MNIST Dataset: Algorithm Comparison

Morgan Baker

12/12/2016

Section I: Abstract

The project I am working on is an old machine learning problem. It is the problem of teaching a machine to recognize handwritten numbers. I am trying to improve on the problem, first by creating an image reader and using different machine learning algorithms to try and achieve the highest accuracy.

Section II: Intro

For the sake of data, Kaggle is using the dataset from the Mixed National Institute of Standards and Technology (MNIST). There are two datasets, one of which is to test, and the other is to train. The training set has 785 columns, one of which being the actual digit represented. and the other 784 columns are to show values of a 28*28 grid. the other file, test.csv, foregoes the digit column, making the machine create an educated guess. I needed to make the machine train on what was given in the training dataset, and then return the results from the test dataset in a csv file. My goal for the project was to get an algorithm to reach 98 percent accuracy.

Section III: Background

Due to being one of the oldest problems in machine history, there is a lot of background data to go off on. The Kaggle forums provided a lot of insight, and provided novices with code they could use to begin their journey. Most of the forum posts told me to either turn to a neural network or to use the Random Forest algorithm. Alas, there are also a specific group of people who have achieved a perfect score on the Kaggle leaderboards. However, these Kaggle users have been accused of cheating. The way these users got a perfect score is by parsing the MNIST database for the specific set of integers that Kaggle is using, and then from there fill out a csv file. I want to believe that these users were wholesome in how they got to their scores, but thanks to the forum posts, I am not completely sure.

Section IV: Methodology

To hopefully figure out this problem, I will be testing the dataset using Python code. I have already created a script to figure out how the average number in train.csv looks. This takes the label digit, and maps every pixel from those columns onto the same plot. I also normalize the values, taking a value of 1-255 and making the value a floating decimal between 0 and 1. I then have four different scripts creating four different learning algorithms, and taking the algorithm with the highest accuracy. The algorithms I am thinking of are K Nearest Neighbors, a Support Vector Classifier (SVC), a Convolutional Neural Network, and the Random Forest algorithm.

Section V: Experiments

The first implementation I decided to try was K Nearest Neighbors. However, because I was having trouble with both creating my own algorithm and importing the algorithm from sklearn, this was not the algorithm that I was going to submit. The algorithm I wrote for K Nearest

Neighbors was using four .txt files instead of the two .csv files. This made the algorithm pretty much invalid for the competition, but I just wanted to see if I could program the algorithm myself. The KNN algorithm gave me a pleasant surprise of being 94 percent accurate with the text files, although I am not sure how successful the algorithm would have been otherwise. For posterity’s sake, I have included the code for this algorithm in the code folder.

I then started to think of other, simpler ways I could sort the data. Sklearn again came to mind, but I was not sure about which algorithm to use. I decided to use a support vector classifier for my next algorithm. Using sci-kit learn as the backbone algorithm, I got about 93.5% accuracy when it was submitted to Kaggle. This, I thought, was not the best of the algorithms, but it was much better than just plain guessing. I then started to think about neural networks. Since the forums had started talking about making convolutional neural networks, I thought it would be a good idea for me to use one. I use the Theano python package, a deep learning library for python. However, Theano started to be a bit weird, so I looked for another option. Lasagne comes in here, running on top of Theano to provide a simple way to run a convolutional neural network. Theano and lasagne together got me to around 95% after 15 epochs. I felt sort of gypped, because of all the praise cnns had been given on the Kaggle Forums. This is when I found the Random Forest algorithm, which uses multiple decision trees to all vote on what the number is. The best part was yet to come, however. Sci-kit learn had a implementation of the random forest algorithm, ready to go right out of the box. I ended up running the algorithm, and got about a 96.5% accuracy.

Section VI: Discussion

The results were not what I hypothesized. I had expected the random forest algorithm to do extremely well, when it was almost on par with the simple convolutional neural network I used from lasagne. While both got around 96%, I can see more potential with a neural network rather than the random forest, since some neural networks from the forums have gotten to >99%, where random forests have not been seen in the forums except for the use of benchmark code. There was even a K nearest neighbors algorithm that was able to achieve 97.1% accuracy. Alas, none of my algorithms reached the goal of 98% accuracy, but I am still new to the field of data science. One reason for why this goal may not have been achievable is a hardware problem. While I was able to get a simple convolutional neural network working, there was the potential of a problem with having a run time that exceeded more than two hours. Some of the algorithms on Kaggle had taken the better part of seven hours to complete, where I do not think my machine would have enough memory to go that far. I am glad that I have some results, but they were not necessarily the ones I was hoping for.

Section VII: Conclusion

In conclusion, the algorithms I tested did not quite make the cut. However, there are still some algorithms I have yet to explore in the world of data science, and hopefully those algorithms have a better accuracy. The final ranking on Kaggle at the time of writing was 943 out of 1379. This leaves me in the top 69 percent of competitors. It is not something I am proud of, but for now, I will take it.

Section VIII: References

Jethva, Minesh. "Knn from Scratch." *Digit Recognizer | Kaggle*. Kaggle, 12 Nov. 2016. Web. 12 Dec. 2016.

Kitalishvili, Koba. "Python and Convolutional Neural Networks." *Digit Recognizer | Kaggle*. Kaggle, 15 Oct. 2015. Web. 12 Dec. 2016.

"Scikit-learn." *Scikit-learn: Machine Learning in Python — Scikit-learn 0.18.1 Documentation*. INRIA, 20 Nov. 2016. Web. 12 Dec. 2016.

Theano Development Team. "Theano: A Python Framework for Fast Computation of Mathematical Expressions." *ArXiv E-prints* Abs/1605.02688 (2016): n. pag. *Theano*. Theano Development Team, May 2016. Web. 12 Dec. 2016.

"Welcome to Lasagne¶." *Welcome to Lasagne — Lasagne 0.2.dev1 Documentation*. Lasagne Contributors, 20 Sept. 2014. Web. 12 Dec. 2016.