0.9612956 0.6546627   
## 0.6390796 0.6400647 0.9604385 0.6464543   
## 0.6385027 0.6393453 0.9603600 0.6460649   
## 0.6258358 0.6267799 0.9590151 0.6334109   
## 0.6180586 0.6205811 0.9583664 0.6256630   
## 0.6007138 0.6063396 0.9568552 0.6085964   
## 0.6055370 0.5979084 0.9559516 0.6108236   
## 0.5938550 0.5917977 0.9552727 0.6082250   
## 0.5863615 0.5710625 0.9530362 0.6059094   
## NaN 0.5265175 0.9480142 NaN   
## NaN 0.5265175 0.9480142 NaN   
## NaN 0.4796226 0.9427952 NaN   
## NaN 0.4094467 0.9357285 NaN   
## NaN 0.3249177 0.9262305 NaN   
## NaN 0.2556225 0.9183936 NaN   
## NaN 0.2556225 0.9183936 NaN   
## NaN 0.1555717 0.9068277 NaN   
## Mean\_Neg\_Pred\_Value Mean\_Precision Mean\_Recall Mean\_Detection\_Rate  
## 0.9699602 0.7305057 0.7275547 0.07291591   
## 0.9691594 0.7240952 0.7203498 0.07220130   
## 0.9685722 0.7204827 0.7150436 0.07165320   
## 0.9684249 0.7191926 0.7134506 0.07151045   
## 0.9684249 0.7191926 0.7134506 0.07151045   
## 0.9676780 0.7116230 0.7064538 0.07082009   
## 0.9669366 0.7067491 0.6997239 0.07015324   
## 0.9658592 0.6969957 0.6895626 0.06912915   
## 0.9642519 0.6796486 0.6748222 0.06769895   
## 0.9637883 0.6749425 0.6705753 0.06729379   
## 0.9635635 0.6731467 0.6686339 0.06710345   
## 0.9632141 0.6679302 0.6652025 0.06676964   
## 0.9616053 0.6555066 0.6501409 0.06531612   
## 0.9614041 0.6546627 0.6482463 0.06512558   
## 0.9605443 0.6464543 0.6400647 0.06436311   
## 0.9604621 0.6460649 0.6393453 0.06429158   
## 0.9591527 0.6334109 0.6267799 0.06310032   
## 0.9585390 0.6256630 0.6205811 0.06250464   
## 0.9571453 0.6085964 0.6063396 0.06114826   
## 0.9564004 0.6108236 0.5979084 0.06033873   
## 0.9558192 0.6082250 0.5917977 0.05971924   
## 0.9539325 0.6059094 0.5710625 0.05774217   
## 0.9497546 NaN 0.5265175 0.05326257   
## 0.9497546 NaN 0.5265175 0.05326257   
## 0.9452283 NaN 0.4796226 0.04856996   
## 0.9392526 NaN 0.4094467 0.04232710   
## 0.9312800 NaN 0.3249177 0.03377447   
## 0.9247031 NaN 0.2556225 0.02687464   
## 0.9247031 NaN 0.2556225 0.02687464   
## 0.9180326 NaN 0.1555717 0.01693582   
## Mean\_Balanced\_Accuracy  
## 0.8487432   
## 0.8447520   
## 0.8417932   
## 0.8409157   
## 0.8409157   
## 0.8370323   
## 0.8332948   
## 0.8276454   
## 0.8194840   
## 0.8171364   
## 0.8160584   
## 0.8141604   
## 0.8058234   
## 0.8047710   
## 0.8002516   
## 0.7998526   
## 0.7928975   
## 0.7894737   
## 0.7815974   
## 0.7769300   
## 0.7735352   
## 0.7620494   
## 0.7372659   
## 0.7372659   
## 0.7112089   
## 0.6725876   
## 0.6255741   
## 0.5870080   
## 0.5870080   
## 0.5311997   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.001878186.

As we can see the model is trained on various values of ‘cp’. The value for ‘cp’ for the best ‘AUC’ is 0.002146499. And the value for ‘cp’ for the best ‘Accuracy’ is ‘0.001878186’. There our other various parameters we can see. Few good one’s would be ‘Kappa’ and ‘Mean\_F1’.

For the value of ‘cp’ for the best Accuracy gives us the following results:   
Accuracy = 0.7291591  
Kappa = 0.69898714  
Mean\_F1= 0.7270520

We can also see the model takes 31.90751 seconds to train given these parameters.

Now lets see the results on the Test dataset.

start\_time <- Sys.time()  
prediction=predict(decision\_tree,test[,-1])  
end\_time <- Sys.time()  
end\_time - start\_time

## Time difference of 1.008238 secs

confusionMatrix(table(test$label,prediction))

## Confusion Matrix and Statistics  
##   
## prediction  
## Eight Five Four Nine One Seven Six Three Two Zero  
## Eight 238 23 15 22 21 6 18 27 18 5  
## Five 16 241 13 4 9 13 15 36 11 30  
## Four 13 5 287 57 1 15 9 5 9 3  
## Nine 28 16 10 277 2 25 2 11 9 6  
## One 8 12 9 0 429 5 3 4 8 0  
## Seven 11 8 18 42 5 357 4 10 6 4  
## Six 23 13 18 9 4 5 302 3 11 16  
## Three 19 40 9 22 15 2 5 304 12 18  
## Two 28 20 14 5 16 12 18 12 287 8  
## Zero 14 18 1 1 1 10 2 10 12 345  
##   
## Overall Statistics  
##   
## Accuracy : 0.7306   
## 95% CI : (0.7169, 0.744)  
## No Information Rate : 0.1198   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.7005   
##   
## Mcnemar's Test P-Value : 2.594e-11   
##   
## Statistics by Class:  
##   
## Class: Eight Class: Five Class: Four Class: Nine  
## Sensitivity 0.59799 0.60859 0.72843 0.63098  
## Specificity 0.95921 0.96134 0.96924 0.97100  
## Pos Pred Value 0.60560 0.62113 0.71040 0.71762  
## Neg Pred Value 0.95795 0.95932 0.97180 0.95750  
## Prevalence 0.09481 0.09433 0.09385 0.10457  
## Detection Rate 0.05669 0.05741 0.06837 0.06598  
## Detection Prevalence 0.09362 0.09242 0.09624 0.09195  
## Balanced Accuracy 0.77860 0.78496 0.84883 0.80099  
## Class: One Class: Seven Class: Six Class: Three Class: Two  
## Sensitivity 0.8529 0.79333 0.79894 0.72038 0.74935  
## Specificity 0.9867 0.97118 0.97330 0.96239 0.96514  
## Pos Pred Value 0.8975 0.76774 0.74752 0.68161 0.68333  
## Neg Pred Value 0.9801 0.97509 0.97997 0.96855 0.97459  
## Prevalence 0.1198 0.10719 0.09004 0.10052 0.09123  
## Detection Rate 0.1022 0.08504 0.07194 0.07242 0.06837  
## Detection Prevalence 0.1139 0.11077 0.09624 0.10624 0.10005  
## Balanced Accuracy 0.9198 0.88226 0.88612 0.84139 0.85724  
## Class: Zero  
## Sensitivity 0.79310  
## Specificity 0.98166  
## Pos Pred Value 0.83333  
## Neg Pred Value 0.97622  
## Prevalence 0.10362  
## Detection Rate 0.08218  
## Detection Prevalence 0.09862  
## Balanced Accuracy 0.88738

The accuracy we got on the test data is 0.7306. Very similar to the accuracy we got while training using k-fold cross validation. The Kappa we got is 0.7005. This is also similar. The time for prediction on the test set is 1.0082 seconds

## Naive Bayes

Lets try the other model and see how it’s performing. For this model I mannually entered different values for the hyper-parameters. The grid\_search object contrains the different values of the hyper-parameters. So what happens here is that our model will be trained on all combinations of the values we have entered.

grid\_search=expand.grid(usekernel=c(TRUE,FALSE),  
 laplace=c(0,3,6),  
 adjust=c(0,3,6))  
train\_control=trainControl(method='cv',number=3,summaryFunction = multiClassSummary)  
  
start\_time <- Sys.time()  
naive\_bayes= train(data[,-1],data$label,'naive\_bayes',trControl=train\_control,tuneGrid = grid\_search)

## Warning: model fit failed for Fold1: usekernel= TRUE, laplace=0, adjust=0 Error in density.default(x, na.rm = TRUE, ...) : 'bw' is not positive.

## Warning: model fit failed for Fold1: usekernel= TRUE, laplace=3, adjust=0 Error in density.default(x, na.rm = TRUE, ...) : 'bw' is not positive.

## Warning: model fit failed for Fold1: usekernel= TRUE, laplace=6, adjust=0 Error in density.default(x, na.rm = TRUE, ...) : 'bw' is not positive.

## Warning: model fit failed for Fold2: usekernel= TRUE, laplace=0, adjust=0 Error in density.default(x, na.rm = TRUE, ...) : 'bw' is not positive.

## Warning: model fit failed for Fold2: usekernel= TRUE, laplace=3, adjust=0 Error in density.default(x, na.rm = TRUE, ...) : 'bw' is not positive.

## Warning: model fit failed for Fold2: usekernel= TRUE, laplace=6, adjust=0 Error in density.default(x, na.rm = TRUE, ...) : 'bw' is not positive.

## Warning: model fit failed for Fold3: usekernel= TRUE, laplace=0, adjust=0 Error in density.default(x, na.rm = TRUE, ...) : 'bw' is not positive.

## Warning: model fit failed for Fold3: usekernel= TRUE, laplace=3, adjust=0 Error in density.default(x, na.rm = TRUE, ...) : 'bw' is not positive.

## Warning: model fit failed for Fold3: usekernel= TRUE, laplace=6, adjust=0 Error in density.default(x, na.rm = TRUE, ...) : 'bw' is not positive.

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :  
## There were missing values in resampled performance measures.

## Warning in train.default(data[, -1], data$label, "naive\_bayes", trControl =  
## train\_control, : missing values found in aggregated results

end\_time <- Sys.time()  
end\_time - start\_time

## Time difference of 2.196884 mins

print(naive\_bayes)

## Naive Bayes   
##   
## 4198 samples  
## 784 predictor  
## 10 classes: 'Eight', 'Five', 'Four', 'Nine', 'One', 'Seven', 'Six', 'Three', 'Two', 'Zero'   
##   
## No pre-processing  
## Resampling: Cross-Validated (3 fold)   
## Summary of sample sizes: 2800, 2798, 2798   
## Resampling results across tuning parameters:  
##   
## usekernel laplace adjust Accuracy Kappa Mean\_F1   
## FALSE 0 0 0.4571197 0.39736948 0.3873496  
## FALSE 0 3 0.4571197 0.39736948 0.3873496  
## FALSE 0 6 0.4571197 0.39736948 0.3873496  
## FALSE 3 0 0.4571197 0.39736948 0.3873496  
## FALSE 3 3 0.4571197 0.39736948 0.3873496  
## FALSE 3 6 0.4571197 0.39736948 0.3873496  
## FALSE 6 0 0.4571197 0.39736948 0.3873496  
## FALSE 6 3 0.4571197 0.39736948 0.3873496  
## FALSE 6 6 0.4571197 0.39736948 0.3873496  
## TRUE 0 0 NaN NaN NaN  
## TRUE 0 3 0.2482213 0.15691226 NaN  
## TRUE 0 6 0.1986740 0.09973676 NaN  
## TRUE 3 0 NaN NaN NaN  
## TRUE 3 3 0.2482213 0.15691226 NaN  
## TRUE 3 6 0.1986740 0.09973676 NaN  
## TRUE 6 0 NaN NaN NaN  
## TRUE 6 3 0.2482213 0.15691226 NaN  
## TRUE 6 6 0.1986740 0.09973676 NaN  
## Mean\_Sensitivity Mean\_Specificity Mean\_Pos\_Pred\_Value Mean\_Neg\_Pred\_Value  
## 0.4550308 0.9398379 0.623333 0.9432079   
## 0.4550308 0.9398379 0.623333 0.9432079   
## 0.4550308 0.9398379 0.623333 0.9432079   
## 0.4550308 0.9398379 0.623333 0.9432079   
## 0.4550308 0.9398379 0.623333 0.9432079   
## 0.4550308 0.9398379 0.623333 0.9432079   
## 0.4550308 0.9398379 0.623333 0.9432079   
## 0.4550308 0.9398379 0.623333 0.9432079   
## 0.4550308 0.9398379 0.623333 0.9432079   
## NaN NaN NaN NaN   
## 0.2401680 0.9155543 NaN 0.9236293   
## 0.1887362 0.9098712 NaN 0.9190020   
## NaN NaN NaN NaN   
## 0.2401680 0.9155543 NaN 0.9236293   
## 0.1887362 0.9098712 NaN 0.9190020   
## NaN NaN NaN NaN   
## 0.2401680 0.9155543 NaN 0.9236293   
## 0.1887362 0.9098712 NaN 0.9190020   
## Mean\_Precision Mean\_Recall Mean\_Detection\_Rate Mean\_Balanced\_Accuracy  
## 0.623333 0.4550308 0.04571197 0.6974344   
## 0.623333 0.4550308 0.04571197 0.6974344   
## 0.623333 0.4550308 0.04571197 0.6974344   
## 0.623333 0.4550308 0.04571197 0.6974344   
## 0.623333 0.4550308 0.04571197 0.6974344   
## 0.623333 0.4550308 0.04571197 0.6974344   
## 0.623333 0.4550308 0.04571197 0.6974344   
## 0.623333 0.4550308 0.04571197 0.6974344   
## 0.623333 0.4550308 0.04571197 0.6974344   
## NaN NaN NaN NaN   
## NaN 0.2401680 0.02482213 0.5778612   
## NaN 0.1887362 0.01986740 0.5493037   
## NaN NaN NaN NaN   
## NaN 0.2401680 0.02482213 0.5778612   
## NaN 0.1887362 0.01986740 0.5493037   
## NaN NaN NaN NaN   
## NaN 0.2401680 0.02482213 0.5778612   
## NaN 0.1887362 0.01986740 0.5493037   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were laplace = 0, usekernel = FALSE  
## and adjust = 0.

This algorithm takes more time than the previous one.

The best combination of the parameters we get is usekernel=FALSE, laplace=0, adjust=0. The resulting metrices we get are:   
Accuracy = 0.4571197   
Kappa = 0.3973648   
Mean\_F1= 0.3873496

The time taken to train the model is 2.196884 mins. We are training on 18 different combinations of hyperparameters compared to 30 of the Decision Tree. Despite having a lower number of combinations it takes much more time.

Now lets see how it performs on the test dataset.

start\_time <- Sys.time()  
prediction=predict(naive\_bayes,test[,-1])  
end\_time <- Sys.time()  
end\_time - start\_time

## Time difference of 3.364846 secs

confusionMatrix(table(test$label,prediction))

## Confusion Matrix and Statistics  
##   
## prediction  
## Eight Five Four Nine One Seven Six Three Two Zero  
## Eight 144 7 2 93 106 0 5 1 1 34  
## Five 117 25 1 53 33 0 10 3 4 142  
## Four 38 7 10 294 11 0 16 2 0 26  
## Nine 6 1 1 350 18 2 0 0 1 7  
## One 2 0 0 4 464 0 4 0 1 3  
## Seven 10 2 2 322 25 71 3 2 0 28  
## Six 8 1 1 30 10 0 332 0 0 22  
## Three 74 0 1 59 67 1 16 72 6 150  
## Two 75 3 1 15 27 1 112 18 81 87  
## Zero 1 1 0 10 3 0 0 0 2 397  
##   
## Overall Statistics  
##   
## Accuracy : 0.4636   
## 95% CI : (0.4484, 0.4788)  
## No Information Rate : 0.293   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4047   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: Eight Class: Five Class: Four Class: Nine  
## Sensitivity 0.30316 0.531915 0.526316 0.28455  
## Specificity 0.93312 0.912551 0.905719 0.98787  
## Pos Pred Value 0.36641 0.064433 0.024752 0.90674  
## Neg Pred Value 0.91301 0.994226 0.997628 0.76915  
## Prevalence 0.11315 0.011196 0.004526 0.29300  
## Detection Rate 0.03430 0.005955 0.002382 0.08337  
## Detection Prevalence 0.09362 0.092425 0.096236 0.09195  
## Balanced Accuracy 0.61814 0.722233 0.716017 0.63621  
## Class: One Class: Seven Class: Six Class: Three Class: Two  
## Sensitivity 0.6073 0.94667 0.66667 0.73469 0.84375  
## Specificity 0.9959 0.90444 0.98054 0.90878 0.91736  
## Pos Pred Value 0.9707 0.15269 0.82178 0.16143 0.19286  
## Neg Pred Value 0.9194 0.99893 0.95625 0.99307 0.99603  
## Prevalence 0.1820 0.01787 0.11863 0.02334 0.02287  
## Detection Rate 0.1105 0.01691 0.07909 0.01715 0.01929  
## Detection Prevalence 0.1139 0.11077 0.09624 0.10624 0.10005  
## Balanced Accuracy 0.8016 0.92555 0.82360 0.82174 0.88055  
## Class: Zero  
## Sensitivity 0.44308  
## Specificity 0.99485  
## Pos Pred Value 0.95894  
## Neg Pred Value 0.86813  
## Prevalence 0.21343  
## Detection Rate 0.09457  
## Detection Prevalence 0.09862  
## Balanced Accuracy 0.71897

The Accuracy on the test dataset is 0.4636, and the Kappa is 0.4047. As we can see the Decision tree performs better on the test dataset as well. This was somewhat expected based on the results from the k-fold cross validation.

The time to predict is 3.3648 seconds.

## 

## Algorithm Performance Comparison

|  |  |  |
| --- | --- | --- |
|  | **Decision Tree** | **Naïve Bayes** |
| Cross Validation-Accuracy | 0.7291591 | 0.4571197 |
| Cross Validation-Kappa | 0.6989871 | 0.3973695 |
| Cross Validation-Mean F-1 | 0.7270520 | 0.3873496 |
| Test Accuracy | 0.7306 | 0.4636 |
| Test Kappa | 0.7005 | 0.4047 |
| Time to Train | 31.90751 seconds | 2.1968 mins |
| Time to predict Test set | 1.0082 seconds | 3.3648 seconds |

As we can see, the Decision Tree performs better in every aspect i.e Accuracy as well as Time. One possible reason the Decision Tree works better is that it makes splits on the important features only. Where as the Naïve Bayes considers every feature.