Brandon Croarkin

# HW6: naïve Bayes and decision tree for handwriting recognition

## Section 1: Introduction

Optical character recognition (OCR) is a topic that has been at the forefront of machine learning in recent years. OCR is the process of converting images of typed, handwritten or printed text into machine-encoded text. This can come from scanned documents, photos of documents, or even images with superimposed text.

Today, OCR has become capable of a high degree of recognition accuracy from most fonts, with a support for multiple languages as well. It is widely used to assist with information entry from printed paper data records. When successful, it can drastically cut down manual data entry time. As the sophistication of these OCR algorithms have improved, its applications have spread. OCR has almost become ubiquitous and can found in tasks such as passport recognition, converting electronic images of printed documents to searchable versions (Google Books), automatic number plat recognition, and even extract business card information into a contact list.

The goal of this report is to correctly identify digits from a dataset of tens of thousands of handwritten images using the MNIST dataset. To correctly classify images there are many steps that need to be done first. One, a large dataset of data is needed. Without a large collection of data, it is hard if not impossible to be able to account for all the variations in how a digit can be written. Next, these images need to be of uniform dimensions. The algorithms used are unable to compare a 28x28 pixel image to a 50x50 pixel image unless it resizes the images within the algorithm. Furthermore, the images’ pixels need to be on the same color scale. Color images are 3-dimensional and contain data for the level of red, green, and blue in a pixel. This is very different than 1-dimensional black and white images that has a single integer value between 0 and 255, inclusive, with higher numbers meaning darker. Again, it is impossible to compare color images to black and white images using these algorithms without pre-processing to put them on the same scale.

### The Data

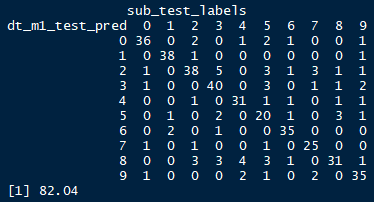
MNIST (“Modified National Institute of Standards and Technology”) is the de facto “hello world” dataset of computer vision. It was originally released in 1999 and has since served as the basis for benchmarking classification algorithms. It consists of a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-set image. These images are 28x28 pixel grayscale.

## Section 2: Decision Tree

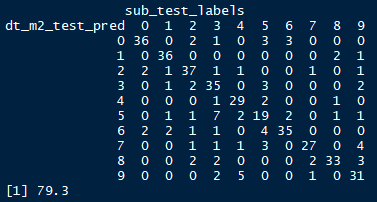
Decision trees are one of the simplest, yet most widely used classification techniques. They are powerful classifiers, which utilize a tree structure to model the relationship among the features and the potential outcomes. These models are built using a heuristic called recursive partitioning, also known as “divide and conquer”, that splits the data into subsets, which are then repeatedly split into even smaller subsets until the data in the subsets are homogenous or another stopping criterion has been met.

Unlike Naïve Bayes, decision trees usually run best if the data is discretized. All the data was scaled from the 255 scale to a 0 to 1 scale. For this analysis, the pixel data was initially discretized by splitting each pixel into “Dark” and “Light”. This was done using different cutoff values and a different number of cutoffs to determine the best combination for results. The results of the different models are below. All were performed with the C5.0 package and a seed set to 11. All testing was done with a subset of the total data (4,000 observations out of the initial 42,000). The accuracy of the models was calculated by splitting this subset into a training and dataset with 3599 and 401 observations respectively.

**Model 1:** default parameters with categorical dataset based on two categories with a pixel cutoff of 0.3

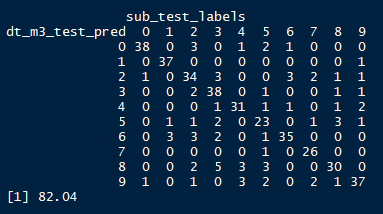


**Model 2:** default parameters with numeric dataset



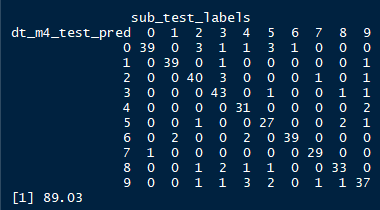
There is a clear drop-off in performance from the categorical and numeric datasets so further testing was done solely on the categorical datasets.

**Model 3:** default parameters on categorical dataset with 3 categories split at 0.3 and 0.6



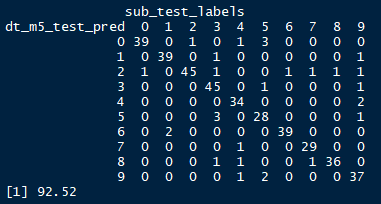
The accuracy is actually the exact same with more categories, so additional increases in performance will likely need to be found elsewhere.

**Model 4:** three trials of boosting were used with the 3 categories for pixel labels.



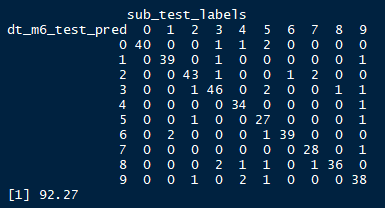
There is a noticeable increase in performance added from introducing boosting into the model.

**Model 5:** ten trials of boosting were used with the 3 categories for pixel labels.



There is another noticeable increase in performance from further boosting runs added.

**Model 6:** ten trials of boosting were used with the 3 categories for pixel labels and pruning.



There is no major difference in the accuracy, in fact there was a minor drop in performance.

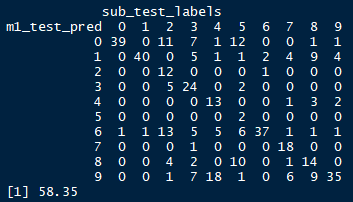
Further improvement to this model could potentially come from altering the cutoff splits in different ways, adding more categories, or adding further boosting. Introducing random forest algorithms could also be a way to improve the tree model further.

## Section 4: Naïve Bayes

A naïve Bayes classifier

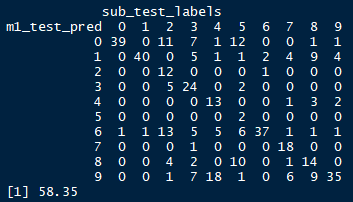
The e1071 package in R was primarily used to run Naïve Bayes in R, but the caret package was also used to test the performance in a different R package.

**Model 1:** default parameters



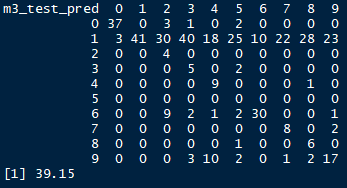
The Naïve Bayes out of the box does not perform very well, especially when compare to the decision tree models.

**Model 2:** Laplace smoothing was added to the default parameters.



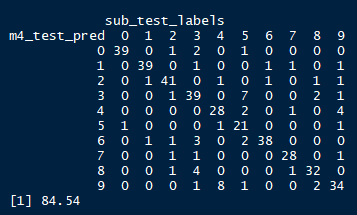
The addition of Laplace smoothing did not appear to make any impact on the model’s performance.

**Model 3:** caret package was used with 3-fold cross validation.



The Naïve Bayes from the caret package performed very poorly out of the box. It appears that this is because it is predicting nearly everything as a 1.

**Model 4:** e1071 naïve Bayes model with discretized pixel values with 3 categories and Laplace smoothing.



This was the best performing naïve Bayes model used. It appears that discretized data helps in the prediction for both decision tree and naïve Bayes.

Further improvement to this model could come from finding what features were the best at classifying and using just those features in the classification task.

## Section 4: Algorithm Performance Comparison

The decision tree model clearly outperformed the naïve Bayes model. The best accuracy in testing for the naïve Bayes was 85%, whereas the decision tree reached 93% accuracy. In addition to the better accuracy, the decision tree model ran much faster than the naïve Bayes models. While the decision tree models ran in a few minutes or less even with up to 10 trials of boosting, the naïve Bayes would take up to 30 minutes to run certain models. This was also all on just the subset data that was approximately 10% of the data.

The speed of the decision trees can likely be attributed to the speed of making the calculations after the tree is made. Once the tree is made, computing a given observation is computationally very fast. Naïve Bayes incurs much more computational cost in creating a prediction, even after the model is made.

Both algorithms had outputs that were not interpretable. Plots of the decision tree models, even those with pruning, still were too large to give any plot to get information from. The summary outputs from the naïve Bayes algorithms were also too long to read through and interpret. In this respect, both models were fairly equal in interpretability, despite decision trees notoriety of being an interpretable model.

Both models also performed best with discrete data. In particular, the naïve Bayes model saw a significant boost in performance when discrete data was used.

## Section 5: Kaggle Test Result

The best performing model in testing was Model 5 using the decision tree with 3 categories, 10 trials of boosting, and no pruning. This was the model that was submitted to Kaggle and had ~95% accuracy. Overfitting did not appear to be a problem for the decision tree model. In fact, it actually had a higher accuracy in the Kaggle submission than it did in all my trials. This suggests that overfitting was not an issue.

