**To Catch a Killer**

**Ben Harwood**

On October 12, 2017, streaming service Netflix released a new series title “Mindhunter” that told the (fictionalized) story of the establishment and development of the FBI’s behavioral science division in the late 1970’s and early 1980’s (indeed explaining the origins of the term serial killer). Without getting too in depth, the show revolves around two FBI agents travelling the country interviewing people convicted of exceptionally heinous, vile acts. Among these were Ed Kemper and Jerry Brudos in the first season, and David Berkowitz and Charles Manson in season 2. In addition, each episode had a short (say 5 minute) section showing the antics of an unnamed man who is heavily implied to be Dennis Rader, the BTK Killer of Wichita, Kansas.

Rader killed 10 people between 1974 and 1991. He gave himself the name “BTK” (short for “bind, torture, kill”) when he began sending letters to the police bragging about his crimes. While the case was solved through information that police found hidden on a floppy disk (apparently he was smart enough to evade capture for 30 years but not smart enough to realize sending the police a floppy disk with his latest taunting letter could pose a problem, even after he asked the police and they told him it would be safe), the police were also able to match the handwritten letters he had sent them previously to him.[[1]](#footnote-1)

There are numerous ways that handwriting analysis can be done. In a previous article, the present author discussed two techniques, and the current article is an expansion of that discussion to examine three more.

**Section 1: Analysis and Models**

In this section, the data to be used for the examination will be explored, and then three different models will be developed: *k*-nearest neighbor; support vector machines; random forests.

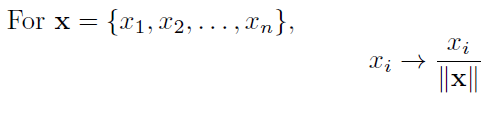
**Section 1.1: Data**

As with the previous discussion, two data sets are used. The first is a collection of 1400 images. However, as was mentioned in the introduction, each image is broken down into a code. But how does a computer encode an image? Pixel by pixel.

Each image is made up of a certain number of pixels. Generally speaking, a *pixel* is the smallest single component of any digital image. The more pixels present in an image the better the detail, clarity, color, etc. of the said image. In computer code, an image is turned into a vector of values that are, for each pixel, the color code. Without getting into an unnecessarily technical discussion about color codes, the primary of concern for the purpose of this study is the *value*, the intensity of the pixel, or the relative lightness/darkness. Values range from 0 for not present to 255 for the highest intensity. Each image was a handwritten digit from 0 to 9 and was comprised of 784 pixels. In the dataset each image is a vector 785 units long, with the corresponding values for each pixel intensity as well as a label indicating what number the image was.

Digit counts


There were no missing values present. From a data cleaning perspective, the labels were converted to category labels. Additionally, because the algorithms were better with normalized data, each pixel vector was transformed according to the standard vector normalization formula:



This induced some division by zero issues, so every case this happened was assigned 0.

The second dataset consisted of information for an additional 1000 images, encoded the same way, however there were no labels. The same transformation was applied to the pixel vectors. Finally, the larger data set was randomly split into two smaller sets, one comprising 80% of the total to be used for training the various models and the remaining 20% kept in reserve to be used for testing.

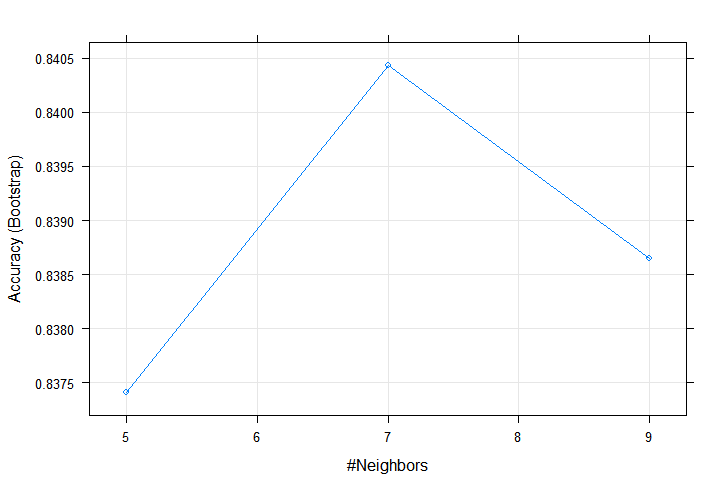
**Section 1.2: Models**

As mentioned at the beginning of Section 1, three models will be explored here, beginning with *k*-nearest neighbor.

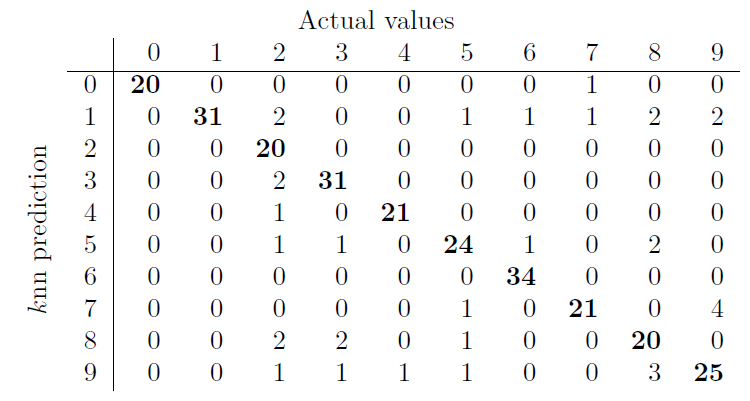
**Section 1.2.1: *k*-nearest neighbor**

The idea of the *k*-nearest neighbor algorithm is in a sense similar to *k*-means clustering in that it is assumed that the data points are grouped together with those most similar. Think of the saying “birds of a feather flock together”. The algorithm is a supervised learning algorithm[[2]](#footnote-2) that uses a distance metric to find the *k* closest neighbors to each data point. Basically, it seeks to capture the “similarity” between various points, so that when new data is introduced it can see which group of points the new data is most similar to an assign it a label. Where this is different from *k*-means clustering is that with *k*-means one must specific the number of clusters, while *k*nn requires the number of neighbors desired.

The first step is to apply the *k*-nearest neighbor algorithm to the training set. The algorithm runs through several different values of *k* to find the optimal number of neighbors, and as seen below that optimal number in this case was 7.



The fitting model determined by the algorithm is then applied to the testing set, yielding (in this case) an accuracy rate of 88.2%.



The table above shows the number of predicted digits vs the actual digits, where the emboldened values show the matches or correct predictions. Notice that each 0 and 1 was accurately predicted, and that every predicted 2 and 6 was correct.

|  |  |
| --- | --- |
| Figure 1: knn predicted test-digit counts | Figure 2: Actual test-digit counts |

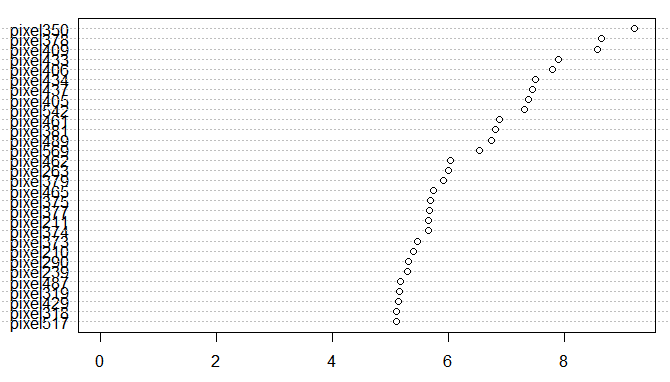
**Section 1.2.2: Random Forests**

As was mentioned in the introduction, decision trees were previously used to study the very same data being studied here. The easiest way to think about a random forest is to imagine a collection of decision trees (hence the name forest), each with their own outcome and then the outcomes being voted on. Such as the Ent moot from The Lord of the Rings: The Two Towers.

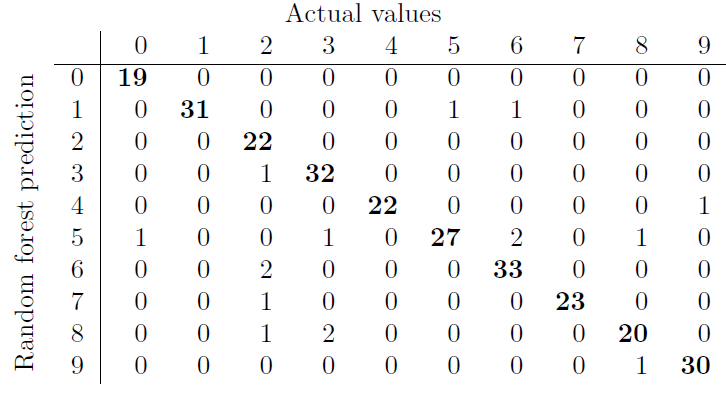


Fantasy novel parlances aside, a random forest is a supervised learning algorithm just like *k*-nearest neighbor. It trains a model based on a set of data that can then be used to make predictions on another set. Additionally, it is able to report which variables are “most important” to the ultimate prediction.

With all this in mind[[3]](#footnote-3) a random forest was built on the same training data as in section 1.2.1. In total, 500 trees were grown, with 28 variables tried at each split. As seen below, pixel 350 seemed to be most important.



The random forest was 94.3% accurate in its predictions as seen here:



|  |  |
| --- | --- |
| Figure 3: Random forest predicted test-digit counts | Figure 4: Actual test-digit counts |

Notice that the random forest predictions for 0 and 2 were perfect, and it accurately predicted every 1, 4 and 7.

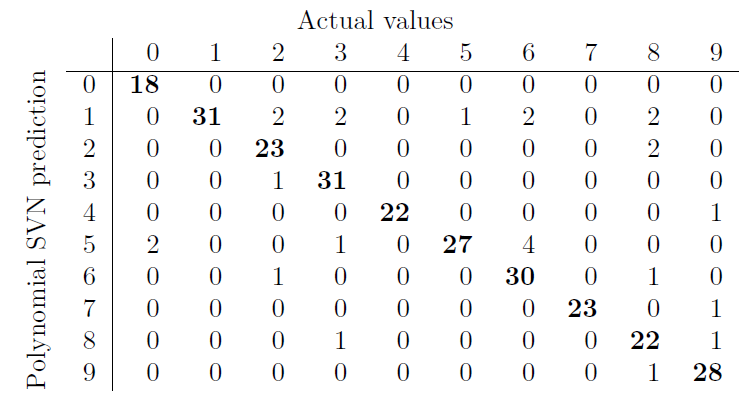
Time to see if the third technique is as effective as the other two.

**Section 1.2.3: Support Vector Machine**

As much as it sounds like some kind of self-help group, a support vector machine is a very useful technique for splitting data. It is very mathematical in nature, relying heavily on the wonder that is linear algebra. Without getting into the mathematics (which could comprise its own 1000 page book), the basic idea is it considers each data point as it would exist in physical space (whatever 785 dimensional space looks like, in the present example) and then finding a way to separate the data into two groups with as wide a gap as possible between the two groups. In the case of two-dimensional data, one might relate the separation as such:



It is not uncommon that a nonlinear model is needed, so certain kernel functions are used. In the case of the present data, roughly equivalent results were generated using a linear kernel function and a polynomial kernel function. Indeed, each of the two models were 91.1% accurate in their predictions of the testing set. The polynomial results follow, the linear results are nearly identical.

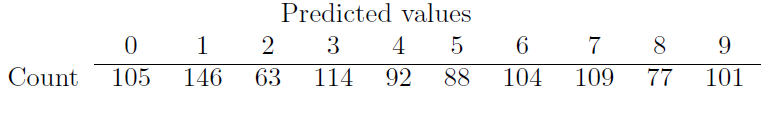


|  |  |
| --- | --- |
| Figure 5: Poly SVM digit-test predictions | Figure 6: Actual test-digits |

**Section 2: Results**

The previous section touched on three methods of classifying the handwritten digits. This section will examine the final predictions made for each and relate those results to the results of the previous analysis using Naïve Bayes and Decision Trees.

Recall that the *k*-nearest neighbor model was highly accurate (88.2%). Here are its predictions:



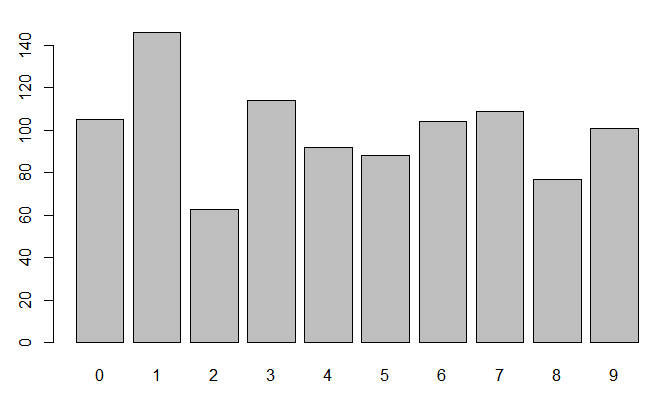
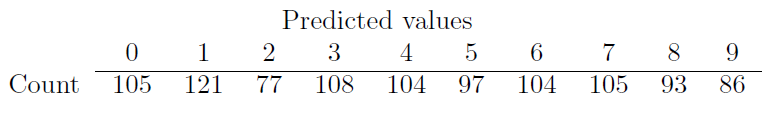


Figure 7:k-nearest neighbor digit predictions

Similarly, the random forest model was highly accurate (94.3%). Predictions follow:



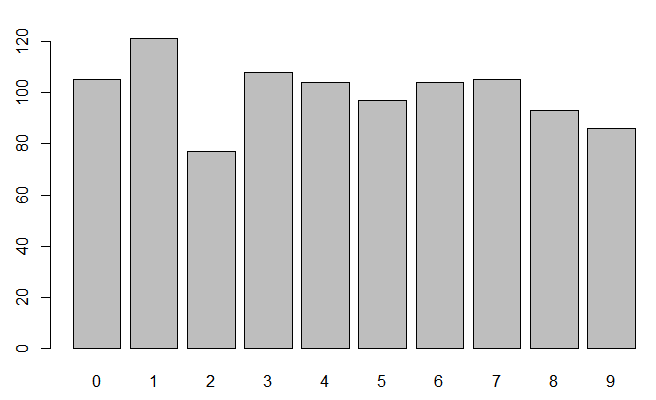
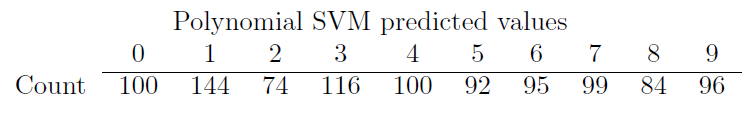


Figure 8: Random forest digit predictions

Rounding out the collection from section 2, the (polynomial) support vector machine’s (91.1%) predictions:



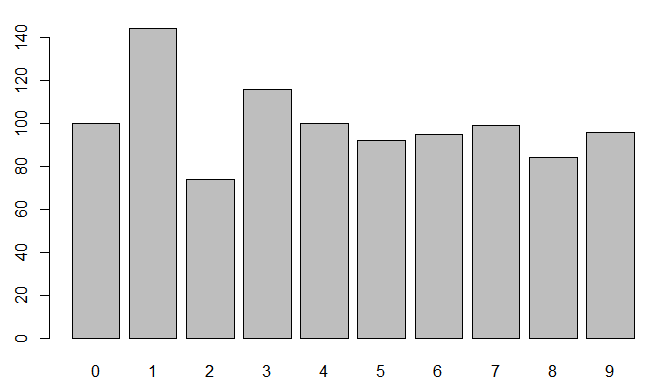
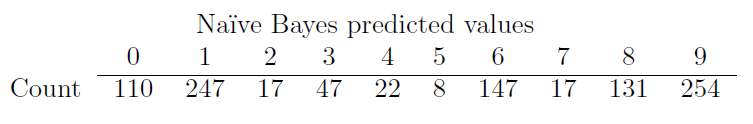


Figure 9: Polynomial SVM digit predictions

The three models discussed in Section 1 agreed in their predictions on 845 of the 1000 images in the unlabeled set.

Naïve Bayes, when applied to the same training and testing data sets yielded a model that was only 51.8% accurate, with the following predictions:



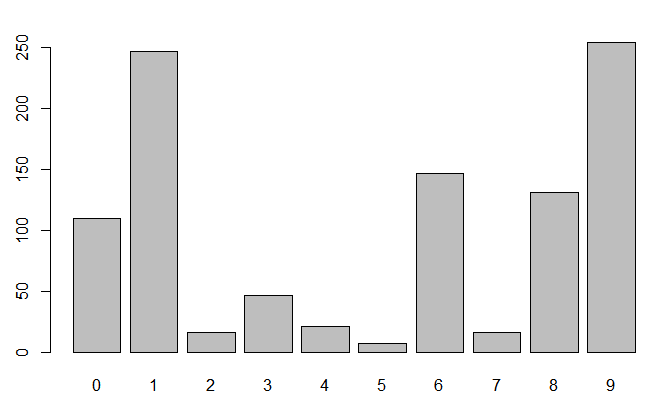
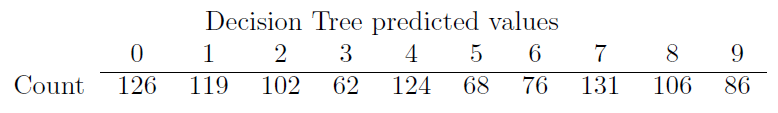


Figure 10: Naive Bayes digit predictions

Finally, a single Decision Tree based on the same training and testing sets resulted in a model that was 64.28% accurate:



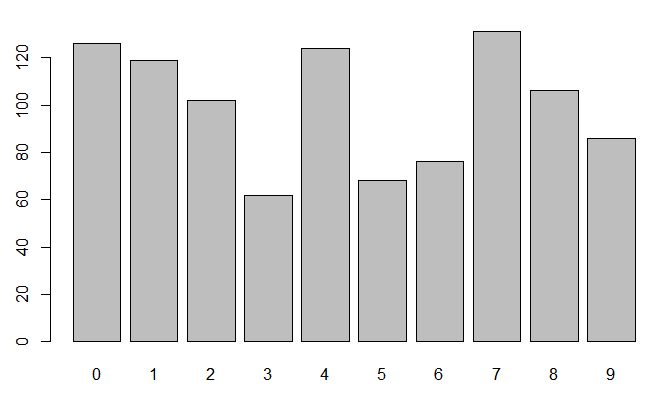
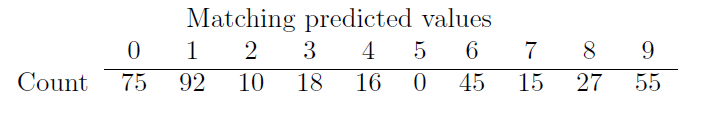


Figure 11: Decision Tree digit predictions

In total, there were 353 data points where all five of the techniques agreed, with the following spread:



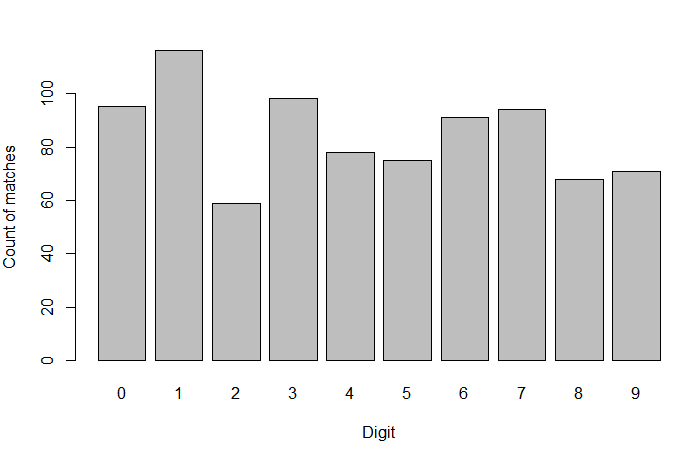


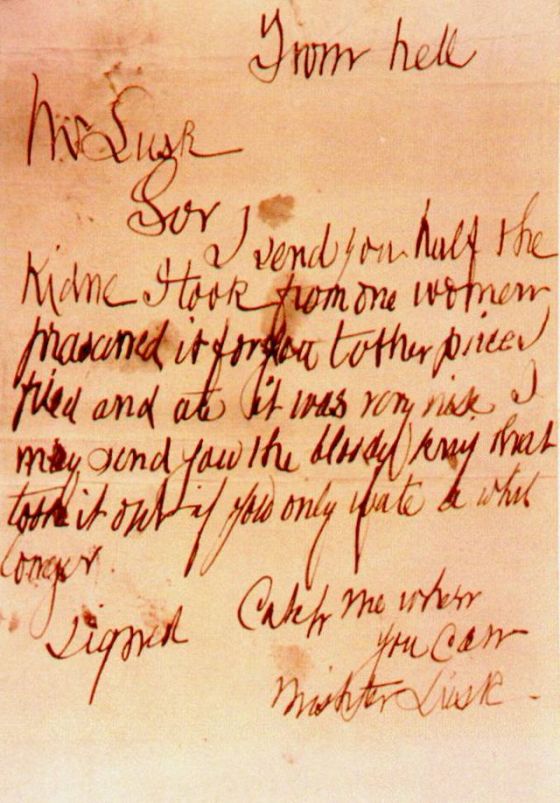
Figure : Matching predicted digits

**Section 3: Conclusion**

As mentioned in the introduction, Dennis Rader was not captured because of the police department used techniques as were described here. The present author has been unable to verify if these techniques were even explored. But there is another serial killer, perhaps the most famous of them all, that this sort of analysis may prove beneficial in solving…

Jack the Ripper.

The case of Jack the Ripper (the infamous serial killer who brutally murdered 5 prostitutes in London’s East End in 1888) has never been officially solved. There is rampant speculation that a highly respected doctor of the day was the villain, but nothing conclusive. Here is an example of the Ripper’s handwriting:



There are numerous places where the ink is more intense than others, and in the same vein as what has been explored here, it seems feasible that an analysis of this writing and others could be done and compared to writings of the suspected physician or any of the other suspects (provided, of course, that examples of these men’s writing could be obtained).

Or, it could be that the case of Jack the Ripper is meant to go unsolved.

1. Note: the handwritten notes were matched to him forensically, however the proceeding discussion does still apply [↑](#footnote-ref-1)
2. Supervised learning algorithms rely on labeled input data with which to learn a function that produces appropriate outcomes when provided new data that is unlabeled. Compare this to unsupervised learning algorithms that learn the structure of un-labeled data. [↑](#footnote-ref-2)
3. And with Treebeard’s permission [↑](#footnote-ref-3)