Decision Tree v/s Naïve Bayes

Sudhanshu Kulkarni

# Introduction

The data set comes from the Kaggle Digit Recognizer competition. The goal is to recognize digits 0 to 9 in handwriting images by constructing the prediction models using Naïve Bayes and decision tree algorithms.

## Importing Libraries

First, we begin with installing and importing libraries. We have installed and imported the “dplyr” library to help us with sampling the data. Also, we have imported the “RWeka” to build prediction models using the Naïve Bayes and decision tree algorithms.

library(RWeka)

## Warning: package 'RWeka' was built under R version 3.5.3

library(dplyr)

## Warning: package 'dplyr' was built under R version 3.5.2

##   
## 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

## 

## Loading Data

Then, we load the csv data into two data sets i.e. trainData and testData to train and predict respectively. Since the original data set is too large to be loaded, I have randomly sampled 1400 observations of trianData, called sampleTrainData, and 1000 observations of the testData, called sampleTestData. We then check the structures to know if any data preprocessing is required.

setwd("C:\\Sudhanshu\\SU\\Semester 2\\707\\Assignment 6\\")  
trainData <- read.csv("Kaggle-digit-train.csv")  
sampleTrainData <- sample\_n(trainData,1400)  
View(sampleTrainData)  
str(sampleTrainData)

## 'data.frame': 1400 obs. of 785 variables:  
## $ label : int 7 6 7 1 0 1 8 6 2 4 ...  
## $ pixel0 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel1 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel2 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel3 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel4 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel5 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel6 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel7 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel8 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel9 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel10 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel11 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel12 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel13 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel14 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel15 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel16 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel17 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel18 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel19 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel20 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel21 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel22 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel23 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel24 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel25 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel26 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel27 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel28 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel29 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel30 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel31 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel32 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel33 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel34 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel35 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel36 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel37 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel38 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel39 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel40 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel41 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel42 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel43 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel44 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel45 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel46 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel47 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel48 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel49 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel50 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel51 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel52 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel53 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel54 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel55 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel56 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel57 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel58 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel59 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel60 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel61 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel62 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel63 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel64 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel65 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel66 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel67 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel68 : int 0 0 0 0 0 0 0 128 0 0 ...  
## $ pixel69 : int 0 0 0 0 0 0 0 255 0 0 ...  
## $ pixel70 : int 0 0 0 0 0 0 0 191 0 0 ...  
## $ pixel71 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel72 : int 0 74 0 0 0 0 0 0 0 0 ...  
## $ pixel73 : int 0 234 0 0 0 0 0 0 0 0 ...  
## $ pixel74 : int 0 254 0 0 0 0 0 0 0 0 ...  
## $ pixel75 : int 0 78 0 0 0 0 0 0 0 0 ...  
## $ pixel76 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel77 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel78 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel79 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel80 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel81 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel82 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel83 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel84 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel85 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel86 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel87 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel88 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel89 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel90 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel91 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel92 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel93 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel94 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel95 : int 0 0 0 0 0 0 0 0 4 0 ...  
## $ pixel96 : int 0 0 0 0 0 0 0 191 46 0 ...  
## $ pixel97 : int 0 0 0 0 0 0 0 255 145 0 ...  
## [list output truncated]

testData <- read.csv("Kaggle-digit-test.csv")  
sampleTestData <- sample\_n(testData,1000)  
View(sampleTestData)  
str(sampleTestData)

## 'data.frame': 1000 obs. of 785 variables:  
## $ label : Factor w/ 1 level "?": 1 1 1 1 1 1 1 1 1 1 ...  
## $ pixel0 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel1 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel2 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel3 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel4 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel5 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel6 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel7 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel8 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel9 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel10 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel11 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel12 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel13 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel14 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel15 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel16 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel17 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel18 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel19 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel20 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel21 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel22 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel23 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel24 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel25 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel26 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel27 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel28 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel29 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel30 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel31 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel32 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel33 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel34 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel35 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel36 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel37 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel38 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel39 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel40 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel41 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel42 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel43 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel44 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel45 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel46 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel47 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel48 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel49 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel50 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel51 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel52 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel53 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel54 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel55 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel56 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel57 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel58 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel59 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel60 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel61 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel62 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel63 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel64 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel65 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel66 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel67 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel68 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel69 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel70 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel71 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel72 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel73 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel74 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel75 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel76 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel77 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel78 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel79 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel80 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel81 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel82 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel83 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel84 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel85 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel86 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel87 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel88 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel89 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel90 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel91 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel92 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel93 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel94 : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ pixel95 : int 0 42 0 0 0 0 0 0 0 0 ...  
## $ pixel96 : int 0 73 0 0 0 0 0 0 0 0 ...  
## $ pixel97 : int 0 52 0 0 0 0 0 0 0 0 ...  
## [list output truncated]

## 

## Data Preprocessing

Now, we will convert numeric data type to nominal data type using the RWeka filter interface as the decision tree and the Naïve Bayes algorithm cannot handle data in numeric format.

NN <- make\_Weka\_filter("weka/filters/unsupervised/attribute/NumericToNominal")   
sampleTrainData <- NN(data=sampleTrainData, control= Weka\_control(R="1-3"), na.action = NULL)  
sampleTestData <- NN(data=sampleTestData, control= Weka\_control(R="1,3"), na.action = NULL)

# Decision Tree

We then build the decision tree using the J48 algorithm.

treeModel=J48(label~., data = sampleTrainData)  
treeModel=J48(label~., data = sampleTrainData, control=Weka\_control(U=FALSE, M=2,C=0.5))

Then, we use 3 fold cross-validation to evaluate the model accuracy.

evaluationModel3 <- evaluate\_Weka\_classifier(treeModel,numFolds = 3,seed = 1, class = TRUE)  
evaluationModel3

## === 3 Fold Cross Validation ===  
##   
## === Summary ===  
##   
## Correctly Classified Instances 946 67.5714 %  
## Incorrectly Classified Instances 454 32.4286 %  
## Kappa statistic 0.6393  
## Mean absolute error 0.068   
## Root mean squared error 0.2418  
## Relative absolute error 37.8143 %  
## Root relative squared error 80.6435 %  
## Total Number of Instances 1400   
##   
## === Detailed Accuracy By Class ===  
##   
## TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class  
## 0.827 0.026 0.761 0.827 0.792 0.772 0.912 0.704 0  
## 0.906 0.021 0.847 0.906 0.875 0.859 0.955 0.846 1  
## 0.652 0.036 0.662 0.652 0.657 0.620 0.826 0.515 2  
## 0.638 0.049 0.609 0.638 0.623 0.577 0.815 0.517 3  
## 0.622 0.044 0.585 0.622 0.603 0.562 0.815 0.459 4  
## 0.508 0.049 0.512 0.508 0.510 0.461 0.754 0.356 5  
## 0.728 0.024 0.793 0.728 0.759 0.731 0.879 0.697 6  
## 0.731 0.035 0.727 0.731 0.729 0.694 0.865 0.619 7  
## 0.512 0.042 0.550 0.512 0.530 0.485 0.750 0.336 8  
## 0.563 0.034 0.626 0.563 0.593 0.555 0.805 0.494 9  
## Weighted Avg. 0.676 0.036 0.674 0.676 0.674 0.639 0.841 0.564   
##   
## === Confusion Matrix ===  
##   
## a b c d e f g h i j <-- classified as  
## 105 0 3 3 1 3 2 3 5 2 | a = 0  
## 0 144 2 6 0 2 0 2 2 1 | b = 1  
## 8 6 88 11 0 3 9 4 6 0 | c = 2  
## 2 5 5 95 7 15 4 4 10 2 | d = 3  
## 2 3 0 3 79 14 8 5 3 10 | e = 4  
## 7 0 3 18 9 65 1 8 13 4 | f = 5  
## 9 1 9 6 6 6 115 1 5 0 | g = 6  
## 4 2 9 3 6 3 0 117 3 13 | h = 7  
## 0 8 12 7 8 8 4 5 66 11 | i = 8  
## 1 1 2 4 19 8 2 12 7 72 | j = 9

Here, we get 67.57% accuracy.

Similarly, we use 10 fold cross-validation to evaluate the model accuracy.

evaluationModel10 <- evaluate\_Weka\_classifier(treeModel,numFolds = 10,seed = 1, class = TRUE)  
evaluationModel10

## === 10 Fold Cross Validation ===  
##   
## === Summary ===  
##   
## Correctly Classified Instances 991 70.7857 %  
## Incorrectly Classified Instances 409 29.2143 %  
## Kappa statistic 0.675   
## Mean absolute error 0.0622  
## Root mean squared error 0.23   
## Relative absolute error 34.616 %  
## Root relative squared error 76.7088 %  
## Total Number of Instances 1400   
##   
## === Detailed Accuracy By Class ===  
##   
## TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class  
## 0.795 0.035 0.697 0.795 0.743 0.717 0.884 0.630 0  
## 0.862 0.023 0.830 0.862 0.846 0.826 0.930 0.743 1  
## 0.689 0.036 0.669 0.689 0.679 0.644 0.840 0.531 2  
## 0.617 0.046 0.617 0.617 0.617 0.572 0.786 0.506 3  
## 0.646 0.034 0.656 0.646 0.651 0.616 0.844 0.514 4  
## 0.578 0.042 0.583 0.578 0.580 0.538 0.785 0.429 5  
## 0.753 0.023 0.804 0.753 0.778 0.751 0.884 0.702 6  
## 0.838 0.023 0.822 0.838 0.830 0.808 0.913 0.647 7  
## 0.574 0.035 0.622 0.574 0.597 0.558 0.790 0.423 8  
## 0.664 0.028 0.708 0.664 0.685 0.655 0.827 0.474 9  
## Weighted Avg. 0.708 0.032 0.707 0.708 0.707 0.675 0.851 0.568   
##   
## === Confusion Matrix ===  
##   
## a b c d e f g h i j <-- classified as  
## 101 1 6 4 1 6 5 0 2 1 | a = 0  
## 1 137 8 2 0 0 3 5 3 0 | b = 1  
## 5 7 93 3 0 5 6 3 12 1 | c = 2  
## 10 3 8 92 5 12 4 1 10 4 | d = 3  
## 3 3 1 9 82 7 0 5 1 16 | e = 4  
## 8 2 4 16 8 74 4 2 7 3 | f = 5  
## 9 3 5 4 6 6 119 0 5 1 | g = 6  
## 2 2 4 4 1 0 1 134 4 8 | h = 7  
## 4 6 7 10 8 12 6 1 74 1 | i = 8  
## 2 1 3 5 14 5 0 12 1 85 | j = 9

Here, we get 67.57% accuracy.

Now, we will apply the model on the test dataset to predict the values of unknown data.

pred=predict (treeModel, newdata = sampleTestData, type = c("class"))  
pred

## [1] 8 2 0 9 8 1 9 4 3 1 9 9 2 6 4 5 7 9 8 4 4 3 4 9 2 2 5 6 8 3 6 1 3 0  
## [35] 0 6 9 6 9 9 7 1 1 1 0 4 7 3 4 0 2 6 6 1 7 9 1 0 0 5 6 4 6 5 3 0 5 4  
## [69] 2 4 1 5 1 4 0 1 9 4 3 9 3 4 8 1 5 0 4 5 1 7 2 4 0 0 4 3 6 4 8 4 6 2  
## [103] 4 8 1 2 8 1 1 3 6 1 7 0 9 7 6 4 9 2 2 5 3 9 2 2 9 2 2 4 7 5 4 3 6 3  
## [137] 5 2 6 8 8 1 8 4 5 3 3 2 6 2 6 6 1 1 3 1 3 9 6 2 0 7 1 2 6 4 2 1 9 2  
## [171] 9 9 2 9 0 0 8 8 6 2 0 0 0 6 2 4 4 3 8 7 3 1 8 7 2 5 7 3 0 9 1 5 1 7  
## [205] 8 9 3 3 9 5 6 6 5 4 3 3 1 3 5 1 9 0 9 7 2 8 3 7 9 4 1 1 1 2 3 9 5 1  
## [239] 5 9 7 0 5 7 3 9 6 3 9 6 1 7 3 3 1 4 7 9 0 9 4 7 9 7 1 9 1 7 4 5 1 7  
## [273] 6 4 0 3 3 2 7 3 8 9 9 6 5 7 0 3 7 2 6 1 4 9 9 1 4 3 0 8 1 7 1 5 6 2  
## [307] 9 2 3 9 4 0 6 3 9 0 1 5 1 1 8 9 4 8 6 6 2 7 4 4 9 6 3 1 1 1 0 2 3 1  
## [341] 2 3 3 8 0 0 5 9 6 3 4 8 7 8 2 5 6 1 9 9 3 4 2 0 6 7 2 2 4 3 4 6 2 5  
## [375] 9 1 9 7 0 2 8 3 3 8 3 9 7 9 8 6 9 2 1 6 2 8 3 1 1 5 0 0 9 2 1 1 3 1  
## [409] 2 7 6 4 5 4 5 1 1 2 3 8 2 3 3 0 4 8 6 7 0 1 4 1 9 7 9 8 8 8 1 8 3 0  
## [443] 2 8 9 9 1 4 9 3 2 1 7 2 4 4 7 4 0 0 2 9 2 5 7 4 7 5 5 7 4 2 1 6 5 8  
## [477] 6 4 6 6 4 2 3 6 3 5 7 3 1 2 1 3 0 3 7 3 8 6 2 5 8 7 4 0 5 9 5 2 9 6  
## [511] 6 1 3 2 6 6 7 7 2 3 0 1 6 2 9 7 5 1 5 0 0 5 5 8 0 2 1 5 1 5 1 6 9 4  
## [545] 9 1 6 1 1 1 1 1 1 1 0 3 4 8 3 9 6 1 7 5 8 2 7 5 3 6 5 8 9 0 1 0 1 9  
## [579] 0 6 1 0 2 8 7 9 6 0 9 2 9 5 1 5 7 0 1 1 0 1 7 5 9 9 0 4 2 6 3 2 4 2  
## [613] 9 3 7 8 2 4 3 1 4 4 7 6 6 2 4 8 3 9 6 6 2 3 1 3 8 7 7 9 0 3 7 4 9 6  
## [647] 4 4 0 8 1 3 1 8 1 1 7 2 1 8 7 9 4 2 9 9 9 5 7 0 3 4 1 6 6 6 6 2 2 2  
## [681] 9 9 7 8 8 3 3 2 5 1 6 8 9 9 0 7 3 4 2 5 1 0 7 0 5 4 5 9 8 0 1 9 9 8  
## [715] 3 5 8 4 5 8 4 5 3 6 2 5 7 2 5 6 4 6 6 2 7 8 8 5 1 6 0 5 2 2 8 8 6 5  
## [749] 2 3 6 5 3 2 7 1 9 8 5 1 8 0 8 8 5 4 5 6 6 7 7 6 3 1 1 1 4 7 6 9 7 4  
## [783] 3 9 1 2 0 9 1 3 5 2 4 1 8 2 0 3 3 6 0 3 3 5 3 7 9 2 6 9 8 6 7 7 2 7  
## [817] 5 1 3 0 8 3 2 3 4 8 0 5 4 0 0 4 2 0 5 1 1 5 3 8 1 1 1 3 5 9 5 9 1 4  
## [851] 0 8 1 4 9 1 9 3 3 3 3 3 7 9 5 1 5 6 6 7 0 3 2 2 4 0 6 9 4 6 6 5 8 9  
## [885] 8 7 0 3 9 9 1 3 9 8 4 0 6 1 7 9 5 3 8 2 3 1 5 2 2 3 1 2 2 8 7 7 9 4  
## [919] 7 9 5 8 4 5 4 0 2 3 3 8 2 3 7 7 2 7 0 6 5 6 5 9 8 8 0 2 6 5 2 4 3 6  
## [953] 9 2 4 9 3 0 4 1 4 1 6 4 3 3 0 0 2 6 4 1 3 4 6 9 1 1 2 6 8 7 0 2 8 4  
## [987] 6 0 0 2 9 7 1 6 8 0 0 3 2 4  
## Levels: 0 1 2 3 4 5 6 7 8 9

# Naïve Bayes

We then build the Naïve Bayes using the RWeka interface.

NB <- make\_Weka\_classifier("weka/classifiers/bayes/NaiveBayes")  
nb\_model=NB(label~., data=sampleTrainData)

Then, we use 3 fold cross-validation to evaluate the model accuracy.

evaluationModelNB3 <- evaluate\_Weka\_classifier(nb\_model, numFolds = 3, seed = 1, class = TRUE)  
evaluationModelNB3

## === 3 Fold Cross Validation ===  
##   
## === Summary ===  
##   
## Correctly Classified Instances 996 71.1429 %  
## Incorrectly Classified Instances 404 28.8571 %  
## Kappa statistic 0.6792  
## Mean absolute error 0.0577  
## Root mean squared error 0.2392  
## Relative absolute error 32.0798 %  
## Root relative squared error 79.7619 %  
## Total Number of Instances 1400   
##   
## === Detailed Accuracy By Class ===  
##   
## TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class  
## 0.764 0.014 0.843 0.764 0.802 0.784 0.944 0.774 0  
## 0.931 0.012 0.908 0.931 0.919 0.909 0.969 0.872 1  
## 0.659 0.024 0.748 0.659 0.701 0.673 0.912 0.656 2  
## 0.631 0.025 0.752 0.631 0.686 0.655 0.886 0.652 3  
## 0.512 0.020 0.722 0.512 0.599 0.576 0.883 0.561 4  
## 0.523 0.032 0.620 0.523 0.568 0.531 0.890 0.512 5  
## 0.886 0.028 0.800 0.886 0.841 0.821 0.949 0.780 6  
## 0.656 0.016 0.840 0.656 0.737 0.714 0.923 0.746 7  
## 0.698 0.079 0.474 0.698 0.564 0.523 0.878 0.488 8  
## 0.789 0.070 0.532 0.789 0.635 0.605 0.903 0.510 9  
## Weighted Avg. 0.711 0.031 0.733 0.711 0.713 0.688 0.916 0.664   
##   
## === Confusion Matrix ===  
##   
## a b c d e f g h i j <-- classified as  
## 97 0 5 0 0 2 8 0 13 2 | a = 0  
## 0 148 0 2 0 2 1 0 5 1 | b = 1  
## 5 2 89 8 0 5 10 0 15 1 | c = 2  
## 4 2 13 94 3 4 4 3 15 7 | d = 3  
## 1 2 1 1 65 9 8 5 10 25 | e = 4  
## 6 2 2 9 2 67 3 1 32 4 | f = 5  
## 1 0 4 2 0 5 140 0 6 0 | g = 6  
## 0 0 2 3 6 1 0 105 1 42 | h = 7  
## 1 6 2 6 4 11 1 1 90 7 | i = 8  
## 0 1 1 0 10 2 0 10 3 101 | j = 9

Here, we get 71.14% accuracy.

Similarly, we use 10 fold cross-validation to evaluate the model accuracy.

evaluationModelNB10 <- evaluate\_Weka\_classifier(nb\_model, numFolds = 10, seed = 1, class = TRUE)  
evaluationModelNB10

## === 10 Fold Cross Validation ===  
##   
## === Summary ===  
##   
## Correctly Classified Instances 999 71.3571 %  
## Incorrectly Classified Instances 401 28.6429 %  
## Kappa statistic 0.6816  
## Mean absolute error 0.0573  
## Root mean squared error 0.2385  
## Relative absolute error 31.8547 %  
## Root relative squared error 79.5425 %  
## Total Number of Instances 1400   
##   
## === Detailed Accuracy By Class ===  
##   
## TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class  
## 0.748 0.012 0.864 0.748 0.802 0.786 0.964 0.833 0  
## 0.931 0.015 0.892 0.931 0.911 0.899 0.961 0.848 1  
## 0.704 0.026 0.742 0.704 0.722 0.694 0.905 0.652 2  
## 0.644 0.022 0.780 0.644 0.706 0.678 0.886 0.650 3  
## 0.543 0.016 0.767 0.543 0.636 0.617 0.916 0.652 4  
## 0.484 0.031 0.614 0.484 0.541 0.505 0.884 0.511 5  
## 0.886 0.032 0.778 0.886 0.828 0.807 0.960 0.767 6  
## 0.594 0.012 0.864 0.594 0.704 0.688 0.945 0.794 7  
## 0.682 0.072 0.489 0.682 0.570 0.527 0.879 0.480 8  
## 0.867 0.079 0.524 0.867 0.653 0.633 0.912 0.545 9  
## Weighted Avg. 0.714 0.031 0.739 0.714 0.714 0.691 0.923 0.681   
##   
## === Confusion Matrix ===  
##   
## a b c d e f g h i j <-- classified as  
## 95 0 6 0 1 2 11 1 10 1 | a = 0  
## 0 148 3 2 0 2 1 0 3 0 | b = 1  
## 3 2 95 5 0 3 13 0 13 1 | c = 2  
## 1 2 12 96 2 5 4 4 17 6 | d = 3  
## 2 1 3 1 69 8 6 3 6 28 | e = 4  
## 6 1 2 11 4 62 4 1 34 3 | f = 5  
## 1 2 4 0 1 4 140 0 6 0 | g = 6  
## 0 0 1 4 6 0 0 95 2 52 | h = 7  
## 1 9 2 3 3 12 0 1 88 10 | i = 8  
## 1 1 0 1 4 3 1 5 1 111 | j = 9

Here, we get 71.36% accuracy.

Now, we will apply the model on the test dataset to predict the values of unknown data.

predNB = predict (nb\_model, newdata = sampleTestData, type = c("class"))  
predNB

## [1] 8 2 5 0 1 1 9 9 3 1 9 9 2 6 0 8 7 9 2 0 8 5 8 5 2 2 5 6 8 2 8 1 4 0  
## [35] 8 6 9 2 9 9 8 1 1 1 0 4 9 3 9 0 2 6 6 1 7 9 1 0 0 8 8 4 6 8 7 6 8 6  
## [69] 2 8 1 5 2 9 0 1 9 7 5 9 3 9 8 8 3 0 4 7 1 7 8 9 0 0 4 3 6 9 8 5 6 8  
## [103] 4 8 1 2 8 1 1 3 8 1 3 0 9 7 6 8 8 7 4 1 7 9 8 2 9 2 2 4 7 5 4 2 6 7  
## [137] 6 2 5 8 8 1 6 4 6 3 2 6 0 1 6 1 1 1 4 1 3 9 6 2 0 7 1 8 5 4 2 1 4 2  
## [171] 9 9 5 4 0 5 7 8 3 8 5 0 0 8 6 4 4 2 8 7 3 1 9 7 2 5 9 8 0 9 1 0 1 3  
## [205] 6 5 9 6 7 3 6 2 5 6 3 9 1 4 0 1 0 8 9 3 2 1 8 7 9 4 1 1 1 4 3 9 5 1  
## [239] 3 9 9 3 4 9 2 9 6 8 4 2 1 7 0 3 1 4 7 9 5 9 8 7 8 7 1 9 1 4 4 8 1 7  
## [273] 6 6 6 4 7 2 7 0 9 9 9 6 5 9 0 3 9 2 6 1 8 9 9 1 4 8 0 6 1 7 1 5 8 2  
## [307] 9 2 2 9 4 8 6 8 9 6 1 3 1 1 5 9 3 4 6 6 2 7 0 4 7 6 3 1 1 1 0 8 3 1  
## [341] 8 8 3 8 0 0 3 9 5 8 7 8 1 5 8 8 6 1 9 9 4 4 2 0 6 9 6 2 9 2 4 6 6 8  
## [375] 7 1 9 9 0 8 0 3 5 8 3 9 7 9 0 6 9 4 1 6 1 8 3 6 1 8 0 0 9 2 1 1 3 1  
## [409] 2 7 6 8 3 9 0 1 1 8 3 9 8 3 4 0 9 8 6 7 0 1 4 8 4 9 9 8 8 8 1 8 3 0  
## [443] 3 8 9 9 1 5 9 3 9 6 7 3 2 0 7 4 0 6 8 9 2 6 3 5 7 8 6 9 5 1 1 6 5 9  
## [477] 6 4 6 6 2 2 8 6 9 9 7 0 1 8 1 3 8 3 7 0 5 6 2 3 5 7 4 0 2 9 8 2 5 6  
## [511] 6 1 8 3 6 6 7 7 1 2 0 1 6 2 9 7 2 3 5 8 0 8 2 8 6 8 1 5 1 5 1 6 4 5  
## [545] 9 1 6 1 1 1 1 1 8 1 0 3 4 1 3 9 6 1 4 2 9 2 7 9 2 6 8 8 4 5 1 0 2 9  
## [579] 0 6 1 0 3 6 7 8 6 6 9 2 9 5 1 3 7 0 1 1 8 1 9 6 5 9 8 9 2 6 3 2 6 3  
## [613] 8 8 7 8 2 4 3 1 0 4 9 6 2 3 4 9 3 8 6 6 2 8 1 2 8 9 7 5 0 3 9 4 9 6  
## [647] 4 4 0 9 1 8 1 8 1 9 9 6 1 7 7 9 4 6 9 4 9 5 9 8 3 4 1 6 6 6 6 0 2 2  
## [681] 9 9 7 8 0 3 3 2 3 1 6 8 9 9 0 7 3 4 2 5 1 8 7 0 9 6 3 8 8 0 1 4 9 8  
## [715] 2 3 8 9 2 1 4 8 3 6 6 5 9 2 4 6 9 8 6 2 7 8 8 0 1 6 0 8 2 5 6 8 6 5  
## [749] 2 7 6 8 7 2 1 1 5 8 6 1 9 0 5 8 5 4 5 6 6 9 7 2 8 1 1 1 9 7 6 9 7 8  
## [783] 3 9 1 2 4 9 1 3 8 6 4 1 8 1 0 3 6 6 2 9 2 0 3 7 9 2 2 9 1 6 7 7 5 7  
## [817] 8 1 3 6 9 2 6 3 4 8 0 2 5 5 0 9 6 0 5 1 1 8 2 8 1 1 1 5 4 9 2 9 1 2  
## [851] 9 1 1 4 9 1 9 5 3 5 1 1 7 9 8 1 8 6 6 7 0 3 6 2 4 0 8 9 8 6 6 8 8 9  
## [885] 3 7 2 5 0 9 1 3 9 7 4 6 6 1 7 4 5 3 8 5 3 1 8 3 2 8 1 2 2 8 7 7 9 9  
## [919] 0 9 0 8 5 8 6 0 2 4 6 8 2 5 9 9 2 1 0 6 8 6 5 9 9 9 0 4 6 5 6 4 3 6  
## [953] 9 2 9 9 3 0 9 1 4 7 6 4 3 5 0 6 2 6 4 1 3 9 6 9 1 1 3 6 6 9 0 8 1 4  
## [987] 6 0 2 2 9 7 1 6 8 0 8 3 8 4  
## Levels: 0 1 2 3 4 5 6 7 8 9

# Algorithm performance comparison

The below table shows us the accuracy percent of Decision Tree and Naïve Bayes algorithm.

Table 1 - Accuracy Comparison

|  |  |  |
| --- | --- | --- |
|  | 3 fold cross-validation | 10 fold cross-validation |
| Decision Tree | 67.57% | 70.79% |
| Naïve Bayes | **71.14%** | **71.36%** |

As we can see from table 1, the Naïve Bayes algorithm does a better job for image recognition as compared to Decision Tree. Also, the processing time of Naïve Bayes algorithm is better than Decision Tree as its classifiers are computationally fast when making decisions. Also, we have observed that when the training data is low the Naïve Bayes algorithm classifies data more accurately. However, with high training data, the decision tree, even though it takes more time, more accurately predicts the labels.