John Fields
Dr. Ami Gates
IST707 - Homework #7
8/22/19

## Introduction

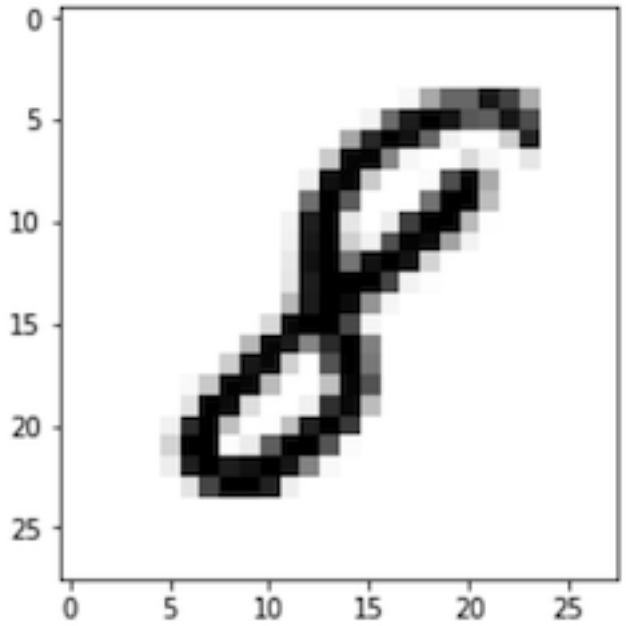



*Figure 1 - Sample of a digit from the MNIST dataset*

The Modified National Institute of Standards and Technology (MNIST) dataset of handwritten digits 0-9 was first released in 1999 and has become a classic challenge for using machine learning on images. The MNIST dataset from Kaggle.com includes 42,000 examples that can be used to train machine learning models and 28,000 examples that can be used for testing. The goal for this analysis is to build several models to correctly identify each of the handwritten images in the test dataset.

There are many practical applications of this technology and handwritten legal documents have many use cases where this technology can be applied. There is much hype in the news media around how artificial intelligence will replace various professions like attorneys and other legal positions. However, as described in a 2018 Forbes article, this technology is more likely to augment the work of attorneys and law clerks by helping them to find documents faster with less manual work.[i] There are many other similar uses for optical character recognition and this paper will explore some of the techniques that can be used to read the handwritten digits 0-9.

## Analysis and Models

About the Data

This dataset includes 70,000 images of the digits 0-9 that were scanned and then added to two Excel spreadsheets (train.csv and test.csv). Here is a description of the data in these files from Kaggle.com:

> "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."[ii]

The training data includes 42,000 examples and the first step to setup the data was to divide the information so that 70% can be used for training the models and 30% can be used for a trial test. The final test will be done on the 28,000 unlabeled records which will be submitted to Kaggle.com for scoring.

After some initial tests on unnormalized data during Homework #6, an additional transformation to normalize the data from values of 0-255 to 0-1 was implemented. Normalization was done to reduce the potential skew in the results. After reading in the data and normalizing, the next step is to look at the data in the 70% training and 30% trial datasets to see if we have an even distribution of values. Figures 3 and 4 below show an even distribution of digits in both datasets so no additional normalization is required.

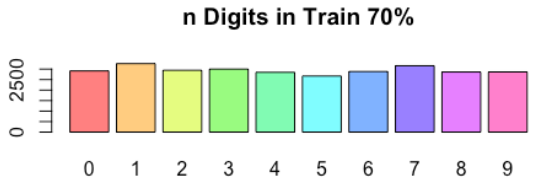



*Figure 2 - Count of digits in the training dataset*

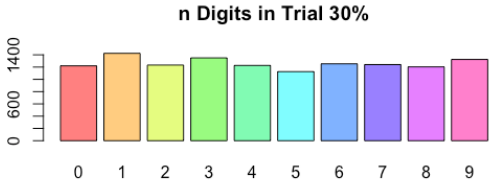



*Figure 3 - Count of digits in the trial dataset*

Additional transformations were performed on the train and trial data to cleanse the data and normalize prior to starting the analysis.

1. The label was changed to a factor and identified as Y.
2. The column headings were changed to X.1 to X.784.
3. Validation of the "levels" from 0-9 and the numeric class for all values except the label.

Notes related to the transformation process above:

1. Steps 1 and 3 were only performed on the test data set since it did not include a label.
2. The label value ("Y") was retained in both the train and trial datasets. Prior to running the algorithms on the trial dataset, the label will be removed.

After the transformations are complete, the distribution of the data and labels was verified in Figures 2 and 3 below.

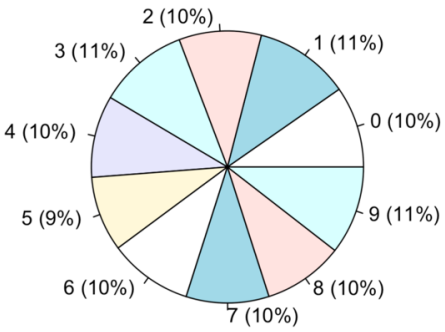



*Figure 4 - Distribution of digits in trial 30%*

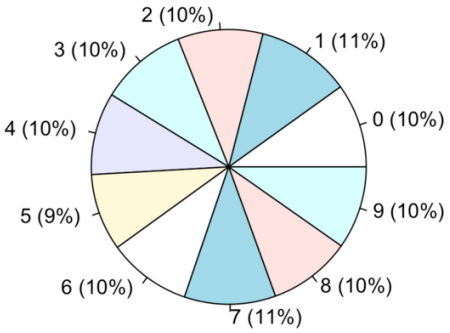



*Figure 5 - Distribution of digits in training 70%*

The steps required for the exploratory data analysis, cleansing and processing are now complete and the data is ready for modeling.

Model 1 - k-Nearest Neighbor (k-NN)

"Nearest neighbor classification is part of a more general technique known as instance-based learning, which does not build a global model, but rather uses the training examples to make predictions for a test instance."[iii] As the name of the algorithm suggests, mathematical calculations are performed to determine the distances between data points when making predictions on test data.

Applying k-NN in analysis software packages like R and Python is typically accomplished with a few lines of code. However, since the prediction process in kNN takes place in one step, the algorithm runs longer so this is something that must be considered when deploying this model in production with a large dataset.

The k-NN algorithm was trained with the Training 70% data and then run on the Trial 30% data with the resulting confusion matrix:

| | Predicted Class | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Actual Class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1191 | 1 | 0 | 0 | 1 | 7 | 15 | 0 | 1 | 4 |
| 1 | 0 | 1418 | 1 | 1 | 0 | 0 | 3 | 1 | 0 | 0 |
| 2 | 23 | 91 | 1016 | 10 | 15 | 3 | 4 | 49 | 17 | 5 |
| 3 | 2 | 31 | 9 | 1226 | 1 | 20 | 8 | 17 | 18 | 18 |
| 4 | 0 | 34 | 0 | 0 | 1101 | 1 | 6 | 8 | 1 | 76 |
| 5 | 9 | 31 | 0 | 38 | 9 | 990 | 21 | 1 | 3 | 24 |
| 6 | 23 | 27 | 0 | 0 | 6 | 7 | 1188 | 2 | 0 | 0 |
| 7 | 2 | 68 | 1 | 0 | 3 | 0 | 0 | 1133 | 0 | 34 |
| 8 | 10 | 74 | 2 | 60 | 9 | 22 | 6 | 7 | 972 | 41 |
| 9 | 12 | 25 | 1 | 18 | 14 | 0 | 2 | 39 | 3 | 1210 |

*Figure 6 - k-NN confusion matrix for trial data*

Model 2 - Support Vector Machine

> *"A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side."*[iv]

The SVM analysis will utilize three different "kernels" (Polynomial, Linear, Radial) to test which performs best with the digits dataset. For each kernel, the confusion matrix table will be compared, and the results will be discussed later in the Results section.

SVM - Polynomial:

```
Call:
svm(formula = DATASET.train.7$Y ~ ., data = DATASET.train.7, kernel = "polynomial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  3
      gamma:  0.00127551
     coef.0:  0

Number of Support Vectors:  28665
```

*Figure 7 - SVM polynomial statistics for training*

| | Predicted Class | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Actual Class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 675 | 497 | 2 | 6 | 0 | 0 | 18 | 19 | 3 | 0 |
| 1 | 0 | 1424 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 3 | 781 | 420 | 8 | 0 | 0 | 3 | 8 | 9 | 1 |
| 3 | 0 | 820 | 8 | 505 | 0 | 0 | 1 | 7 | 9 | 0 |
| 4 | 0 | 1071 | 0 | 0 | 70 | 0 | 1 | 76 | 0 | 9 |
| 5 | 3 | 922 | 1 | 144 | 1 | 22 | 10 | 7 | 12 | 4 |
| 6 | 1 | 778 | 3 | 0 | 0 | 0 | 468 | 1 | 2 | 0 |
| 7 | 1 | 914 | 0 | 0 | 0 | 0 | 0 | 325 | 1 | 0 |
| 8 | 2 | 761 | 4 | 82 | 0 | 0 | 1 | 20 | 329 | 4 |
| 9 | 4 | 1008 | 1 | 4 | 1 | 0 | 1 | 272 | 1 | 32 |

*Figure 8 - SVM polynomial confusion matrix for trial data*

The output of the SVM polynomial was surprising and the initial cause was expected to be a data error or program flaw. After running the linear and radial kernels on the same data, the conclusion is that the polynomial kernel does not work well on this dataset. This will be explored further in the results section.

SVM - Linear:

```
Call:
svm(formula = DATASET.train.7$Y ~ ., data = DATASET.train.7, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.00127551

Number of Support Vectors:  5702
```

*Figure 9 - SVM linear statistics for training*

| | Predicted Class | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Actual Class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1182 | 0 | 8 | 3 | 2 | 9 | 7 | 1 | 8 | 0 |
| 1 | 0 | 1403 | 2 | 6 | 1 | 1 | 2 | 3 | 6 | 0 |
| 2 | 8 | 9 | 1117 | 18 | 18 | 4 | 27 | 15 | 14 | 3 |
| 3 | 1 | 8 | 20 | 1235 | 3 | 38 | 4 | 10 | 26 | 5 |
| 4 | 0 | 3 | 12 | 1 | 1154 | 0 | 13 | 8 | 3 | 33 |
| 5 | 10 | 3 | 13 | 60 | 9 | 977 | 20 | 5 | 22 | 7 |
| 6 | 14 | 1 | 7 | 2 | 10 | 3 | 1209 | 2 | 5 | 0 |
| 7 | 2 | 4 | 11 | 1 | 5 | 3 | 2 | 1186 | 3 | 24 |
| 8 | 2 | 20 | 14 | 49 | 4 | 18 | 13 | 8 | 1065 | 10 |
| 9 | 5 | 7 | 6 | 15 | 47 | 8 | 1 | 47 | 7 | 1181 |

*Figure 10 - SVM linear confusion matrix with trial data*

SVM - Radial:

```
Call:
svm(formula = DATASET.train.7$Y ~ ., data = DATASET.train.7, kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.00127551

Number of Support Vectors:  11578
```

*Figure 11 - SVM radial statistics for training*

| | Predicted Class | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Actual Class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 1193 | 0 | 4 | 1 | 3 | 2 | 8 | 0 | 8 | 1 |
| 1 | 0 | 1401 | 3 | 6 | 1 | 4 | 4 | 2 | 3 | 0 |
| 2 | 10 | 6 | 1117 | 11 | 22 | 5 | 25 | 19 | 15 | 3 |
| 3 | 1 | 10 | 23 | 1209 | 2 | 50 | 8 | 15 | 25 | 7 |
| 4 | 0 | 4 | 7 | 1 | 1166 | 0 | 8 | 3 | 2 | 36 |
| 5 | 8 | 10 | 1 | 42 | 11 | 1026 | 17 | 2 | 8 | 1 |
| 6 | 13 | 6 | 7 | 0 | 10 | 12 | 1200 | 0 | 5 | 0 |
| 7 | 3 | 10 | 13 | 2 | 15 | 2 | 0 | 1163 | 4 | 29 |
| 8 | 3 | 20 | 9 | 26 | 5 | 32 | 12 | 7 | 1076 | 13 |
| 9 | 10 | 7 | 5 | 20 | 41 | 8 | 1 | 38 | 11 | 1183 |

*Figure 12 - SVM radial confusion matrix for trial data*

Model 3 - Random Forest:

> "As the name suggests, the algorithm constructs a large number of different trees (as definied by the user) by randomly selecting the features that can be used to build each tree (as opposed to using all the features for each tree). Typically, the trees in a random forest also have the parameters set to ensure each tree will also be relatively shallow, meaning that the algorithm creates a large number of shallow decision trees (decision bonsai?). Once the trees are constructed, each tree is used to predict the outcome for a new record, with these multiple predictions then serving as votes, with a majority rules approach applied."[v]

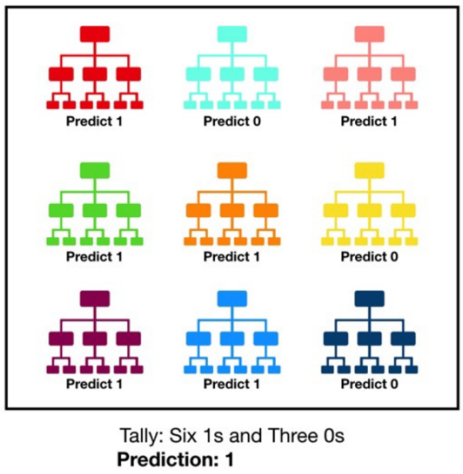
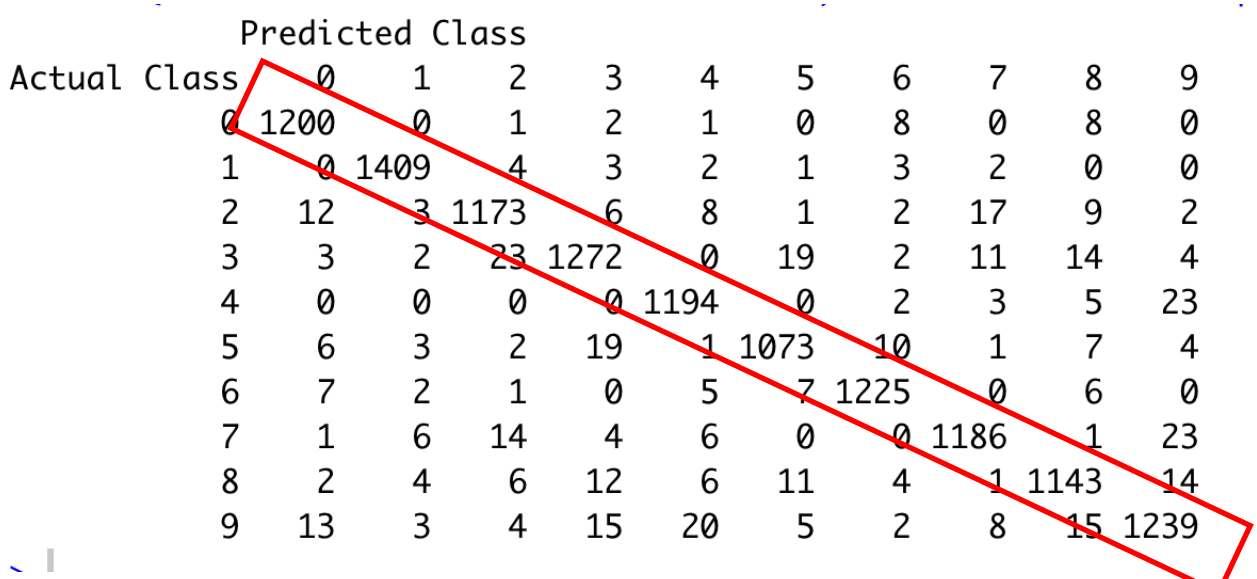



*Figure 13 - Visualization of a random forest model*

The random forest model was run using the classication type and 100 "trees". The training model evaluated 28 variables at each split and generated the predictions from the trial dataset shown in Figure 15.

```
Call:
 randomForest(formula = DATASET.train.7$Y ~ ., data = DATASET.train.7,      ntree = 100)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 28
```

*Figure 14 - Random forest statistics for training*



| Actual Class \ Predicted Class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1200 | 0 | 1 | 2 | 1 | 0 | 8 | 0 | 8 | 0 |
| 1 | 0 | 1409 | 4 | 3 | 2 | 1 | 3 | 2 | 0 | 0 |
| 2 | 12 | 3 | 1173 | 6 | 8 | 1 | 2 | 17 | 9 | 2 |
| 3 | 3 | 2 | 23 | 1272 | 0 | 19 | 2 | 11 | 14 | 4 |
| 4 | 0 | 0 | 0 | 0 | 1194 | 0 | 2 | 3 | 5 | 23 |
| 5 | 6 | 3 | 2 | 19 | 1 | 1073 | 10 | 1 | 7 | 4 |
| 6 | 7 | 2 | 1 | 0 | 5 | 7 | 1225 | 0 | 6 | 0 |
| 7 | 1 | 6 | 14 | 4 | 6 | 0 | 0 | 1186 | 1 | 23 |
| 8 | 2 | 4 | 6 | 12 | 6 | 11 | 4 | 1 | 1143 | 14 |
| 9 | 13 | 3 | 4 | 15 | 20 | 5 | 2 | 8 | 15 | 1239 |

*Figure 15 - Random forest confusion matrix for trial data*

Now that the models have been tested and the accuracy levels calculated, the results of the predictions will be discussed in the next section.

## Results

Model 1 - k-Nearest Neighbor (k-NN)

The prediction with k-NN on the trial data was 90.8% accurate. For the test data from Kaggle with 28,000 examples, the k-NN performed even better with trial data at 91.4%.

| |
|---|
| k-NN Trial 30% Data Accuracy: 90.8% |
| Kaggle Test Data: Accuracy: 91.4% |

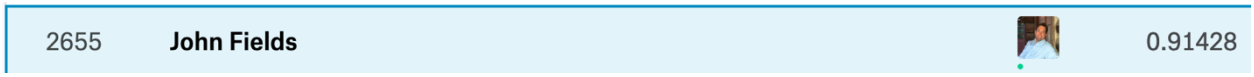



*Figure 16 - Kaggle leaderboard score for k-NN*

Model 2 - Support Vector Machine

The SVM model was run with the Polynomial, Linear and Radial kernels.

| Kernel | Trial Data Accuracy (%) | Time for Training (minutes) |
|---|---|---|
| Polynomial | 33.9 | 69.8 |
| Linear | 92.9 | 5.6 |
| Radial | 93.1 | 11.0 |

*Table 1 - Comparison of SVM kernel accuracy and model build times*

The polynomial kernel struggled to find a hyperplane that separated the dataset correctly and predicted the digit "1" for most of the trial examples. The linear and radial kernels were both faster and provided much better accuracy. Since the radial had the best results in trials, this was also submitted to Kaggle to test the model on the 28,000 examples in the test dataset and returned a 93% accuracy.

SVM Radial on Trial 30% Data Accuracy: 93.1%

Kaggle Test Data: Accuracy: 93.2%


*Figure 17 - Kaggle test results for SVM radial kernel*

Model 3 - Random Forest:

The Random Forest was the last model tested and it generated the best results for accuracy and generalized well on the Kaggle test dataset.

Trial 30% Data Accuracy: 96.1%

Kaggle Test Data: Accuracy: 96.3%


*Figure 18 - Kaggle test results for Random Forest*

A comparison of the time for training the model also generated some interesting results. The SVM Polynomial was the outlier with a disproportionately long time (and the poorest results as described above). The kNN also takes more time since it combines the training/prediction in one step. The Random Forest and SVM Linear were the lowest with 4.6 and 5.6 minutes respectively.

| **Model** | **Time for Training (minutes)** |
|---|---|
| kNN | 28.7* |
| SVM | |
| - Polynomial | 69.8 |
| - Linear | 5.6 |
| - Radial | 11.0 |
| Random Forest | 4.6 |

*Figure 19 - Model completion time comparison*

** training/prediction in one step for kNN*

## Conclusion

The use of machine learning techniques has proved to be beneficial for solving many challenging problems like the automatic reading of handwritten digits. As the technology and knowledge continues to increase in this area, new applications and methods will continue to be developed.

Below is a summary of the observations from this analysis which can be utilized to understand the capabilities and limits of the techniques analyzed:

1. The models predicted the digits 0-9 from test data with accuracies between 91-96%.
2. When applying the models to test data, the models did an excellent job of "generalizing" on unseen data with higher test accuracies compared to the trial accuracies.
3. The Support Vector Machine results are a great example of how using algorithms as a "black box" can give poor results without testing a variety of alternative models. The first SVM model tested provided had a 34% accuracy and the last model 93%.

Considering all of the factors reviewed in this paper, the recommendation for the digits dataset would be to use the Random Forest model. This conclusion is based on the higher accuracy across training/testing datasets for Decision Trees and faster processing time.

Continued advancements in machine learning and other data science techniques have helped to automate processes like image recognition and there are an unlimited number of applications where these techniques and technology can be applied. However, prior to putting these models into a production environment, careful analysis needs to occur to ensure that the most appropriate model is applied for the type of data being tested and the best desired outcome is achieved.

[i] Marr, Bernard. "How AI And Machine Learning Are Transforming Law Firms And The Legal Sector." *Forbes*, Forbes Magazine, 23 May 2018, www.forbes.com/sites/bernardmarr/2018/05/23/how-ai-and-machine-learning-are-transforming-law-firms-and-the-legal-sector/#2b8f183c32c3.

[ii] Kaggle.com. "Digit Recognizer." *Kaggle*, www.kaggle.com/c/digit-recognizer/data.

[iii] Tan, Pang-Ning. *Introduction to Data Mining*. Pearson Education, 2019

[iv] Patel, Savan. "Chapter 2 : SVM (Support Vector Machine) - Theory." *Medium*, Machine Learning 101, 4 May 2017, medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.

[v] Romero, Brett. "Data Science: A Kaggle Walkthrough – Creating a Model." *Brett Romero - Data Inspired Insights*, 2016, brettromero.com/data-science-kaggle-walkthrough-creating-model/.