Capstone Report

Digit Recognizer

Author: Allison Senden

October 28th, 2019

**I. Definition**

**Project Overview**

Digit recognition is being studying by those practicing their computer vision and machine learning skills. I specifically chose this project topic because I would like to get into some work at my current job where they are helping to design a model that can handle taking in and interpreting the information on faxes. This would include handwritten text and digits. I think that many companies can benefit by having a model that can take handwritten notes and interpret them into useable information. This would help to alleviate someone manually completing this task which I could imagine would take hours. If the company can use a model that can do the same work, then they can pay that person to do some more meaningful work that help get them to their goal in better way.

**Problem Statement**

The goal is to be able to correctly identify the handwritten digit by applying algorithms that learn based on the training set, which is thousands of pictures of handwritten digits. I want to be able to create a model/algorithm that performs better than the baseline models. I will have to do research into neural networks and which algorithms and layers perform best for image classification. The data I will be using is a very commonly known dataset called MNIST. It contains hundreds of thousands of pictures of handwritten digits that I will be able to use to solve the problem. More information about the MNIST dataset can be found here. The training data consists of 60,000 images and the test set contains 10,000 images.

I was also able to find a similar Kaggle competition (https://www.kaggle.com/c/digit-recognizer/overview). Here you can see many people taking the challenge to classify the handwritten digits properly.

**Metrics**

Some models that are historically use to evaluate this dataset are SVM and k-nearest neighbors. I will use these 2 models as a baseline for my work, and then hopefully I create and fine tune a model that will perform even better than these baselines. K-nearest neighbor is good fit for this problem since its goal is to essentially cluster the alike images into the same cluster. This helps me because it’s choosing the images that most likely are the same as those in that same cluster. For me, I can make those clusters each of the different digits 0-9. Then when it’s predicting, it will choose the cluster that the digit resembles the most. In order to evaluate the performance of the model I create, I will evaluate the accuracy. This will tell me the percentage of photos that were correctly classified as the right digit. We only care about when our model is correctly classifying images so that’s why I chose accuracy as the evaluation metric.

**II. Analysis**

**Data Exploration**

At first, I found the Kaggle competition and started by using the data that they provided directly. Once reading in the data, I quickly found out that the test data did not have any labels for it. This is because for the competition you are supposed to submit your best model and they will run it against the test data’s labels to calculate accuracy for you. That way the answer isn’t out there. But for this project, I will need the test labels, so I decided to do some research and see if I can find the data from another source. I was able to find out the the MNIST dataset was actually preloaded into the keras module. All I had to do was import it like I would any other library to use it. So that was how I actually got the data used in my analysis. In both cases though, there were still 60,000 training images and 10,000 test images. Later in the notebook, the training data was broken down a bit more to leave 54,000 training images and 6,000 validation images. Upon further analysis of the training dataset, we can see that we have a good, balanced dataset. Each of the handwritten number images have a fairly similar amount of samples that the model can learn from. I will say there were some slight differences in the datasets being that they came from 2 different sources. From Kaggle, each row represented an image and each column represented the pixel in the image. I was able to read it into a pandas dataframe just like that. That means if I wanted to use the Kaggle data, I would need to transform the dataframe into an array of arrays. On the other hand, using the data from keras, it came already packaged as an array of arrays. Also, they took it a step further and it’s already been pre-processed a bit in the sense that they reshaped the images to represent the size 28x28. That is also fine, but if we need to return to the original size, we just have to do a bit of manipulating. I did have to return to the original size so you will find the code to go between the 2 sizes in the notebook. I needed to do this because the algorithms expected 2D inputs, not the 3 dimensions that come with reshaping the images. Another preprocessing step that was done was transforming the X-data into floats and the labels into integers. This just makes handling the pixels easier and keeps the labels as whole numbers (no decimals). Since we have 10 different integers as the labels, we can one-hot encode these to make it easier to handle when feeding it into the model. Originally I thought I was going to need to reshape the images again to have a single channel on top of being resized to 28x28, but then realized I did not as the models can't intake dimensions higher than 2.



**Exploratory Visualization**

I started off by first exploring some of the images that I was going to try classifying. We can see that the images are pretty easy to classify to a human eye, but since they are a bit odd shaped sometimes, I can see how a computer or algorithm might have a bit of difficulty. When visualizing the images, notice we have to use the reshaped images (28x28). This is so that we give it the 2D affect which allows us to plot the image's pixels. I thought it was important to get an idea of what the handwritten samples look like before deciding the preprocessing steps and defining the model architecture.



**Algorithms and Techniques**

In order to solve this problem, I will implement a k-nearest neighbor model that’s tuned so that it can classify the images properly. I feel that it best suites the problem at hand. I think choosing a k=10 as there are 10 numbers (0-9) makes the most sense, but we will see once we do more analysis. It also turns out that this is one of the most commonly used algorithms to attack this classificaiton of images problem. This algorithm is commonly used for this application because it clusters inputs together and then tries to classify those like features similarily. This would make sense for our problem at hand, because the algorithm can find similarities in the images and then group them based on its findings. We have 10 different numbers that it can classify so I would expect to see 10 different clusters - one for each number possibility. For the benchmark model, as I mentioned earlier, I will apply a SVC model. I like to use this model most of the time for benchmarking. It performs decently well for a wide variety of problems. Although, I found that it's best at classification and regression problems via this site: https://medium.com/@dataturks/understanding-svms-for-image-classification-cf4f01232700. The way that SVM works is it calculates a few different boundries and by comparing the differences, it can make a guess as to the classification it should be. This will work in our case because it can calculate the differences of boundries of the written line and where it is on the image to be able to decipher what number it is. After applying both algorithms, I will compare the performance by using the accuracy metric. This means, I will compare the number of images the model classified correctly vs. the incorrectly classified ones. This is the best metric for our use case as we only care how many images we are able to correclty classify.

**Benchmark**

Some models that are historically use to evaluate this dataset are SVM and k-nearest neighbors. I will use these 2 models as a baseline for my work, and then hopefully I create and fine tune a model that will perform even better than these baselines. K-nearest neighbor is good fit for this problem since its goal is to essentially cluster the alike images into the same cluster. This helps me because it’s choosing the images that most likely are the same as those in that same cluster. For me, I can make those clusters each of the different digits 0-9. Then when it’s predicting, it will choose the cluster that the digit resembles the most. After running the benchmark model, we get an accuracy score of about 11%. So this is not a very good result, but remember, this is just our benchmark. From here, we hope to improve the classification by using a different model architecture.

**III. Methodology**

**Data Preprocessing**

I sort of explained the preprocessing steps in another section prior to this, but let's cover again to be very clear. Since I used the data from keras, it came already packaged as an array of arrays. The images had already been preprocessed a bit so they were resized to allow the image have the shape (28x28). Since we chose to use SVC and k-nearest neighbor as our models, we have to ensure that those models will be able to intake the data we are giving it. Both of those models expect 2D data, which means the reshaped 3D data to allow the size of 28x28 will need to be transformed. So we completed that tranformation step and noticed some other things that needed to be done. The X-data was turned into floats and the labels (y-data) into integers. This was done on both the training and the test datasets. This just makes handling the pixels easier and keeps the labels as whole numbers (no decimals). Since we have 10 different integers as the labels, we can one-hot encode these to make it easier to handle when feeding it into the model. Originally I thought I was going to need to reshape the images again to have a single channel on top of being resized to 28x28, but then realized I did not as the models can't intake dimensions higher than 2. So you will see an extra cell in the preprocessing section of the notebook, although I admit, it didn't need to be done.

**Implementation**

Next, I implemented a k-nearest neighbor model architecture and ran it against our dataset. The first order of business was to determine the k-value that would give me the best results. So I looped through 1-30 as options for k and printed off the accuracies as it was looping through them. This process took a pretty long time, but worth it because I did find a k-value that would give me a really good result! Deciding the k-value took a bit of time, but in the end, it was worth it. The optimal k-value turned out to be 1. When I then ran the k-nearest neighbor model with k=1, it ended up scoring a 97% accuracy! Awesome! I was really happy with this so I stuck to it for the final model.

**Refinement**

During the implementation phase, I mentioned that I needed to determine the optimal k-value. This was actually something that came to me after I tried to implement the k-nearest neighbor model with a k-value of 10. I thought that 10 made the most sense in my mind because there were 10 different classes or numbers that we were trying to predict, so naturally that is how many clusters would appear in the data. After looking more at the documentation, I realized that I should actually test that theory instead of assuming that 10 was the optimal value for k. So I went back and refined my implementation to include looking for the best k-value. This also meant that I was going to need to further break down the data so that I would have a validation set to be able to calculate the accuracies of each k-value I wanted to test. So I broke the training set down further into a validation and training set. Then I was able to use those 2 breakdowns to calculate the optimal k, and then once chosen, use the training data to fit the model and test data to predict and find the accuracy of the model.

**IV. Results**

**Model Evaluation and Validation**

I chose this model for many of the reasons I listed above. It is a clustering algorithm that's good and classification problems and picking out similarities between the feature inputs. For this classification of handwritten numbers, that description fits perfectly. Not to mention the fact that through research, I found many others have chosen the route of applying this algorithm to image processing problems because of it's strengths. That is why we can see that this algorithm does much better than the benchmark model SVC. Upon printing the classification\_report, we can see the accuracy in which the model is able to classify each of the numbers. They all have very high accuracy rates (or in the classification\_report it's referred to as precision). Since the training handwritten numbers come from a different pool of people than the test handwritten numbers, we see that this model can indeed learn from different people's handwriting. Meaning the different angles they write at, the handedness of them, etc. The model can predict the numbers with a 97% accuracy even given different datasets, so I would say that this solution has the ability to be generalized to unseen data pretty well.



**Justification**

I believe the output of the k-nearest neighbor model does a great job at solving the problem at hand - correct classification of handwritten numbers. Since it performs at a 97% accuracy, we can safely assume that most of the time the model is doing a great job at predicting the correct number. This is by far a much better performance than the benchmark SVC model at 11%. I think that the attributes of a k-nearest neighbor algorithm are better suited to image classificaiton problems than SVC. Although, with some fine tuning, I think that we could have gotten a better result from the SVC. Since it was just the benchmark, we didn't fuss with trying to make it perform better.

**V. Conclusion**

**Free-Form Visualization**

I wanted to show the results of the model's prediction for this portion of the project. You can see that I've printed the image that is classified and the model's prediciton of what number it should be. You can put in any image you like, but I've just chose one at random for visualization purposes. This is mostly for validation purposes and checking the results of the model are truly accurate or what we would expect. Going by this, I think it's safe to say it's what we expected!



**Reflection**

I found this project extremely interesting. It was cool to work with image data for a change since it's much different then we are used to with text and other data types filling the columns instead of pixel data. I learned a lot about how to deal with pixels, image sizing and more. I was very surprised to learn that the optimal value for k ended up being 1. I thought for sure that it was going to be 10. It was crazy to see the accuracy actually decreased as the k value increased. The most difficult part of the project for me was figuring out the image sizing. I'm not used to working with arrays of numbers as a representation of an image. It was a bit more abstract and took me a minute to wrap my head around. This is why visualizing some of the images that we were trying to classify was so important to me. Overall, I'm happy with the results of the model and project.

**Improvement**

To improve upon this project, I would have liked to get it running in the GPU instance. I could spend some time debugging the errors I was running into with AWS Console to have the ability to play with the models a bit more as they would take much less time to train. Another thing that I would like to do is take the project one step further. In my free time, I will probably experiment with applying a keras neural network to it as well. This will allow me to pass in more than the 2 dimensions that SVC and k-nearest neighbor require.