Final Paper: Digit Recognition

Philip Hasse

12/12/16

Abstract

This paper goes into the history of digit recognition from the point of view of many different experiments done in order to get a great solution. I then go into a detailed description of my methodology and what I was trying to do with my experiments. These experiments are talked about in detail with both their benefits and flaws that I concluded after performing them. Finally I conclude on what my paper achieved, in my opinion, and what I learned from these experiments.

Introduction

Machine learning is an extremely interesting and useful way to solve problems that would normally take a long time for humans to do on a large scale. The project that I have chosen to work on is the Digit Recognition Kaggle competition. The competition challenges users to create an algorithm that will be able to recognize hand written digits and guess what each digit is. This isn't a new idea; there are many different attempts of creating this algorithm over the years.

This type of algorithm can be extremely useful in the real world because almost everyone needs to write math down on paper. If these numbers can be recognized with great accuracy, then we could just scan the page and print it onto the computer for a much more legible, and convenient copy. In theory this is a great idea and should be used if it can be readily available. However, the algorithm needs to be both extremely accurate and have a fairly short computation time. This is a general rule for all machine learning algorithms because they are being designed to help the person not make it a more complicated process.

In my experiments, I attempted to achieve a great accuracy and a short computation time. Later in the paper you will see that computation time was an issue, so some different approaches needed to be taken in order to try and solve this issue. Overall I feel like a found a decent solution to the proposed problem; however it isn't the best solution, or the only solution that has been found to this proposed problem

Background/Related Work

The idea of using a computer to recognize handwritten digits has been around since the early stages of machine learning. Recognizing handwriting is one of the first ideas that comes to mind when thinking of how a computer can help humans on a consistent and basic basis. A study done on handwritten digit recognition was done as early as 1994. In this study they looked at many different methods of interpreting these handwritten digits using a dataset supplied by the US National Institute of Standards and Technology (NIST). [1]

Two notable methods that were used were a multi-layer neural network and the K-nearest-neighbor methods. The depending on the amount of multiply/add steps in the neural network the error rate was between 0.7% and 1.7%, which is fairly good. In the K-nearest-neighbor method a 1.1% error was obtained, but it wasn't as computationally demanding [1]. Both of these methods seem to have pretty good results overall. It is noted that these tests required supercomputers at the time since it was 1994.

There were notable problems with overfitting at the start of these tests. The dataset needed to be expanded and become more diverse in order for these methods to be useful in the real world. The paper mentions that the human error rate for reading these digits is about 0.2%, which is almost perfect. If we could get the machine learning algorithm just a little to the 0.2% error rate this technology would be incredibly useful. [1]

Methodology

There were three methods I attempted when approaching the solution to this problem. The first method I attempted was a multi-layer perceptron for classification using between 1 and 150 neurons and an eta between .1 and 1 which we compared through a 10-fold cross-validation. I chose it because it seemed like the strongest algorithm at the time and it could get me the best accuracy. The second method I decided to choose was an algorithm that used K-nearest-neighbors to classify the data. This was a method that I fell back on because the first method didn't come out with great results. The final method I decided to try an algorithm that used random forest. This was a method I decided to try because I wanted to try and lower computation time without hurting my accuracy rate too much. Overall these methodologies have individual advantages that could have been used more effectively with a more efficient algorithm.

Experiments

In all of these experiments there were two data sets used in a csv format, a training set, and a testing set. The training set had one column which labeled the number being portrayed and another 784 columns that held a value from 0-255. Each cell represents a pixel on the picture and the value in the cell represents how full the cell is with ink. The testing set is very similar except it doesn’t have a label column telling us what number the pixels represent.

Experiment 1:

In the first experiment I attempted to solve the problem using a multi-layer perceptron. The results of this experiment were fairly poor. The Kaggle given dataset was very large and my algorithm wouldn’t run without shrinking the dataset significantly. So the training and testing set were both were used with 1000 samples. The algorithm took approximately 9 hours to run, and my first though during this time was that a long computation time would correlate to good results. However, the results were just as poor as the computation time. The solution was found using an eta of .1 and 122 neurons. The cross-validation score came out to approximately -6357.587 and the accuracy calculated was about 28.486%.

Experiment 2:

In the second experiment I attempted to solve the problem by using a K-nearest-neighbors algorithm. In this experiment I used the default parameters and trained it on the training set and tested it on the testing set. The computation time was about an hour and a half for the full dataset so I shrunk the dataset to 5000 sample in both the training and testing sets. The accuracy calculated using this method was about 92.83%.

Experiment 3:

In the third experiment I attempted to solve the problem by using the Random Forest algorithm recommended by Kaggle. This algorithm ran the fastest even with the enormous dataset. However, even though computation time was about a minute the accuracy obtained wasn't the greatest achieving approximately 69.79% accuracy.

Discussion/Analysis

All three experiments had their warrants to use, however, we can see that using K-nearest-neighbors was the most effective method to solve the problem with only about a ~7.2% error. This may be due to many issues that occurred during experimentation. The Multi-Layer Perceptron that I attempted to implement was likely implemented incorrectly which would explain my poor results. However, because of the amount of time the program took to run, it wasn't a plausible strategy to continue changing code and seeing how the results were affected.

After searching for some help online I thought to use K-nearest neighbors, which ended up giving me the best results. The long computation time for the entire dataset encouraged me to give Random Forest a try and see how the results were affected. This method is very quick however the accuracy is greatly affected. With all of these experiments in mind and the result of 92.83% accuracy, I thought that this was a decent enough accuracy rate to finish off my experimentation.

Conclusion

This paper goes over the history of handwritten digit recognition in machine learning, different methods of recognizing digits accurately, and my personal experiments that I attempted in order to solve this problem. The idea of recognizing handwritten digits is not a new one and it is shown that some great solutions were found even as early as 1994. The main issue is that an error rate that is larger than 0.2% can massively change the meaning of these numbers if used on a large scale. The paper approaches 2 main methods of solving this problem using a multi-layer neural networks and K-nearest-neighbors. My experiments mostly fell flat with a huge error using a Multi-Layer Perceptron, a significantly smaller, but still not satisfactory error using the Random Forest algorithm, and a satisfactory error using K-nearest-neighbors. Overall this paper shows how much I learned from doing these experiments and researching the history of this type of machine learning.

Bibliography

[1]L. Bottou, C. Cortes, J. Denker, H. Drucker and I. Guyon, "Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition", pp. 1-11, 1994.

[2]V. Paruchuri, "K nearest neighbors in python: A tutorial", *Dataquest Blog*, 2016. [Online]. Available: https://www.dataquest.io/blog/k-nearest-neighbors-in-python/. [Accessed: 05- Dec- 2016].