Jeff Levesque

Professor Gates

IST 565

Homework #6

**Introduction**

Handwriting recognition provides various organizations the ability to convert text documents, into a digitized equivalent. This conversion allows documents to be better archived, distributed, and further analyzed. Amazon has taken advantage of handwriting recognition, through the kindle platform. More specifically, using the tool ‘Kindle Convert’, printed books can be converted to an ebook.

Though industry such as Amazon have taken advantage of handwriting analysis, numerous libraries have taken great strides converting many printed documents, into digital format. In 2009, the Library of Congress scanned it’s 25,000th printed document electronically. This movement ensures that the longevity of content, since the printed copy, may be brittle requiring more care.

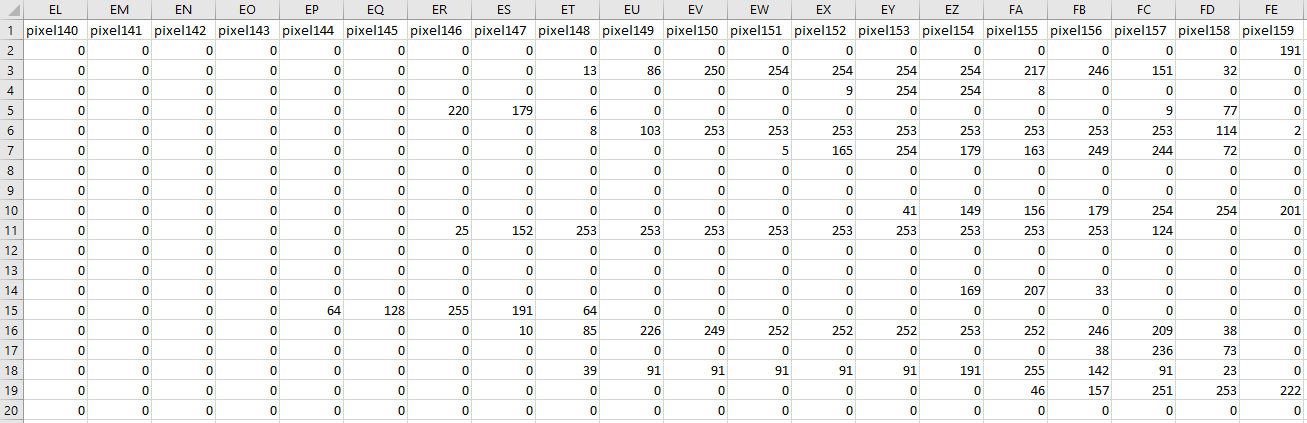
Other uses for handwriting recognition include various electronic devices. For example, many smartphones, as well as tablets have countless applications, allowing users to write directly on the device screen. The corresponding text, then undergoes conversion to the digitized character representation. Tools like these allow students, and professionals to easily take notes freeform.

Though, many different algorithms can be implemented for the classification task, this project will solely focus on decision trees, as well as naïve bayes. More specifically, these two algorithms will be used to classify printed numeric characters. Then, a comparison will be made between the accuracy of the algorithms, as well as the performance to generate a model, and the corresponding prediction.

**Analysis**

Data Preparation:

Two csv datasets representing instances of numeric digits ranging from 0-9 are used. More generally, each numeric digit was represented by a sequence of (roughly 780) pixels. Each pixel contains a numeric value associated with the pixel density. Therefore, higher values, are associated with darker pixels. The train dataset contained 42,000 records, where each row represented a numeric instance. Conversely, each column represented a pixel density. The test dataset was similarly structured, containing 28,000 numeric records.



Since pixel 0 through pixel 11, and pixel 780 through 783 of the train, and test dataset contained a column sum of zero, they were removed from the corresponding dataframe:

## remove redundant pixels

delete = c(

'pixel0',

'pixel1',

'pixel2',

'pixel3',

'pixel4',

'pixel5',

'pixel6',

'pixel7',

'pixel8',

'pixel9',

'pixel10',

'pixel11',

'pixel780',

'pixel781',

'pixel782',

'pixel783'

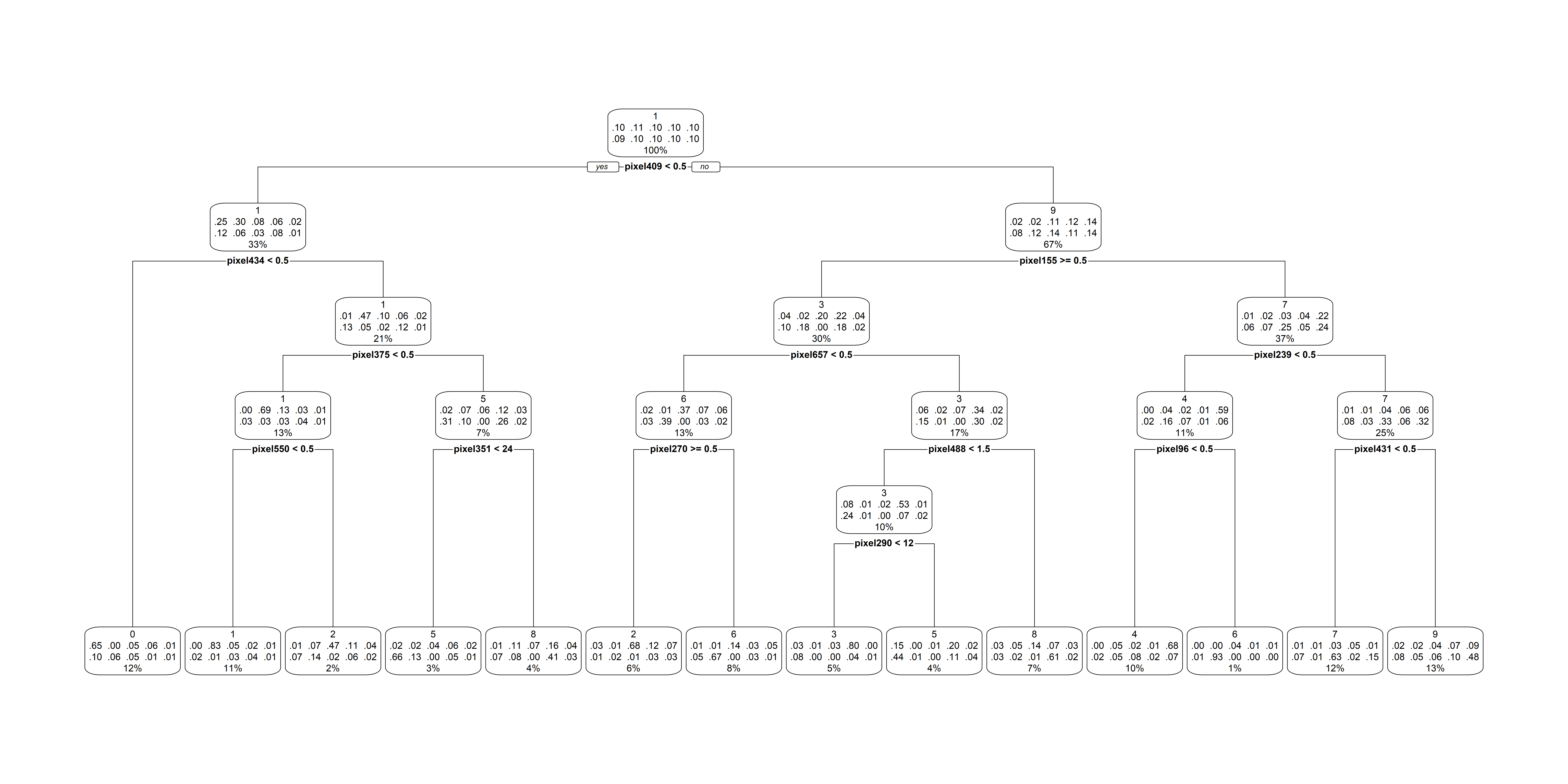
)

df.train = df.train[, !(names(df.train) %in% delete)]

df.test = df.test[, !(names(df.test) %in% delete)]

**Results**

Once the dataframe refactored, rpart and rpart.plot were used to generate decisions trees, along with the corresponding plot. The test dataset was used against the fitted tree, to compute the re-substitution error. This was calculated to be 0.363, which indicates that the fitted decision tree was about 64% accurate. Attempting to prune the tree to 3 levels (root is node 0), yielded a re-subsitution error of 0.542. Therefore, pruning the levels did not yield a better model, and the former was retained:



The corresponding 3-fold CV produced 42,000 rows of result. Due to the length of the result, a provided tree\_analysis.txt can be further reviewed. However, below will provide a preview of the accuracy for the first 5 rows, using 3 folds:

xpred.rpart(fit.tree, xval=3)

0.54613303 0.09147209 0.08077784 0.06883030 0.06451660 0.05535899

1 2 2 2 2 2 2

2 2 2 1 1 1 1

3 2 2 2 2 2 2

4 2 8 1 1 1 1

5 2 2 1 1 1 1

0.04571958 0.04247021 0.02909242 0.01840850 0.01585400 0.01328713

1 2 2 2 2 2 2

2 1 1 1 1 1 1

3 2 2 2 2 2 2

4 1 1 1 1 1 1

5 1 1 1 1 1 1

0.01110262 0.01023625

1 2 2

2 1 1

3 2 2

4 1 1

5 1 1

Next, the naivebayes package was used to create a model, with the laplace = 1 argument in the naive\_bayes function. Laplace is generally used with naïve bayes to help smooth out outlier cases that wasn’t accounted for during training. Upon generating a prediction, the computed re-subsitution error was 0.470, which was significantly higher than the decision tree alternative. Therefore, the naïve bayes model was roughly 53% accurate with digit prediction. The inclusion of laplace in the model generation did not provide significant improvements. Specifically, another train and test session conducted without laplace, yielded the same performance. This is likely because all possible digit cases provided in the test dataset, were accounted for during the train. Therefore, the smoothing effect was not utilized, and not needed.

When comparing performance between the two algorithms, it was found that generating the decision trees was significantly faster than the naïve bayes. This was done by wrapping the fit (and predict) functions with Sys.time(). The difference between the two calls, provided a rough estimate of how long it took to generate a model (or prediction). Additionally, the prediction was much faster for the decisions trees:

Decision Tree:

===========================================================

performance (minutes)

===========================================================

[1] "fitting tree: 2.67286456425985"

[1] "predicting probability: 1.54783892631531"

[1] "predicting class: 1.58008599281311"

Naïve Bayes:

===========================================================

performance (minutes)

===========================================================

[1] "fitting tree: 7.53572297096252"

[1] "predicting probability: 53.0671591758728"

[1] "predicting class: 59.0970938205719"

Overall, the decision tree performed better results, and better runtime performance.

**Conclusions**

Predicting printed text is not a unique problem and has been done countless of times. In the example of the Kaggle “Digit Recognizer”, many participants have elected to implement some variation of neural networks. This is advantageous, since neural networks allow numerous cycles of epoch. Specifically, the traversal of computation from each node/layer completely forward, then backwards is considered an epoch. During each epoch, layers of nodes performing the computation is automatically readjusted, by redistributing weights on the nodes. This repeating cycle generally produces a robust model, with respect to the provided dataset.

This comparison of decision trees against naïve bayes in this study was fair. Specifically, both operated on identical train and test data, as well as preprocessing techniques. However, many participants through the Kaggle challenge, produced significantly higher accuracies. At a quick glance, it is apparent that a significant number of participants used neural networks, including the convoluted neural networks (CNN). The correspond results, produced accuracies of 95% and higher. One may wonder if neural networks generally produce better result? This is difficult to speculate.

One extended possibility is to use methods including bootstrap aggregation (i.e. bagging), or random forests. These implementations, would create multiple decision trees, using random selection with replacement, then average the prediction. Methods such as these would decrease variance and usually preferred over the single decision tree. Therefore, it would not necessarily be interesting to compare bagging techniques against naïve bayes, rather to compare bagging, or random forests against results produced by participants using neural networks.

Additionally, it would be interesting to combine multiple learner techniques, as an ensemble method, and compare to the single neural network results. Overall, the results in this study seem insignificant compared to already produced Kaggle finding. However, this study does provide additional follow up questions that would be interesting to compare.