Joyce Woznica

jlwoznic@syr.edu

November 22, 2019

Joyce Woznica  
Homework #6

Contents

[Introduction 3](#_Toc25268198)

[Data 3](#_Toc25268199)

[Standard Data Preparation and Preprocessing 3](#_Toc25268200)

[Build and Tune Decision Tree Models 6](#_Toc25268201)

[RandomForest 6](#_Toc25268202)

[rpart Decision Trees 7](#_Toc25268203)

[Naïve Bayes 10](#_Toc25268204)

[Algorithm Performance Comparison 12](#_Toc25268205)

[Conclusion 13](#_Toc25268206)

List of Figures

[Figure 1: MNIST Dataset of Handwritten Digits Example Images 3](#_Toc25268208)

[Figure 2: Structure of Kaggle Digit Training Set 4](#_Toc25268209)

[Figure 3: Factor for the Label in the Data Set 4](#_Toc25268210)

[Figure 4: Generate a Smaller Random Sample from the Digit Dataset 5](#_Toc25268211)

[Figure 5: Generate a Smaller Random Sample from the Test Digit Dataset 5](#_Toc25268212)

[Figure 4: Building Training Sets for k-fold Cross Validation Testing (10 sets) 5](#_Toc25268213)

[Figure 7: Code for randomForest Decision Tree Model 6](#_Toc25268214)

[Figure 8: randomForest Output with Confusion Matrix 7](#_Toc25268215)

[Figure 9: randomForest Accuracy 7](#_Toc25268216)

[Figure 10: rpart Decision Tree Function 8](#_Toc25268217)

[Figure 11: k-fold Splitting of Digit Train Data 8](#_Toc25268218)

[Figure 12: Loop – Train and Prediction of 10-fold for rpart Decision Trees 9](#_Toc25268219)

[Figure 13: Sample rpart.plot Decision Tree 10](#_Toc25268220)

[Figure 14: Accuracy of Decision Tree Data Frame 10](#_Toc25268221)

[Figure 15: Naïve Bayes R-Code for 10-fold Validation 11](#_Toc25268222)

[Figure 16: Naïve Bayes Confusion Matrix and Accuracy 11](#_Toc25268223)

[Figure 17: Naïve Bayes mlr R-Code 12](#_Toc25268224)

[Figure 18: Naïve Bayes mlr Confusion Matrix and Accuracy 12](#_Toc25268225)

[Figure 19: Accuracy Comparison Table 13](#_Toc25268226)

# 

# Introduction

The Modified National Institute of Standards of Technology (MNIST) has created a dataset of classic handwritten images, released in 1999, to be used as a benchmark for classification algorithms. The goal is to correctly identify digits form the dataset of tens of thousands of handwritten images using decision trees and Naïve Bayes modeling.

An example of these images can be found in the following figure.

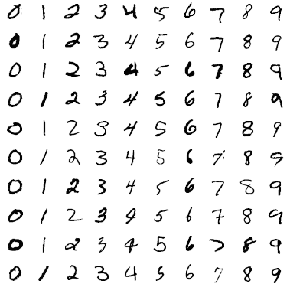
c

Figure 1: MNIST Dataset of Handwritten Digits Example Images

This dataset is a mixing of two databases – Special Database 3 (SD-3), collected from Census Bureau employees, and Special Database 1 (SD-1), collected from high school students, which are both sets of binary images of handwritten digits. The decision to mix the data was the result of originally using SD-3 to train the data and SD-1 to test it. However, SD-3 is a much cleaner dataset and easier to recognize. To make the sampling more independent, these two databases were mixed to form the MNIST dataset.

This document outlines the data collected, the loading process, any modifications required to the data and why, initial visualizations and information about the data, and the results of decision tree modeling as well as Naïve Bayes to arrive at the best model to accurately predict the classification of a 28 by 28 black and white image of a numeric digit.

# Data

## Standard Data Preparation and Preprocessing

The data files *Kaggle-digit-train.csv* and *Kaggle-digit-test.csv* contain gray-scale images of hand-drawn digits, from zero through nine. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (*Kaggle-digit-train.csv*), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Each pixel column in the training set has a name like *pixelx*, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i \* 28 + j, where i and j are integers between 0 and 27, inclusive. Then *pixelx* is located on row i and column j of a 28 x 28 matrix, (indexing by zero).

For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

The test data set, (*Kaggle-digit-test.csv*), is the same as the training set, except that it does not contain the "label" column.

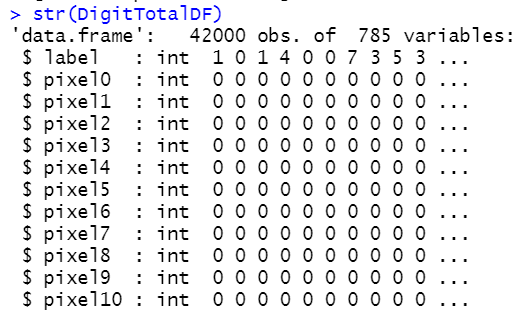


Figure 2: Structure of Kaggle Digit Training Set

It was important to change the label (the actual digit represented) to a factor for the models to be completed. This was done using the following:

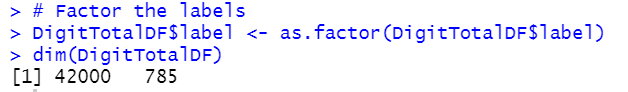
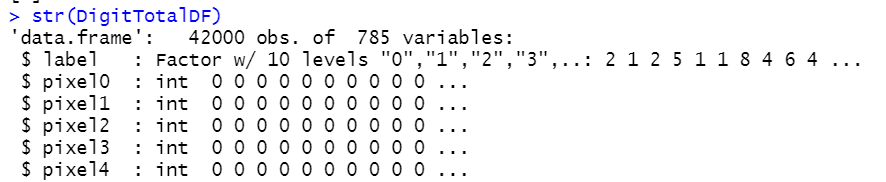
  


Figure 3: Factor for the Label in the Data Set

Due to the sheer size of this data set (42,000 observations), a sample dataset of about ¼ the size was created using the *sample* function as shown in the following figure.

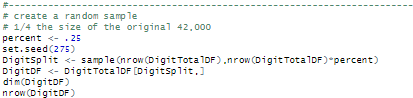
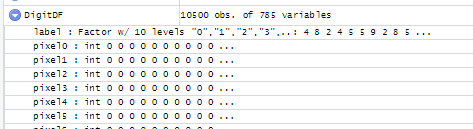
  


Figure 4: Generate a Smaller Random Sample from the Digit Dataset

This created a significantly smaller set of training data, 10,500 observations.

Following this sample, the test data was read into a data frame and a similar sample was created from this data as well.

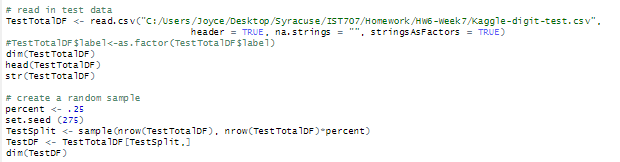
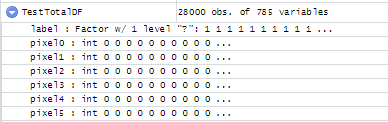
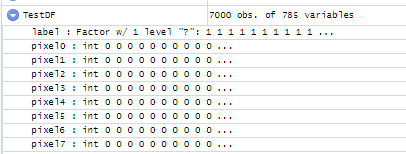
  
  


Figure 5: Generate a Smaller Random Sample from the Test Digit Dataset

For both the decision tree modeling and the Naïve Bayes modeling, the data was separated into 10 training sets for cross validation to occur. This was accomplished using the following:

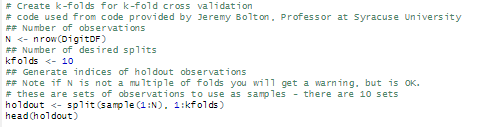


Figure 4: Building Training Sets for k-fold Cross Validation Testing (10 sets)

# Build and Tune Decision Tree Models

In order to predict the written characters, two different methods were used to build decision trees and then verify accuracy of the prediction using these trees.

## RandomForest

First, the package *randomForest* was used for creating decision trees. This package allows the selection of the number of trees to generate as well as gathers a very detailed confusion matrix with accuracy probabilities. For this model, which runs fairly quickly, the decision to create 25 trees was used. It was also important to create a test set that did not have the labels of which digit was contained in the pixel observations. This is part of the code used which can be found in the following figure.

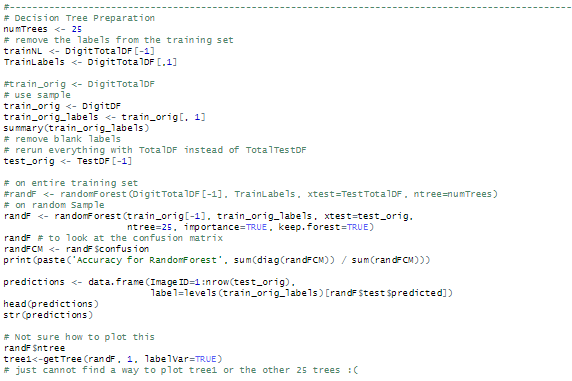


Figure 7: Code for randomForest Decision Tree Model

By providing “keep.forest=TRUE” the 25 trees are maintained within the model; however, the plotting of those trees proved overly complicated and instead the focus was on the accuracy of the predictions from the trained *randomForest* model. The final confusion matrix for this model is shown in the following figure. This is highlighted to reflect the true predictions down the diagonal.

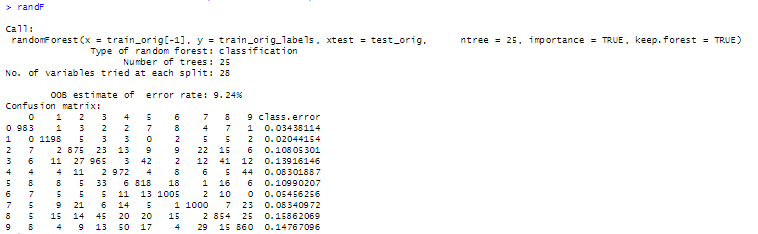
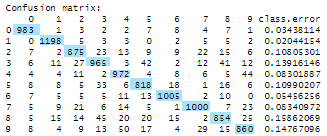
  


Figure 8: randomForest Output with Confusion Matrix

Here we notice that the error on each digit is quite low, with the highest being for the number “8” which makes sense because of the amount of black that may appear in the pixels and how this could be interpreted as a different digit.

Using the fact that the sum of the diagonal which is the true predictions, divided by total sum of all elements in the matrix, the following accuracy was found for the *randomForest* decision tree model. This accuracy number of 90.525% is very high. However, this does not reflect 3-fold cross validation which is shown in the next section using the *rpart* decision trees.



Figure 9: randomForest Accuracy

Random forests were run again with 40, 50 and 75 trees, each producing a greater level of accuracy as will be shown in the summary table at the end of this document.

## rpart Decision Trees

To attempt to simply the testing and modifications for the different *rpart* decision trees, the following function was created allow the passing of the complexity parameter, minimum number of splits, maximum number of splits and graphical palette (which was set to green/blue for this exercise).

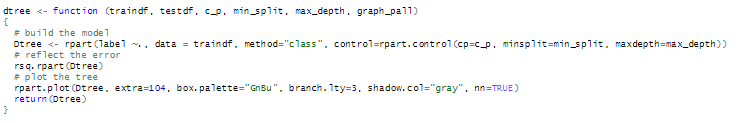


Figure 10: rpart Decision Tree Function

The data was then separated into 10 random sets to be used for k-fold splitting and cross validation.

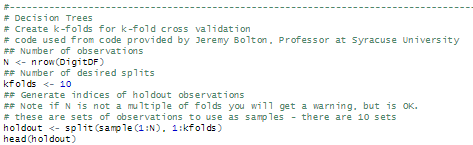


Figure 11: k-fold Splitting of Digit Train Data

The following loop was run to create a 10-fold validation of the digit data looping through each of the 10 datasets created. The accuracy results were appended to a data frame for comparison purposes. However, for this exercise, the *dtree*  function call was manually changed to the parameters shown for each tree created due to lack of time to build a proper function to make the testing easier and more streamlined.

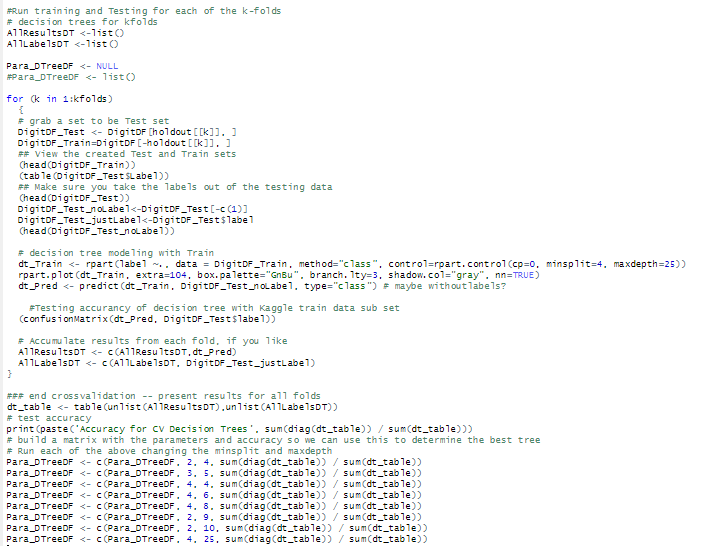


Figure 12: Loop – Train and Prediction of 10-fold for rpart Decision Trees

The building of each of these trees varied with the first iteration using 2 minimum splits and a depth of 4 taking the least amount of time. Each decision tree was graphed, but the graphs were essentially illegible. An example using the simplest tree of at least 2 splits and a depth of 4 is shown in the following figure.

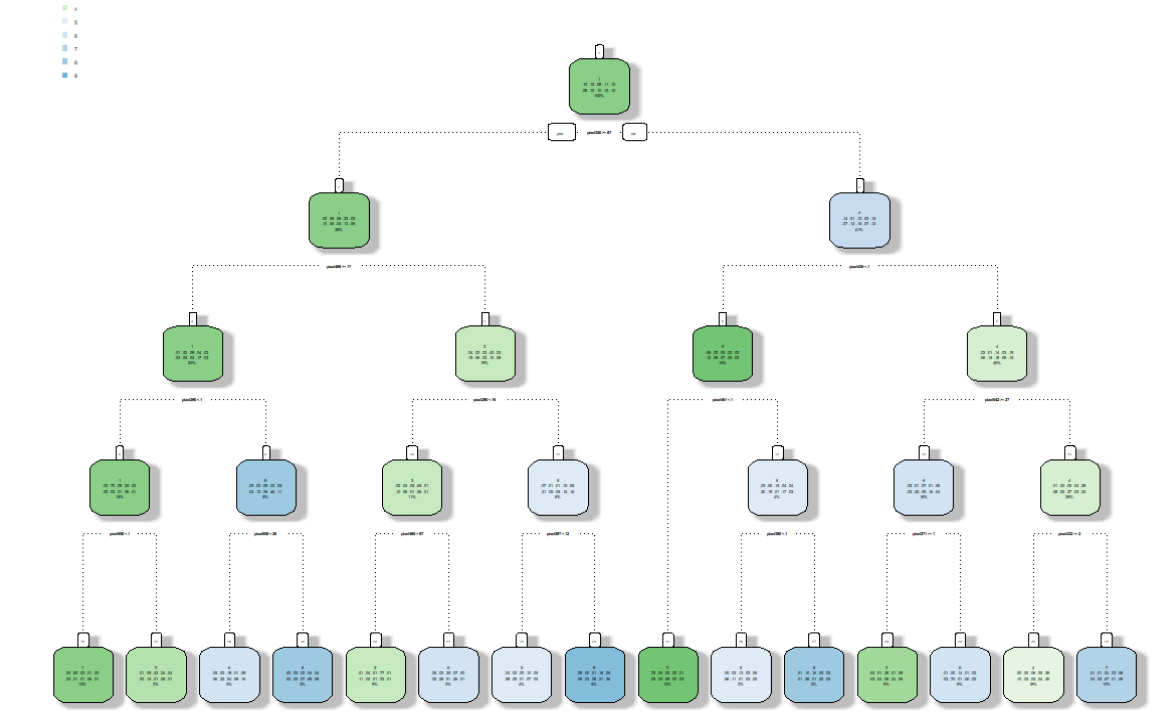


Figure 13: Sample rpart.plot Decision Tree

The results of each decision tree run for the 10-fold validation is shown in the following data frame.

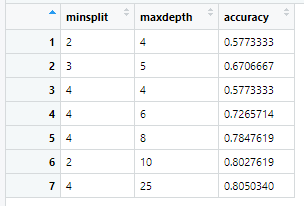


Figure 14: Accuracy of Decision Tree Data Frame

As shown in the figure, the best decision tree was the one with a minimum of 4 splits and up to 25 in depth. This provided an accuracy of just over 80%. Still not the same level of accuracy found using the *randomSplit* function.

# Naïve Bayes

A similar approach was taken to run the Naïve Bayes method for k-fold validation by splitting the training data into 10 sets of random samples. Then the accuracy was determined from the final confusion matrix for this model.

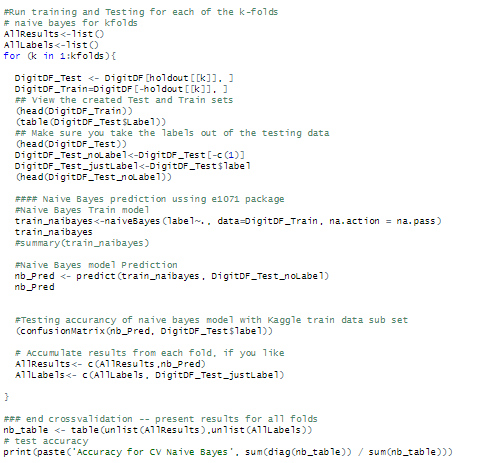


Figure 15: Naïve Bayes R-Code for 10-fold Validation

The resulting confusion matrix as a result of the Naïve Bayes modeling and the accuracy for the model is shown in the following figure.

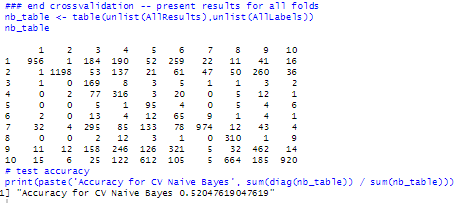


Figure 16: Naïve Bayes Confusion Matrix and Accuracy

The accuracy of this model is poor at just over 52%. Much less than expected.

Following this attempt, some research was done on other ways to run Naïve Bayes that may provide a greater level of accuracy. Using the *mlr* package, another run using Naïve Bayes was run as shown using the code below.

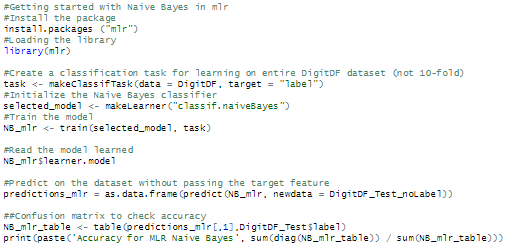


Figure 17: Naïve Bayes mlr R-Code

However, this provided the exact same level or accuracy as the first attempt at Naïve Bayes as shown below.

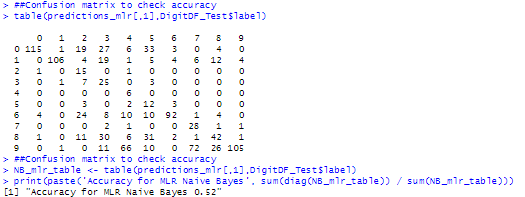


Figure 18: Naïve Bayes mlr Confusion Matrix and Accuracy

# Algorithm Performance Comparison

The models provided very different levels of accuracy and some took a significantly longer time to complete. Using the following code chunk before and after running the models provided the elapsed time for the model.

startTime <- proc.time()  
*model*proc.time() - startTime

A summary table can be found below. Keep in mind that the *randomForest* model and the final *mlr* Naïve Bayes model did not use 10-fold cross validation and this has a large impact on the elapsed run time for the code to create these models. Also, some timers included additional steps than just model creation, but it does give some idea of the time required for the model creation.

| Model Type/Constraints | Accuracy | Elapsed Time (seconds) |
| --- | --- | --- |
| Random Forest Decision Tree with 25 trees | 90.525 | 57.39 |
| Random Forest Decision Tree with 40 trees | 92.384 | 84.91 |
| Random Forest Decision Tree with 50 trees | 93.356 | 110.54 |
| Random Forest Decision Tree with 75 trees | 93.747 | 162.26 |
| Decision Tree with Minimum Splits = 2, Max Depth = 5 | 57.733 | 105.87 |
| Decision Tree with Minimum Splits = 3, Max Depth = 5 | 67.067 | 138.69 |
| Decision Tree with Minimum Splits = 4, Max Depth = 4 | 57.733 | 105.87 |
| Decision Tree with Minimum Splits = 4, Max Depth = 6 | 72.547 | 176.02 |
| Decision Tree with Minimum Splits = 4, Max Depth = 8 | 78.476 | 320.20 |
| Decision Tree with Minimum Splits = 2, Max Depth = 10 | 80.276 | 933.11 |
| Decision Tree with Minimum Splits = 4, Max Depth = 25 | 80.503 | 1385.64 |
| Naïve Bayes with 10-fold Cross Validation | 52.048 | 128.53 |
| mlr Naïve Bayes Model | 52.476 | 13.64 |

Figure 19: Accuracy Comparison Table

These results are expected given the modeling approach taken by each algorithm. Decision trees do well with large datasets, but run slower and are prone to overfitting. However, Naïve Bayes works well with small datasets and has less overfitting and runs much faster overall. Since random forests address overfitting, they tend to be more accurate.

Looking at the information in the table, clearly the *randomForest* with accuracies ranging between 90 to almost 94 percent, provides the best digit prediction. It was also one of the faster running tests. Decision Trees performed very well providing accuracies up to 80% depending on the selected parameters. However, increasing the depth of the decision tree would likely increase the accuracy of this model, but it would be very slow to calculate.

# Conclusion

For this data set due to its large volume and the performance of these models, the recommendation would be to use random forest modeling. However, this was not one of the options for this exercise, so the second recommendation would be Decision Trees with careful attention to pruning.