Digit Recognizer

Learning Computer Vision Fundamentals with the Famous MNIST Data

Patrick Humphries (pvhumphr@usc.edu)
INF 552 Machine Learning for Data Science (32458)
Viterbi School of Engineering
University of Southern California
Spring 2020

Abstract Elon Musk has said: "Failure is an option here. If things are not failing, you are not innovating enough". This paper describes two such failures: "My Perceptron" and "My Neural Network" attempt to classify digits from the MNIST dataset. The results of these two attempts are compared with results from the *sklearn* library. The *sklearn* modules were run with default parameters and could score higher with adjustments.

Results

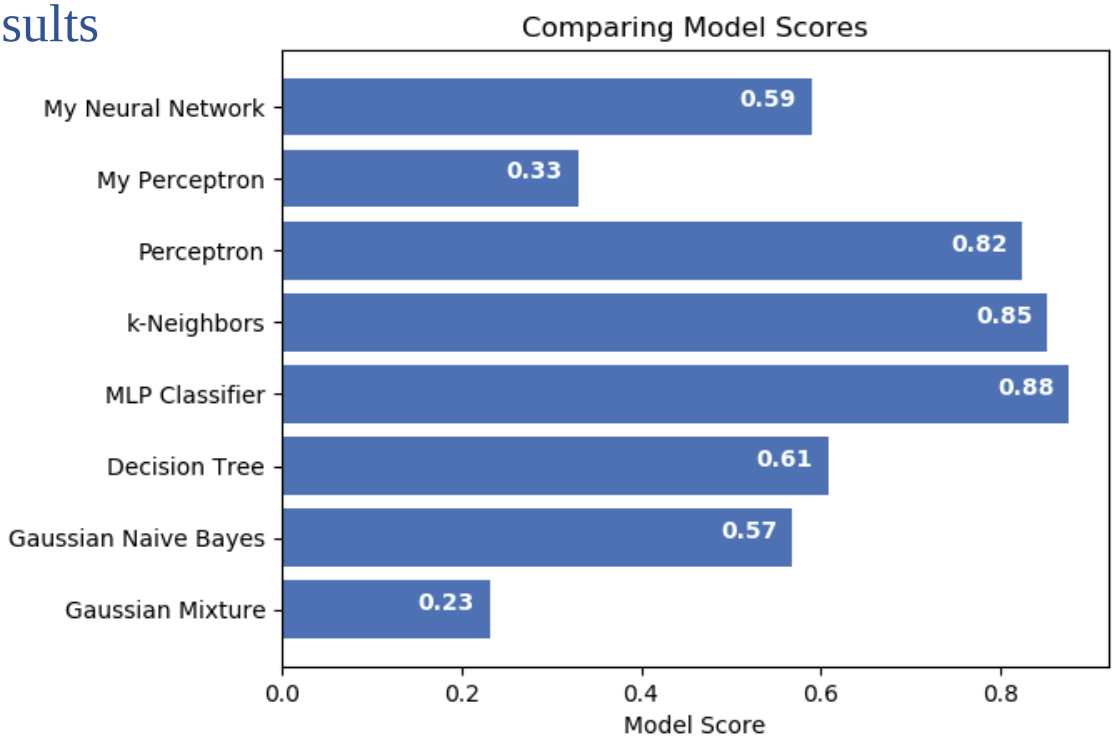


Table of Contents

Data

Source

The "Modified National Institute of Standards and Technology" (MNIST) was released in 1999 under the "Creative Commons Attribution-Share Alike 3.0 License". It contains tens of thousands of handwritten images.

The data was the Digit Recognizer *train.csv* from the Kaggle website (https://www.kaggle.com/c/digit-recognizer). Of the 42,000 images in the *train.csv* dataset, the first 1,000 images were selected using *numpy.genfromtxt* model. Then the data was shuffled into learning and testing datasets using *sklearn.model_selection.train_test_split*. The testing size was set at 25 percent.

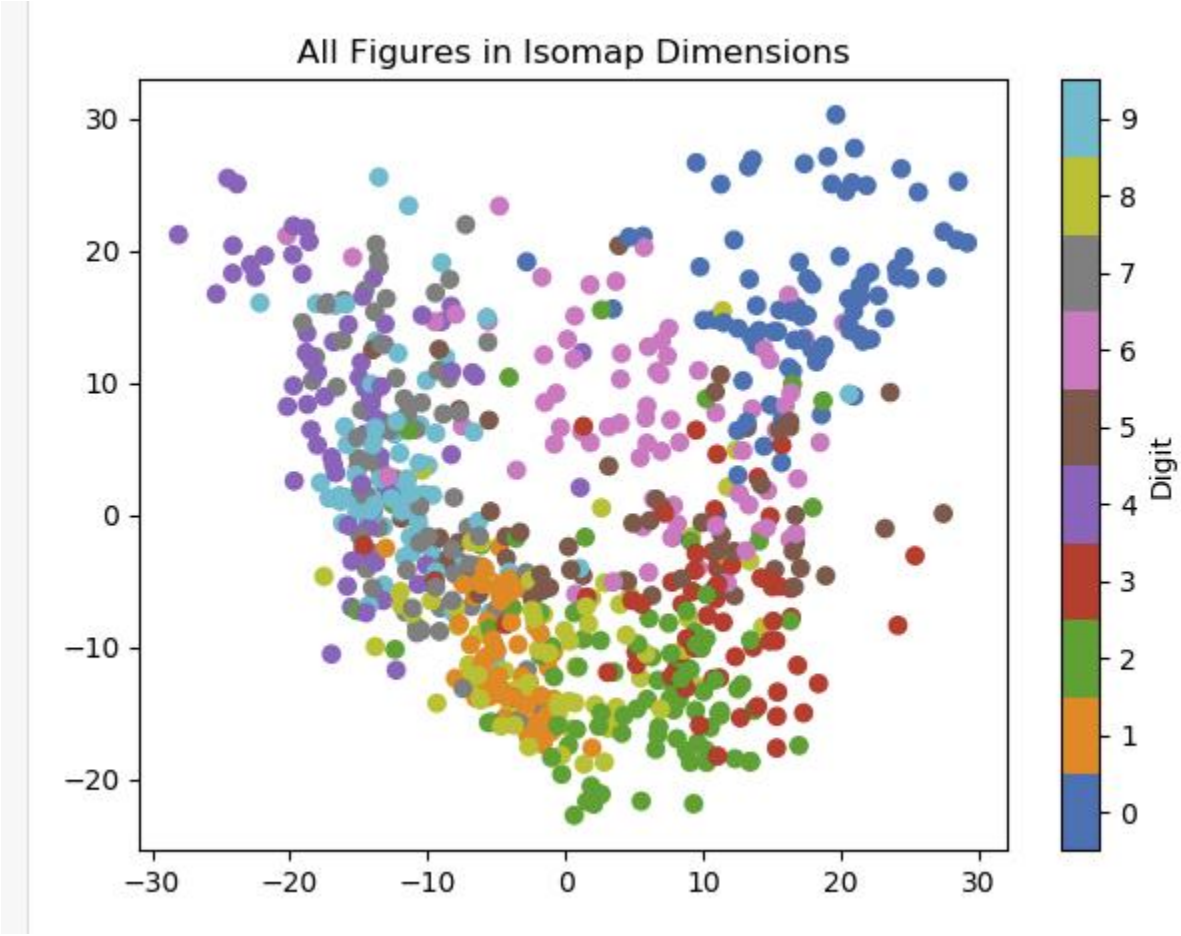
```
Loading testing samples and labels.
X_train.shape: (750, 784) y_train.shape: (750,) X_test.shape: (250, 784) y_test.shape: (250,)
```

Images

The rows in the *train.csv* had the label in the first position followed by the sample composed of 784 values ranging from 0 to 256. The samples were reshaped to 28 by 28 pixels to correspond to the image. The value was normalized to one by dividing by 256. Below is a sample from the first 25 records.

Visualize Data Distributions

The first step in any data analysis is to get an intuitive understanding of the data. This is done in this project by projecting the data in a two-dimension isomap. The samples and labels were reduced to two dimensions using *isomap.transform* module and then plotted.



Evident are definite groupings of digits. However, the boundaries are less than distinct. It will be a challenge to predict a label based on a sample with a high degree of accuracy.

This challenge will be attempted by six sklearn modules using default parameters. While coding from scratch, a perceptron was created (My_Perceptron). This perceptron was not up to the challenge, so a neural network was coded from scratch (My_Neural_Network).

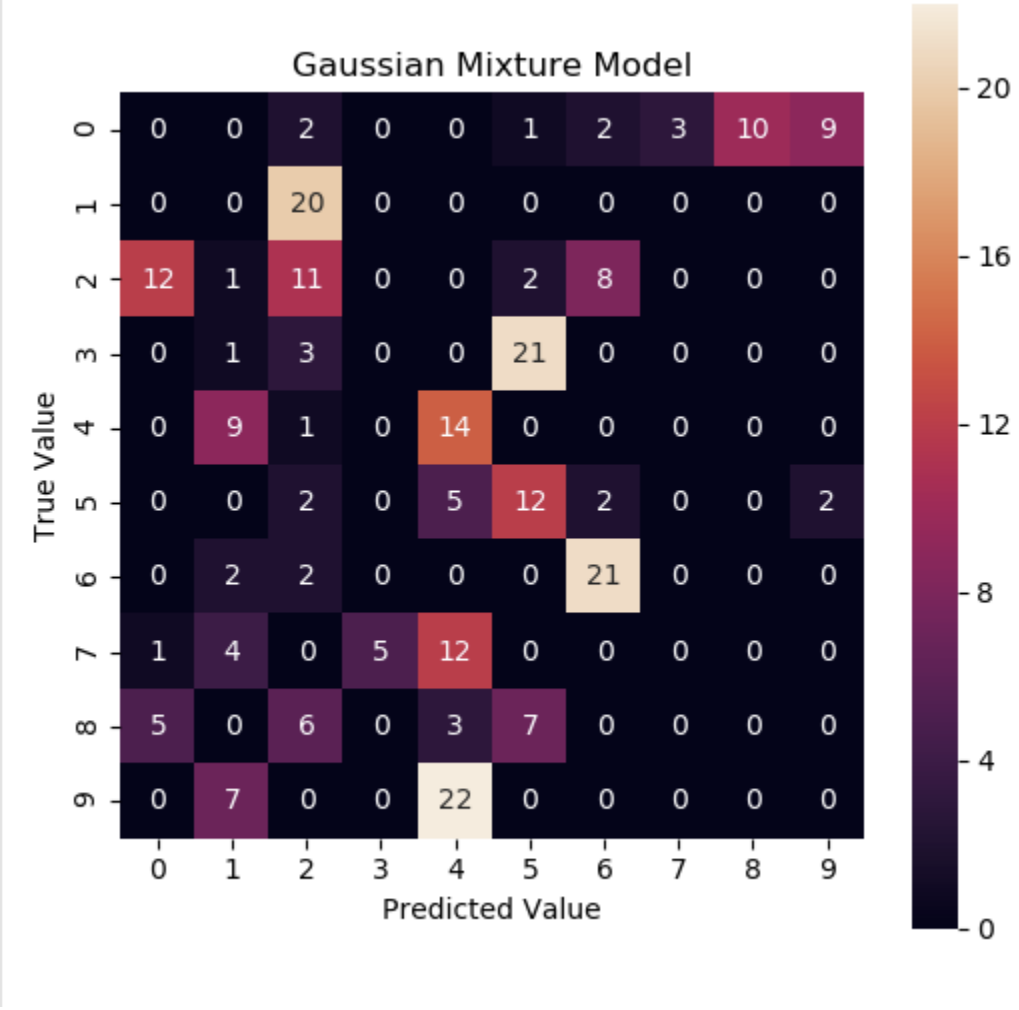
Execution

Gaussian Mixture Model

```
===== Gaussian Mixture Model =====
score: 0.232
```

The Gaussian Mixture Model was expected to be one of the better performers. However, with default parameters it scored a little better that just randomly guessing what the label would be.

Heatmap



As evident from the heatmap, there is no definite distribution of true and predicted values. This model uses probabilities of Gaussian distributions. The commingling of values appeared to diluted the probabilities.

Parameters

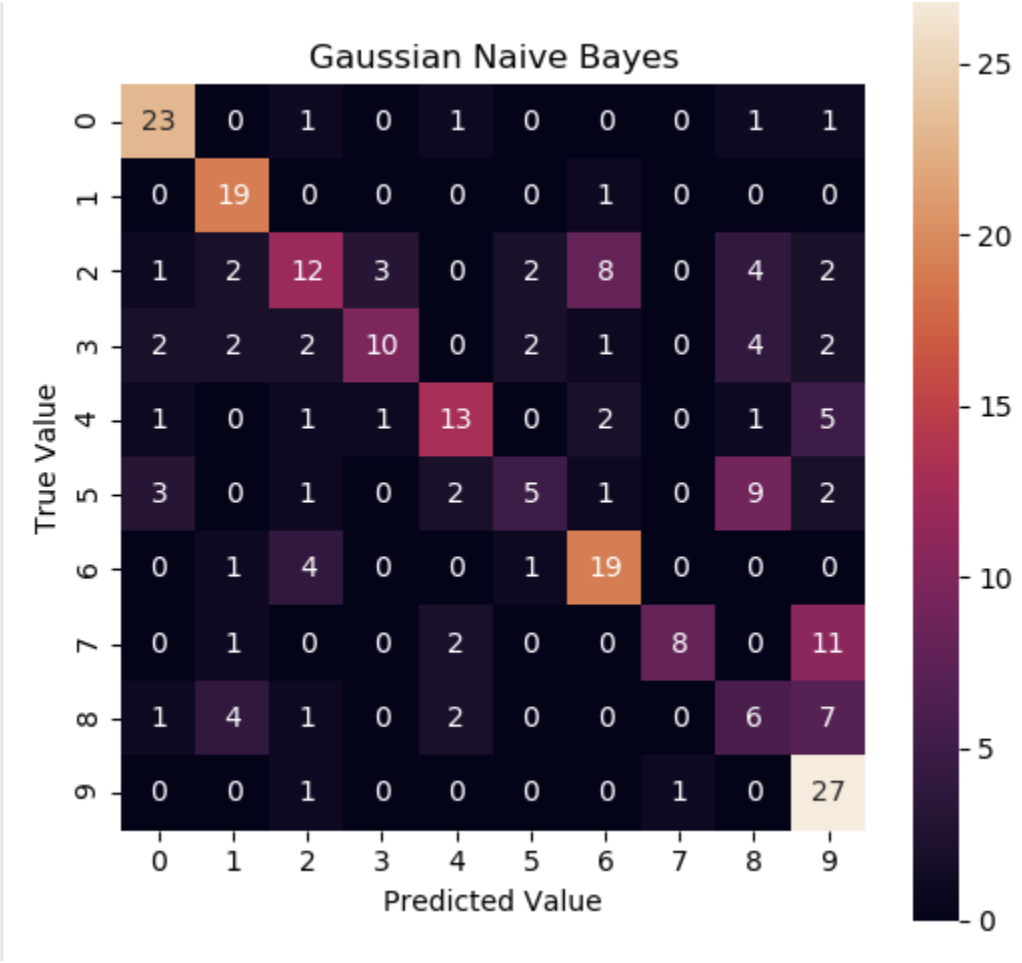
param	value
covariance_type	full
init_params	kmeans
max_iter	1001
means_init	None
n_components	10
n_init	1
precisions_init	None
random_state	None
reg_covar	1e-06
tol	0.001
verbose	0
verbose_interval	10
warm_start	False
weights_init	None

Gaussian Naive Bayes Model

```
===== Gaussian Naive Bayes Model =====
score: 0.568
cross validation scores: [0.50574713 0.60240964 0.6       ]
```

This model faired better, having correct predictions slightly better in half of the instances. There is a beginning of the alignment of true and predicted labels.

Heatmap



Parameters

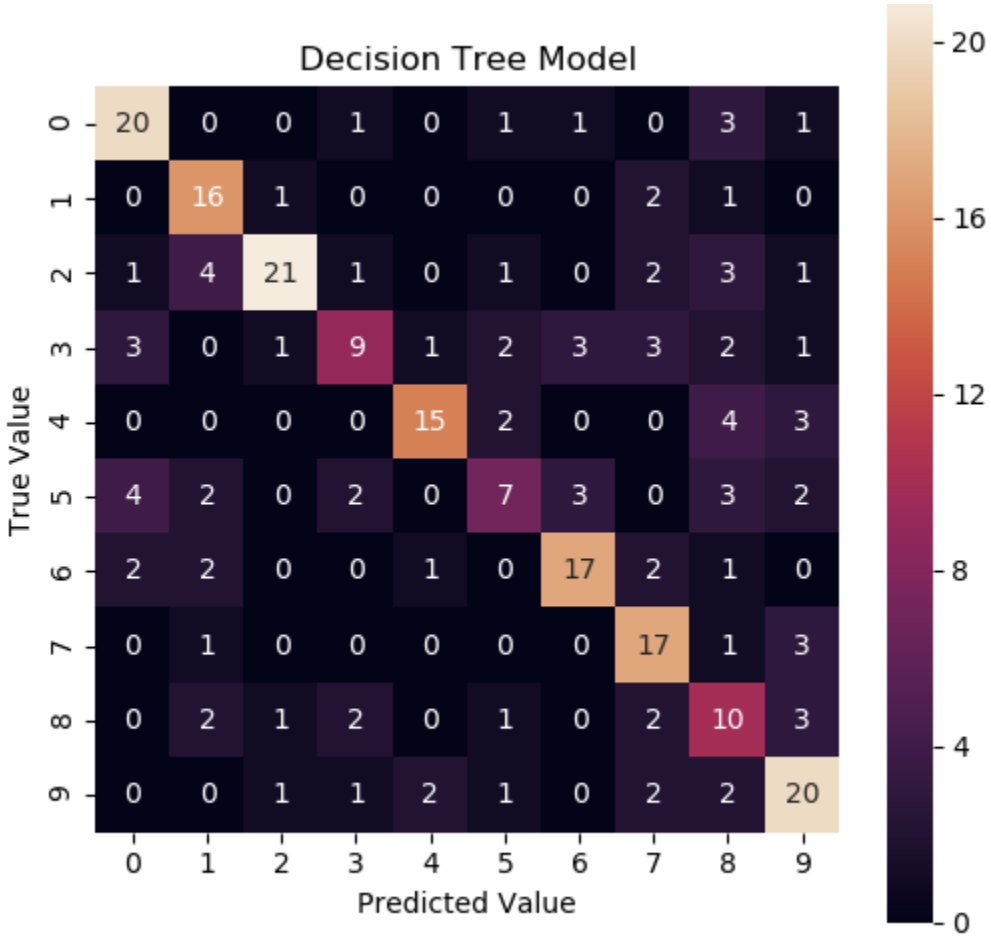
param	value
priors	None
var_smoothing	1e-09

Decision Tree Model

```
===== Decision Tree Model =====
score: 0.608
cross validation scores: [0.47126437 0.46987952 0.425     ]
```

Performance is slightly better, being correct at best 60 percent of the time. As evident in the heatmap, there is a definite correlation between true and predicted labels.

Heatmap



Parameters

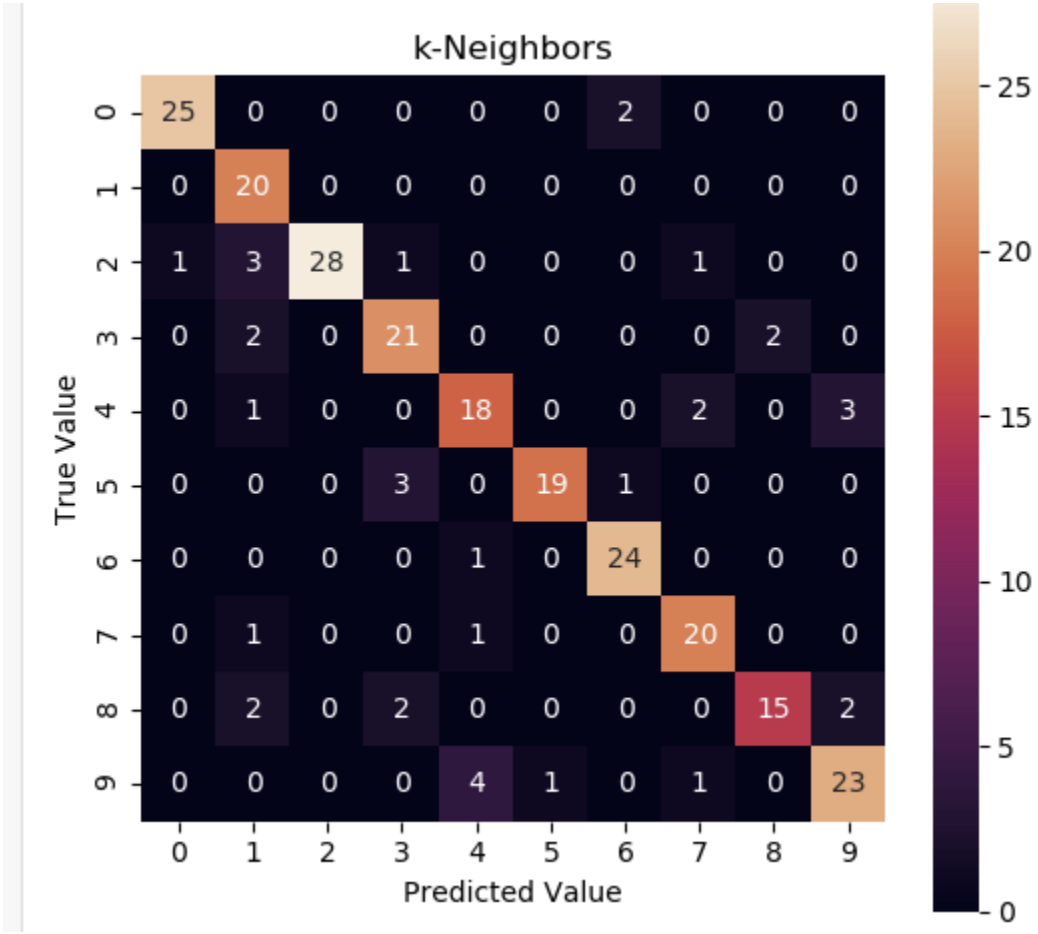
param	value
class_weight	None
criterion	gini
max_depth	None
max_features	None
max_leaf_nodes	None
min_impurity_decrease	0.0
min_impurity_split	None
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0.0
presort	False
random_state	None
splitter	best

k-Neighbors Model

```
===== k-Neighbors Model =====
score: 0.864
cross validation scores: [0.73563218 0.74698795 0.7375    ]
```

Finally, high performance. It would make sense with the clustering of the data. This algorithm looks at a number of closest datapoints of a datapoint to determine its classification.

Heatmap



Parameters

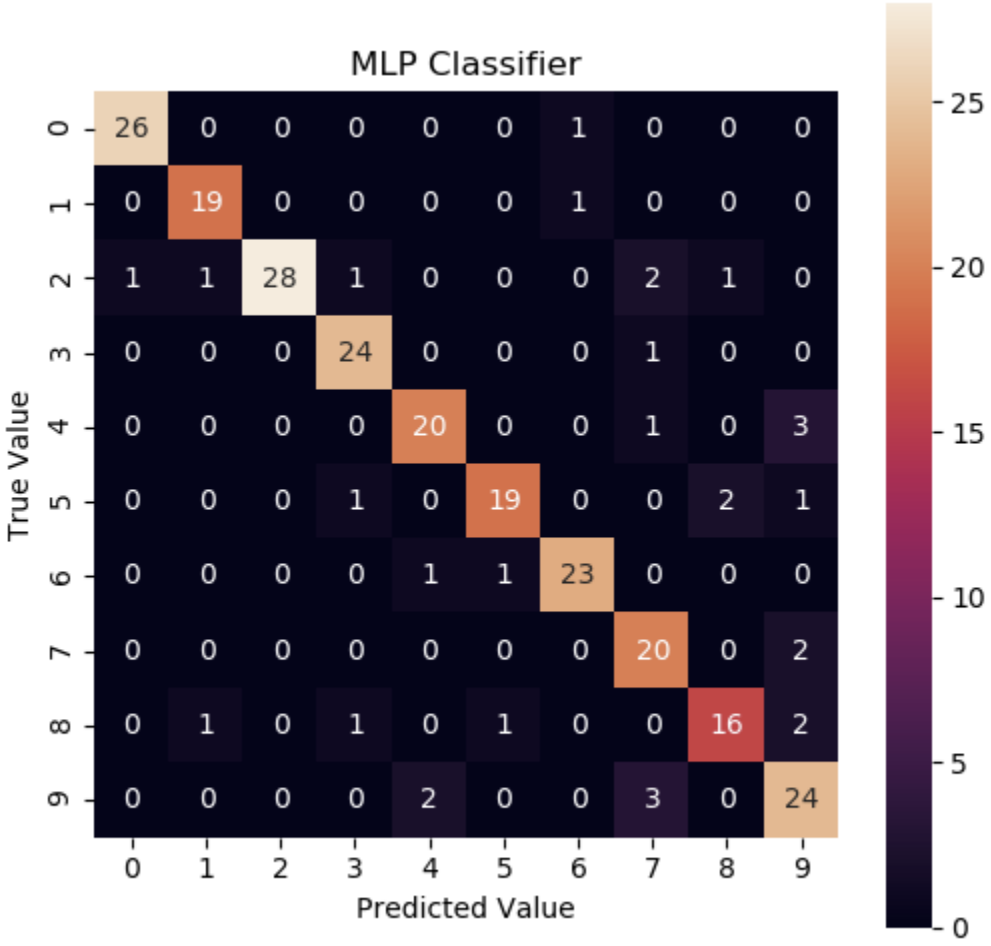
param	value
algorithm	auto
leaf_size	30
metric	minkowski
metric_params	None
n_jobs	None
n_neighbors	3
p	2
weights	uniform

MLP Classifier Model

```
===== MLP Classifier Model =====
score: 0.876
cross validation scores: [0.74712644 0.77108434 0.825     ]
```

The Multiple-Layer Perceptron is a misnomer. It is actually a neural network. As expected, it should be accurate. The definite correlation of true and predicted labels is evident in the heatmap. The parameters are another indication of a neural network. It has parameters for such actions "activation", "epsilon", and "n_iter_no_change".

Heatmap



Parameters

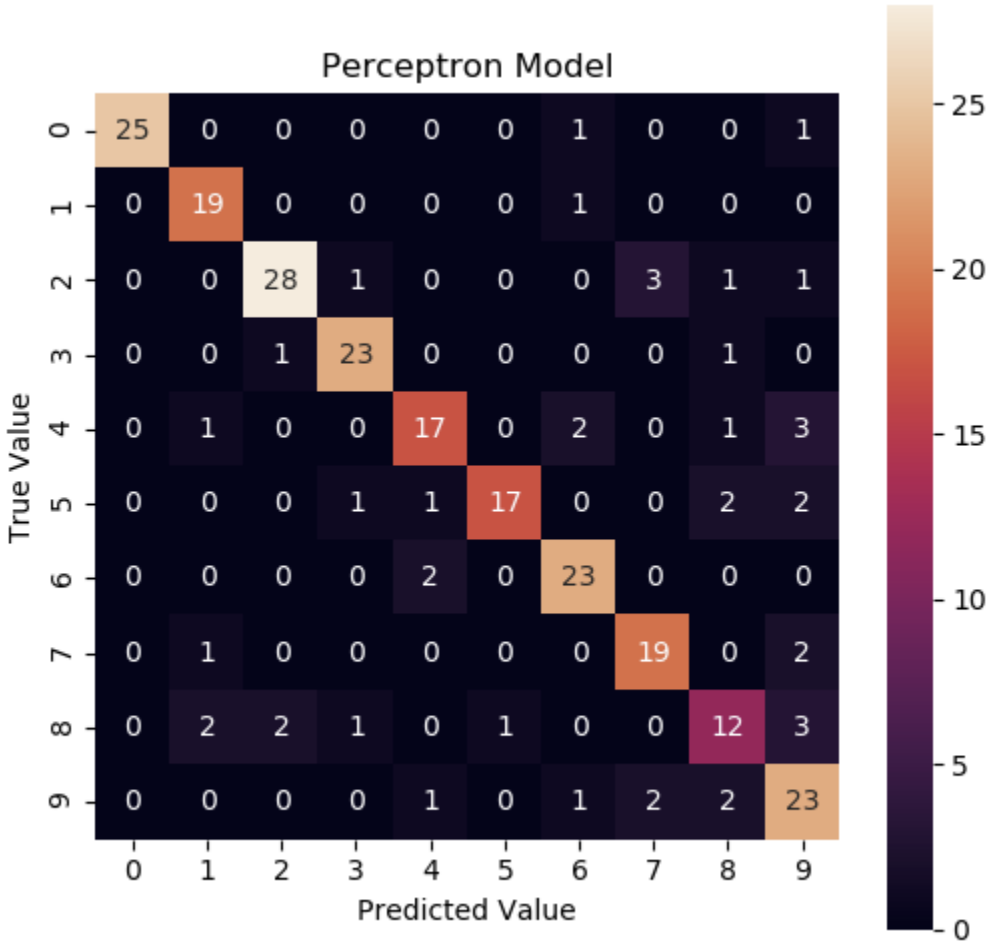
param	value
activation	relu
alpha	0.0001
batch_size	auto
beta_1	0.9
beta_2	0.999
early_stopping	False
epsilon	1e-08
hidden_layer_sizes	(100,)
learning_rate	constant
learning_rate_init	0.001
max_iter	1001
momentum	0.9
n_iter_no_change	10
nesterovs_momentum	True
power_t	0.5
random_state	None
shuffle	True
solver	adam
tol	0.0001
validation_fraction	0.1
verbose	False
warm_start	False

Perceptron Model

```
===== Perceptron Model =====
score: 0.824
cross validation scores: [0.72413793 0.79518072 0.775     ]
```

Where the MLP Classifier was using multiple layers of activations, a simpler version is the Perceptron Model.

Heatmap



Parameters

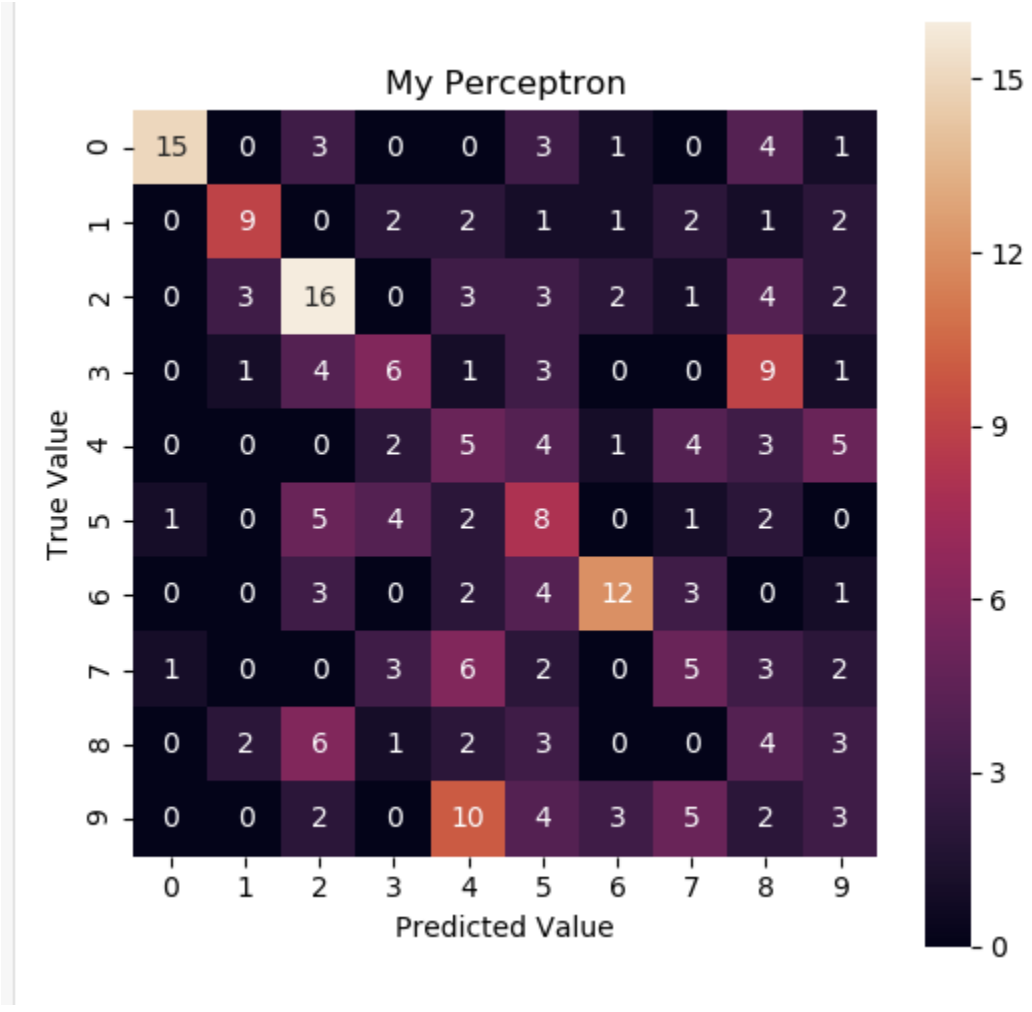
param	value
alpha	0.0001
class_weight	None
early_stopping	False
eta0	1.0
fit_intercept	True
max_iter	1001
n_iter_no_change	5
n_jobs	None
penalty	None
random_state	0
shuffle	True
tol	0.001
validation_fraction	0.1
verbose	0
warm_start	False

My Perceptron Model

```
===== My Perceptron Model =====
score: 0.33
hit: 83 miss: 167 rate: 0.33
```

Well, this is not impressive. This model is actually composed of ten perceptrons. Each perceptron is weighted for a single digit while looking at the entire 784 pixels of an image. The high degree of granularity is present. However, the digits have structure, and each of these perceptrons did not detect structure.

Heatmap



Parameters

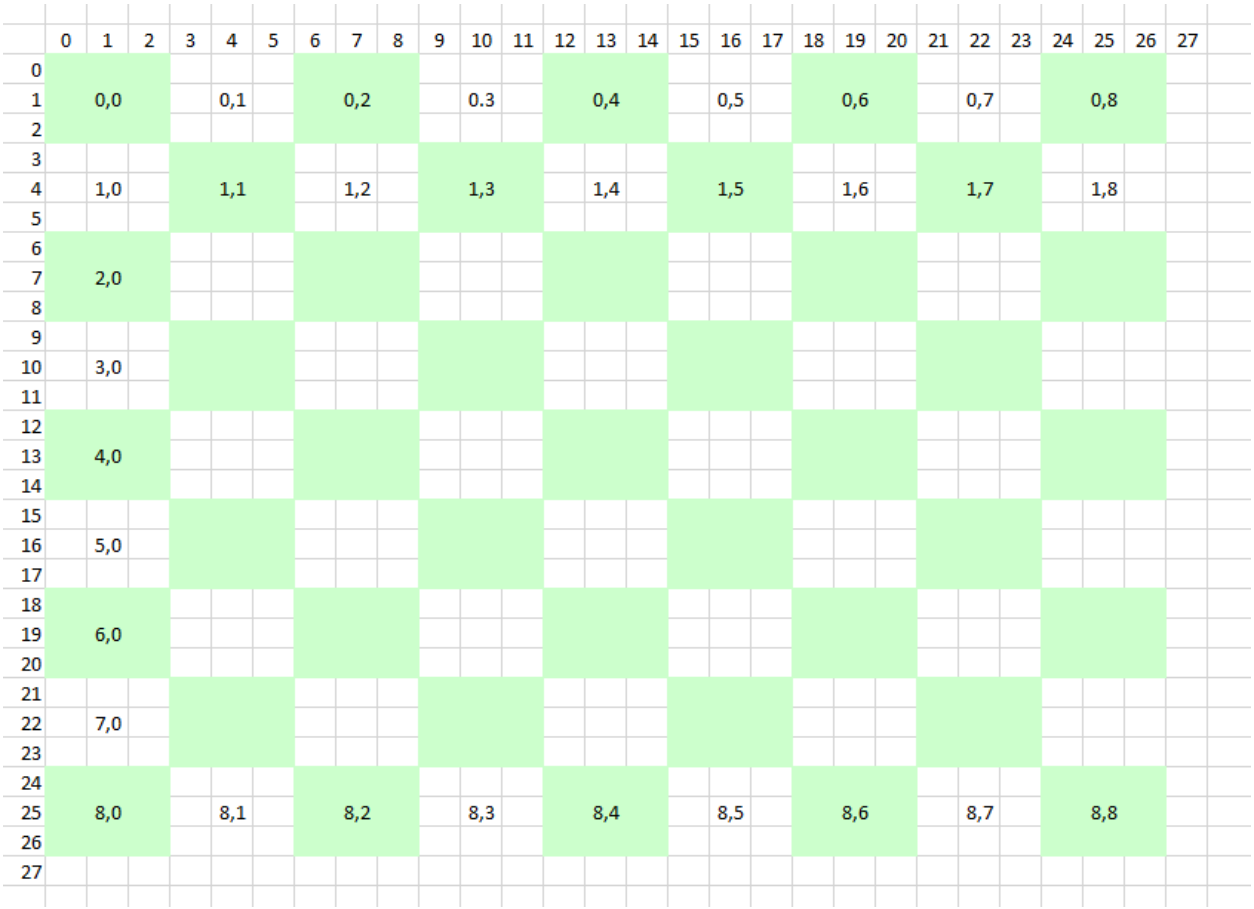
param	value
learning rate (r)	0.0001
bias (b)	1.0
max_iter	1001

My Neural Network Model

```
total_count: 250 total_correct: 148 total_rate: 0.59
```

If My Perceptron was an insect with ten eyes, then My Neural Network is a mutation with 640 eyes. This model contains ten neural networks, one for each digit. Each neural network has 64 perceptrons. Each perceptron was responsible for nine contiguous pixels. This allows each neural network to detect structure.

Sixty-Four Eyes per Digit



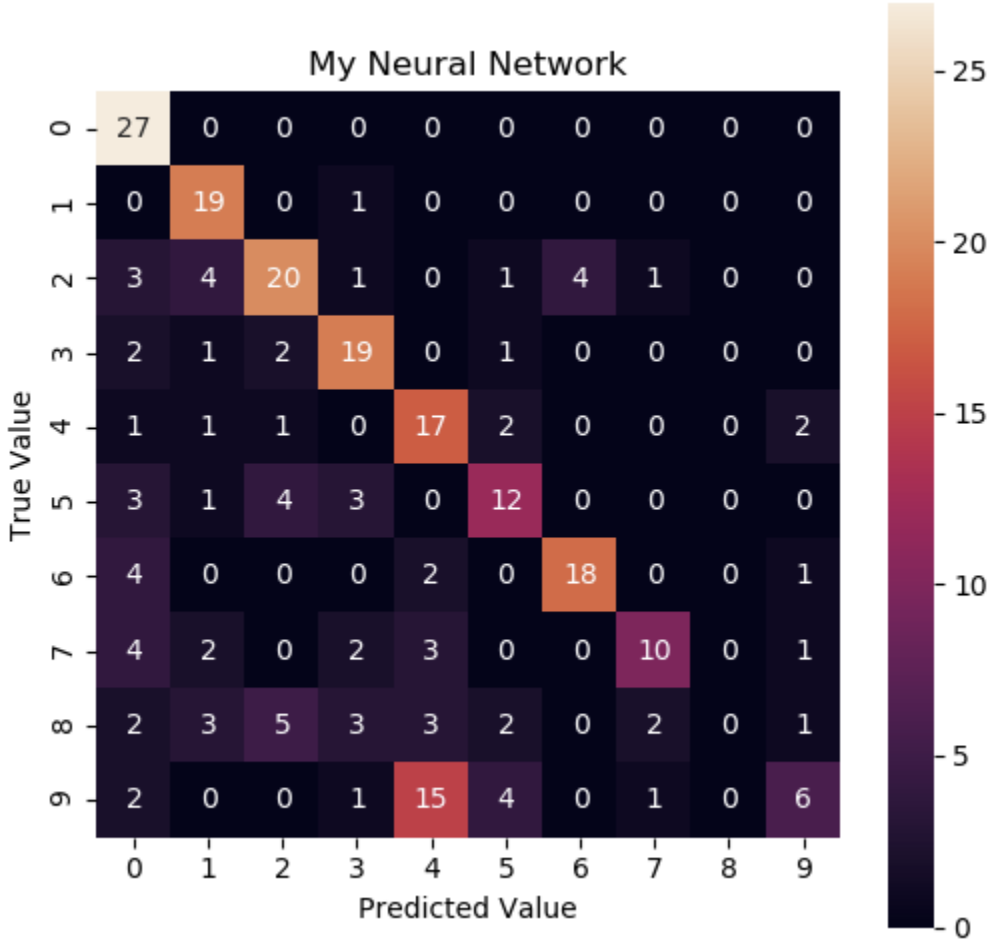
Each image is composed of 28 rows and 28 columns of pixels. Each perceptron is responsible for just nine pixel. As an example, the first perceptron is responsible for pixels (0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2). The 64th perceptron is responsible for (24,24), (24,25), (24,26), (25,24), (25,25), (25,26), (26,24), (26,25), (26,26).

The process for selecting a digit occurs in three steps. First, each perceptron in each neural network averages the activations and votes, 0 or 1, whether the sample provided is the digit for this neural network.

The neural network then averages the votes to yield a probability.

Of the ten neural networks, the one with the highest probability is selected, yielding the digit with the highest probability.

Heatmap

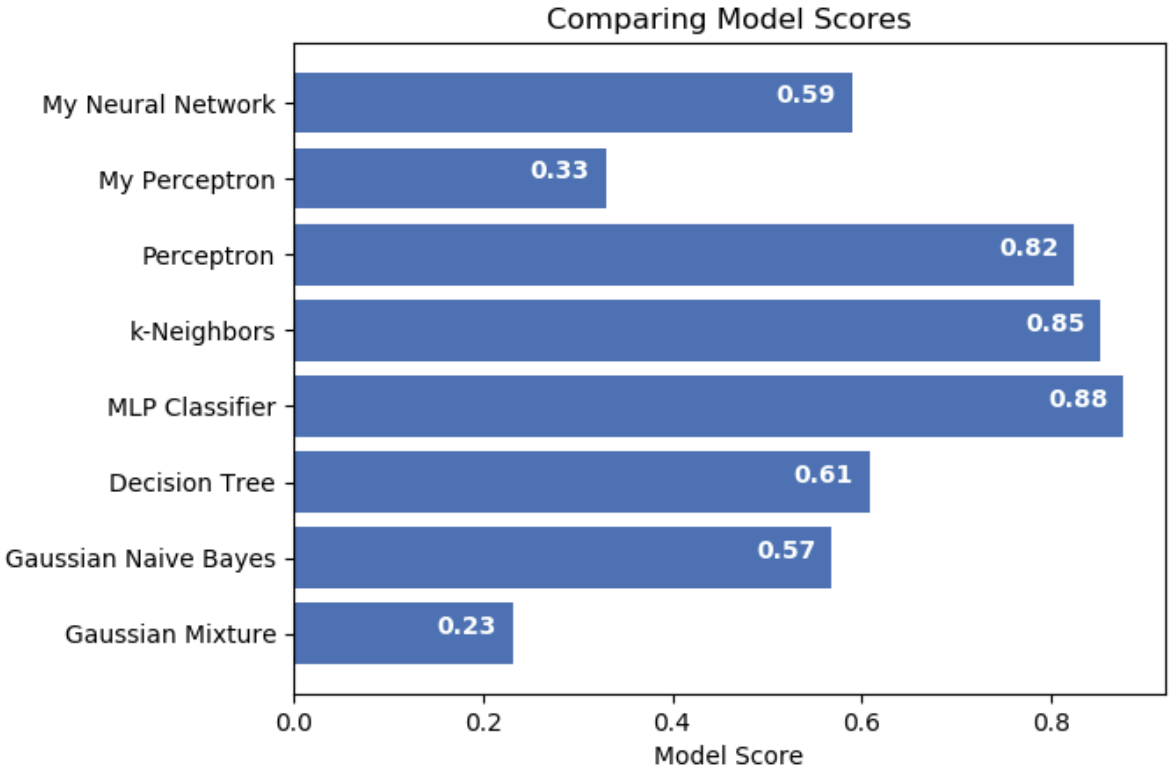


It is evident that My Neural Network did better than My Perceptron. However, it is not competitive with other *sklearn* modules.

Parameters

param	value
learning_rate	0.001
maximum_iterations	1001

Conclusions



The two models were parochial as demonstrated by the results. They are using Python with the *numpy* library for cleaner, faster code. A better score may be obtained using the Random Forest Algorithm, overfitting, and more perceptrons.

Most important, this project demonstrated the goal of the project: "The final project involves applying the material taught in class to a challenge problem (such as on Kaggle) or any other problem of your interest."

This project has piqued my interest in the Kaggle Competition. Reading some of remarks, it is fierce competition. Some developers are using excessive extremes to obtain 99 percent accuracy. There are claims of overfitting to the testing data, which would be a cheat [https://www.kaggle.com/c/digit-recognizer/discussion/7059]. However, I will continue to develop to develop the two failures beyond the project due date. The concepts and codes may useful in future assignments.