Predicting MNIST Digits using K-Nearest Neighbors Algorithm

Michael Robertson

October 2019

1 Introduction

1.1 Data

We use the MNIST (Modified National Institute of Standards and Technology) database of handwritten digits downloaded through Scikit-Learn from OpenML[1]. The dataset contains 70,000 28×28 pixel images of centered digits.

1.2 Model

To classified unlabeled samples, we implemented the *K-nearest neighbors* algorithm. To classify a given example based off a feature set, this model examines the distance between the feature vector and every other example in the train set, and collects those k "neighbors" who have the least distance. Because our 784 pixel-features were numbers, we treated our images as vectors, and use the Euclidean distance as our metric between our training examples and the example we wish to classify. Having collected these k nearest neighbors, we have several ways to classify the given example. Most simply, we could take a tally of the categorizations of each of the neighbors, and classify the example as the category with the greatest tally, and tie-break randomly. However, this approach gives no consideration to distances within the neighbor set. For intermediate complexity, we could use the distances from the example to each neighbor as a tie-breaking mechanism, favoring the classificaiton of the neighbor with the shorter distance. Lastly, we could use the distances from the example to the neighbor as weights in a linear combination of the classification vector (which in our case in binary, because a number is only ever appropriately classified as one number), and examine the magnitude of that vector when projected along each of the classification axes.

The K-nearest neighbors model uses "Lazy learning," or instance-based learning, because it requires no training period prior to prediction. Put another way, no computation can be front loaded before the classifcation attempt,

[1]Source: `https://www.openml.org/d/554`

KNN runs in $O(nmo)$ time, where m, n and o are the number of training examples, the number of unclassified examples we attempt to predict, and the size of the data, respectively. For our case, we have a training set of size 6650, a test set of size 350, and data of size $28^2 = 784$. We thus have a running time on the order of $(6650)(350)(784) \approx 10^8$ for our own naive implementation. However, for a single example, we can predict it in $O(nm)$ time, and we believe advance techniques allow the professional implementation of this algorithm by SciKit-Learn to improve the time complexity.

2 Results

We use the F_1 score to measure our model's success. The F_1 combines precision and recall via the harmonic mean to give a reliable overall measure of our model's performance.

F1 score for models for different k values, using Scikit-Learn										
Class	0	1	2	2	4	5	6	7	8	9
k = 3	0.99	0.96	0.99	0.96	0.98	0.97	0.99	0.99	0.98	0.95
k = 4	0.98	0.97	0.97	0.96	0.97	0.96	0.98	0.96	0.96	0.95
k = 5	0.98	0.98	0.97	0.96	0.97	0.96	0.98	0.96	0.96	0.95

We compare these results to those that we naively implemented

F1 score for Models for different k values, using our KNN implementation										
Class	0	1	2	2	4	5	6	7	8	9
k = 3	0.98	0.98	0.98	0.96	0.97	0.96	0.98	0.97	0.96	0.95
k = 4	0.87	0.81	0.78	0.70	0.75	0.72	0.84	0.87	0.78	0.72
k = 5	0.85	0.78	0.72	0.69	0.77	0.70	0.85	0.83	0.75	0.69

Because we would clearly use the professional model, we only demonstrate the functionality of our model. As such, we only used 500 examples in our training set to predict 500 examples in our validate set. Our implementation of KNN is clearly outperformed by the professional implementation, but our rates are certainly higher than those that could come from guessing alone.

Based on the above results and tuning, we recommend using the SciKit-Learn implementation of KNN, with $k = 3$.

3 Conclusion

Our results show that we can confidently classify handwritten digits in our test set with near perfect accuracy. Our results rival but do not surpass those reported by Weinberger, who report a "test error rate of 1.3" (we assume their "test error rate is similar to our F1 metric, but we're unable to find its definition

in their work) on the MNIST dataset [1]. In their 2009 work, they substitute a Mahanalobis distance metric where we used a Euclidean distance to improve results. The professional implementation from SciKit-Learn allows for a user defined metric, so we could potentially implement this change to improve our performance.

Despite our strong results, care should be taken in expecting these results to generalize to mail, because the input data will likely be of a very different form. Although one might expect that an image of a letter that showed digits in a higher resolution than 28×28 pixels would be easier to predict, results in machine learning are often counter-intuitive, and nothing should be taken for granted. Secondly, if we hope to sort letters, we will also need to classify letters. The increase in the number of classes from which we much choose may also complicate the problem. So, although our results here are certainly promising, there are several major challenges to address before we have a functioning mail sorting system.

4 Contributions

Michael Robertson completed all the work and writing in the github branch 'robertson', and did not work with Yilin Yang whatsoever.

References

[1] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009.