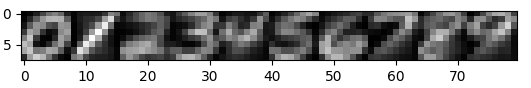
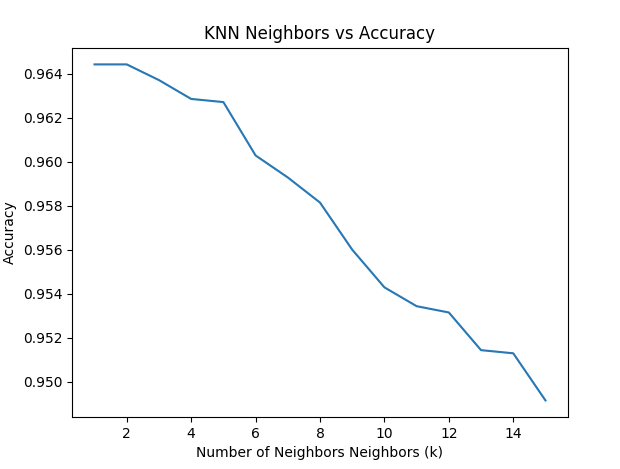
**K Nearest Neighbors**

**3.0 )**

****

**3.1 )**

**Accuracy for k=1 & k = 15**

As shown in the above graph:

The accuracy for k = 1 neighbors was among the best at 0.96875 ~ 0.97.

The accuracy for k = 15 neighbors was also quite high at 0.958 ~ 0.96.

However, after k=2, you could see from the above graph that the model began to overfit, and subsequently decreased in accuracy. However, the decrease in accuracy is negligible, at least in the context of this data set.

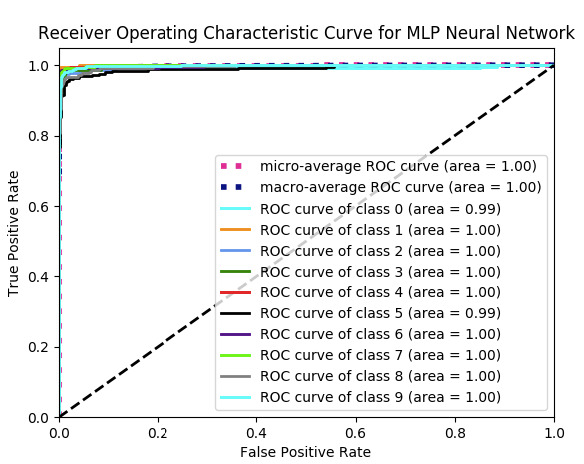
**Breaking Ties for k > 1:**

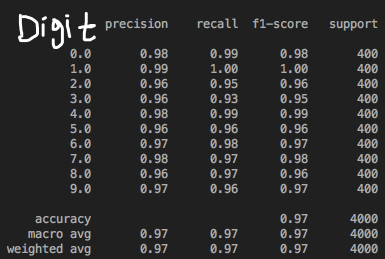
To break ties, I decided to choose the digit with that was most common in the dataset. I chose this method because it was easy to implement, and would be easy to explain to others. This goes well with a KNN classifier because one of its best attributes is ease of explanability. On a more technical side, I found that without breaking ties, my optimal k was k=3, and I had a lower average accuracy. Thus, I conclude that the strategy of choosing the most common label imporved my models performance. This could be because

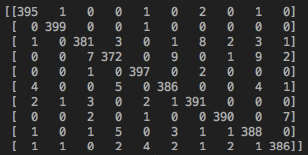
**Optimal k:**

I had two optimal k values: k =1 & k =2. The accuracy of both was: 0.9644285714285715

MLP Neural Network:

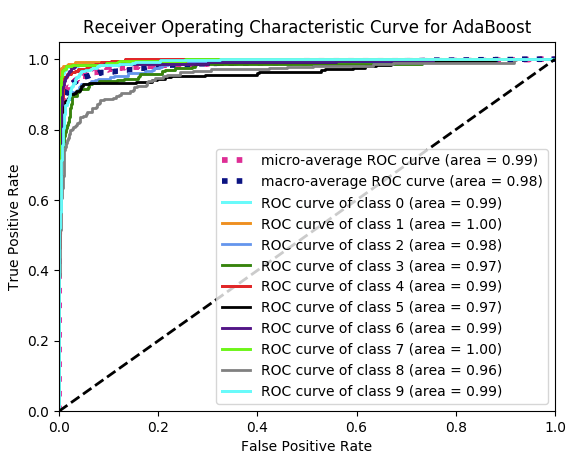


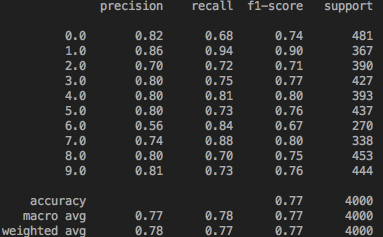




Best Parameters are: {'hidden\_layer\_sizes': (100,), 'max\_iter': 1000, 'random\_state': 3, 'solver': 'adam'}

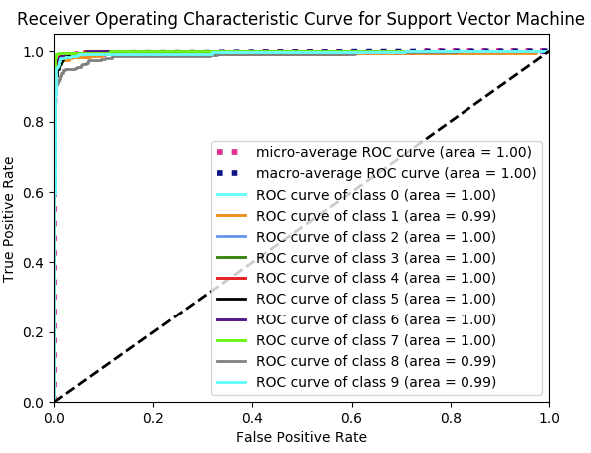
Adaboost:

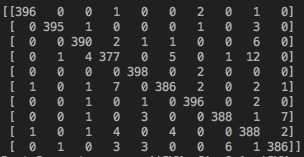


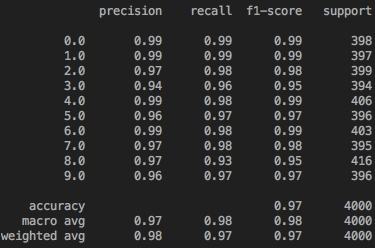


Best Parameters are: {'learning\_rate': 1, 'n\_estimators': 46, 'random\_state': 3}

Support Vector Machine:







Best Parameters are: {'SVM\_\_C': 0.1, 'SVM\_\_gamma': 0.1}