Joyce Woznica

jlwoznic@syr.edu

November 29, 2019

Joyce Woznica  
Homework #7

Contents

[Introduction 3](#_Toc25867672)

[Data 3](#_Toc25867673)

[Standard Data Preparation and Preprocessing 3](#_Toc25867674)

[Analysis 5](#_Toc25867675)

[Visualizations 5](#_Toc25867676)

[K Nearest Neighbor (kNN) 5](#_Toc25867677)

[Support Vector Machines 6](#_Toc25867678)

[Random Forest 9](#_Toc25867679)

[Algorithm Performance Comparison 10](#_Toc25867680)

[Conclusion 10](#_Toc25867681)

List of Figures

[Figure 1: MNIST Dataset of Handwritten Digits Example Images 3](#_Toc25867682)

[Figure 2: Structure of Kaggle Digit Training Set 4](#_Toc25867683)

[Figure 3: Factor for the Label in the Data Set 4](#_Toc25867684)

[Figure 4: Generate a Smaller Random Sample from the Digit Dataset 4](#_Toc25867685)

[Figure 5: Building Training Sets for k-fold Cross Validation Testing (2 sets) 5](#_Toc25867686)

[Figure 6: Function for Displaying Accuracy for Confusion Matrices 5](#_Toc25867687)

[Figure 7: Frequency Digit Counts 5](#_Toc25867688)

[Figure 8: kNN with k=7 6](#_Toc25867689)

[Figure 9: kNN Results Table 6](#_Toc25867690)

[Figure 10: SVM Code with Defaults and Accuracy Results 7](#_Toc25867691)

[Figure 11: SVM Code with Polynomial Kernel with Binarized Data with Accuracy Results 8](#_Toc25867692)

[Figure 12: SVM Results Table 8](#_Toc25867693)

[Figure 13: RandomForest Code with Accuracy Results 9](#_Toc25867694)

[Figure 14: RandomForest Best Number of Trees Loop 10](#_Toc25867695)

# 

# 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.

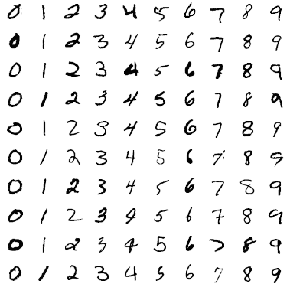
c

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.

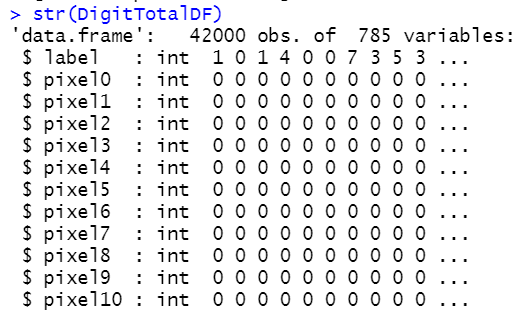


Figure 2: Structure of Kaggle Digit Training Set

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:

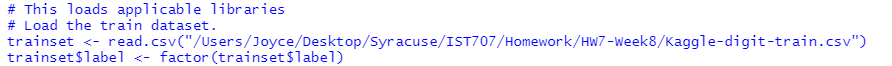
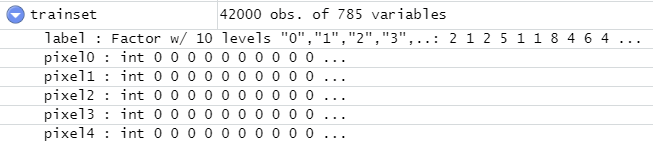
  


Figure 3: Factor for the Label in the Data Set

Due to the sheer size of this data set (42,000 observations), a sample dataset of about 1/15th of the size was created using the *sample* function as shown in the following figure.

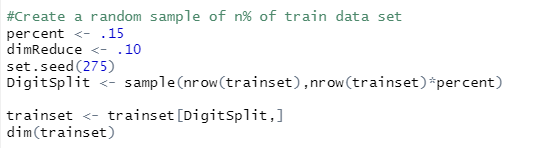


Figure 4: Generate a Smaller Random Sample from the Digit Dataset

This created a significantly smaller set of training data to 6,300 observations.

For the K Nearest Neighbors (kNN), Support Vector Machine (SVM) and *RandomForest* models, a k-fold training set was required to complete for cross validation to occur. This was accomplished using the following:

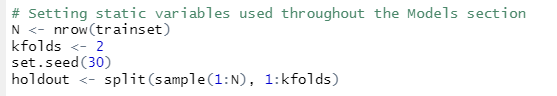


Figure 5: Building Training Sets for k-fold Cross Validation Testing (2 sets)

Finally, a function was generated to provide accuracy for the tests by summing the diagonals and dividing by the total predictions as shown in the following figure.

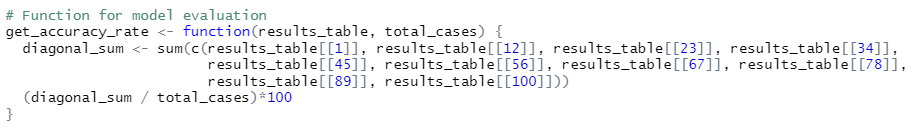


Figure 6: Function for Displaying Accuracy for Confusion Matrices

# Analysis

## Visualizations

Prior to running the models for kNN, SVM and *randomForest*, some generic plots were run on the data to look at the digits represented in the data.

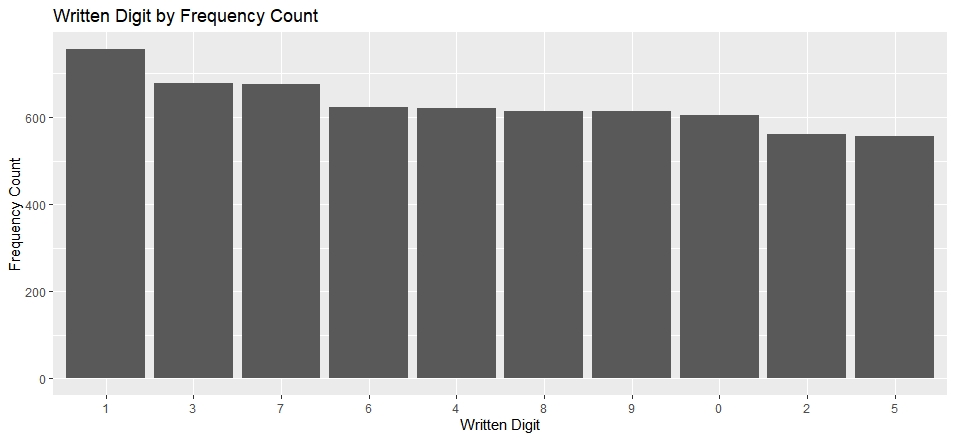


Figure 7: Frequency Digit Counts

This provides a good understanding of the dataset and the digits to be expedited in the predictions provided by the models created.

## K Nearest Neighbor (kNN)

The kNN algorithm looks for data points (also called vectors or rows) that are “closest” to the new data point. The data point is the vector of values we are trying to classify or categorize. The “close” points are called the nearest neighbors. “Closeness” is measured with any number of distance or similarity measures –such as Euclidean, Manhattan, or Cosine Similarity, for example. For the kNN model, the training set was used for creating and testing the models using 2-fold cross validation.

The initial kNN model was run using *k=7* to be a starting point for the models to be analyzed. An example of this code and results can be found in the figure below.

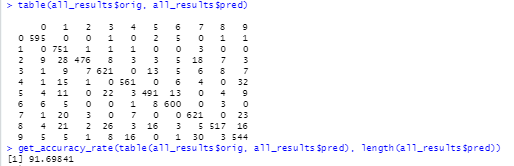
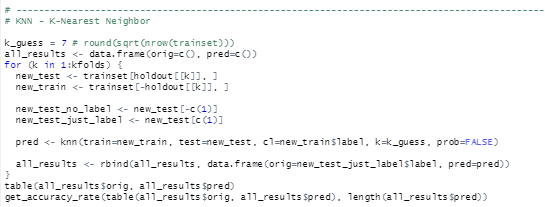


Figure 8: kNN with k=7

This initial modeling with kNN shows an accuracy rate of 91.69841. However, with the thought that this accuracy can be improved, the same code was run with k values of 3, 5 and 8. The table below reflects the accuracy for each of these runs.

| kNN Run with k Value | Accuracy |
| --- | --- |
| K Nearest Neighbor, k = 7 | 91.69841 |
| K Nearest Neighbor, k = 3 | 92.00000 |
| K Nearest Neighbor, k = 5 | 91.93651 |
| K Nearest Neighbor, k = 8 | 91.92063 |
| K Nearest Neighbor, k = 4 | 91.93651 |

Figure 9: kNN Results Table

Notice that increasing the value of kNN based on the k-values shown in the table. For this modeling, the best *k* was found to be 3 with an accuracy of 92%.

## Support Vector Machines

A Support Vector Machine (SVM) performs classification by finding the hyperplane that maximizes the margin between the two classes. The vectors (cases) that define the hyperplane are the support vectors.

Using SVM, the models can be refined by changing the kernel being used and also changing the “cost” applied for misclassification in the model.

Initially, different kernels were used with the SVM model to determine the best level of accuracy. Then the best kernel was used and the cost was modified to determine a refined model using that kernel. The code used is shown in the following figure.

However, initially, the SVM was run using mostly default configuration parameters. This was done after some rows were removed from the dataset as show in the following code.

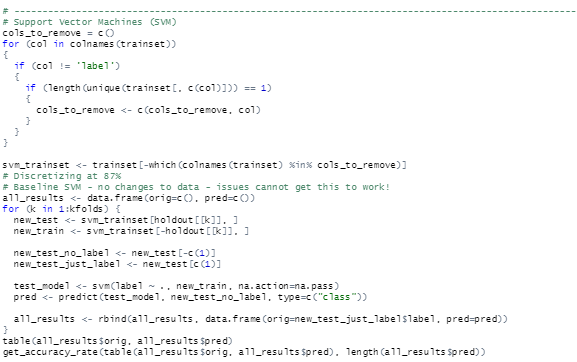
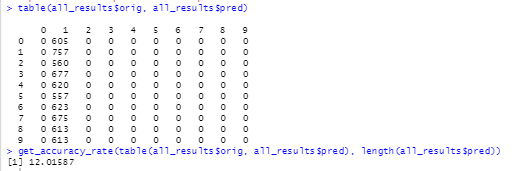
  


Figure 10: SVM Code with Defaults and Accuracy Results

As shown, this provided very poor overall accuracy results, in fact, everything was classified as a digit “1”. A test was the run by binarizing the dataset as shown in the figure below.

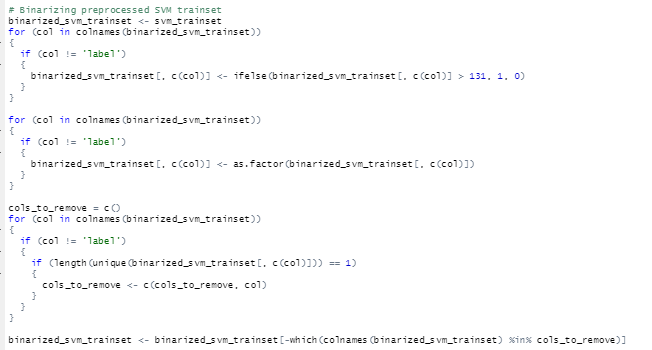
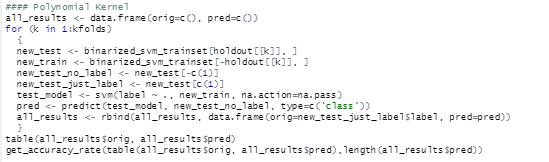
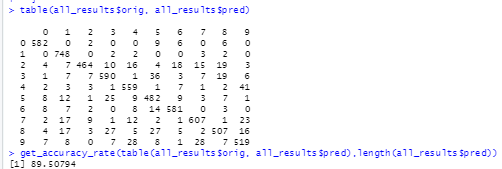
   


Figure 11: SVM Code with Polynomial Kernel with Binarized Data with Accuracy Results

With the polynomial kernel, binarized data, the accuracy was significantly improved to 89.50794. Similar code was run and completed changing the kernel and the cost figures. A summary of this information can be found in the following table.

| Model Type/Constraints | Kernel | Cost | Accuracy |
| --- | --- | --- | --- |
| SVM Model | Default | 1 | 12.01587 |
| SVM Model, binarized data | Radial (default) | 1 | 89.50794 |
| SVM Model, binarized data | Polynomial | 1 | 13.26984 |
| SVM Model, binarized data | Sigmoid | 1 | 87.7619 |
| SVM Model, binarized data | Linear | 1 | 89.01587 |
| SVM Model, binarized data | Radial | 100 | 90.98413 |
| SVM Model, binarized data | Radial | 10 | 91.57943 |
| SVM Model, binarized data | Radial | 10000 | 90.98413 |

Figure 12: SVM Results Table

As shown, the radial kernel provided the best prediction accuracy. Adjusting the cost to 10 with this kernel in the tests run provide the best level of accuracy at 91.57843%.

## Random Forest

Random Forest is a decision tree model that builds several trees to improve the accuracy of the model. The key to modeling with Random Forest is finding the optimal number of trees in the “forest” that will provide the best level of accuracy. For this model, the code shown in the following figure was used to generate the random forest.

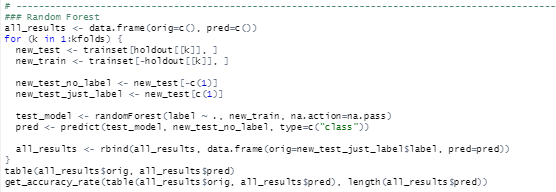
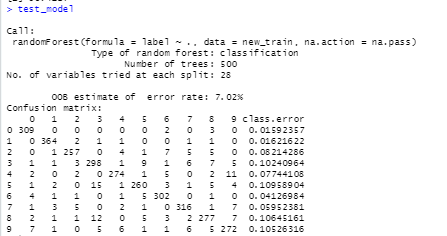
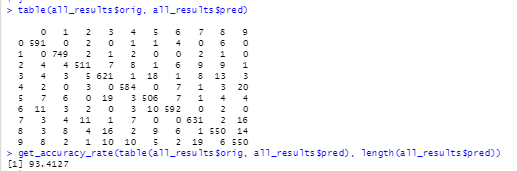
  
   


Figure 13: RandomForest Code with Accuracy Results

As shown above, the number of trees found was 500, but the accuracy was very high at 93.4127. That is a very large number, so a function was written to loop through different numbers of trees to find the best accuracy and then setting the optimal number of trees for random forest. This code is shown in the following figure.

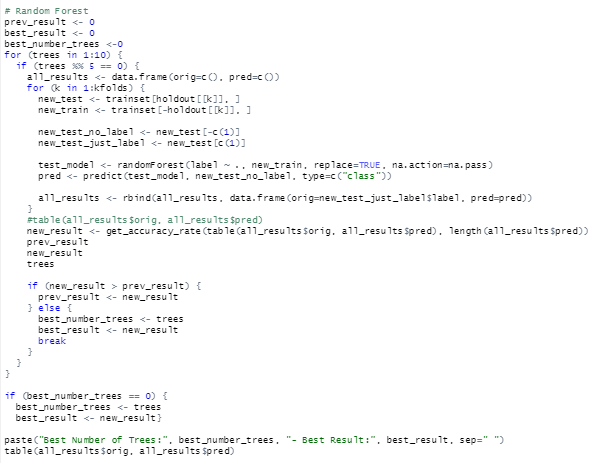
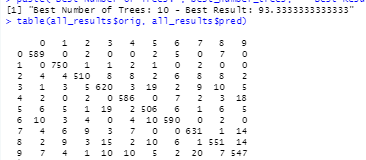
  


Figure 14: RandomForest Best Number of Trees Loop

Using this method, the optimal number of trees is 10 (between 1 and 10) and the optimal result is 93.3333. Random Forest provides a very good overall accuracy for this data set. Notice that 500 trees only provided a slight improvement in accuracy, but a much more complex model.

# Algorithm Performance Comparison

The models provided very different levels of accuracy, but in most cases, all were very high close to 90 percent accuracy all the time. This could be due to the fact that due to the overall size of the data set, in order to run more timely tests, the data was reduced. In addition, for this exercise, the training set was the only set used. With a larger computer and processor, the entire dataset could be used with likely higher accuracy.

# 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.