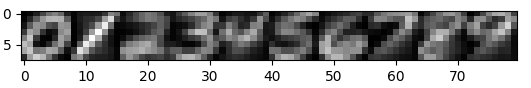
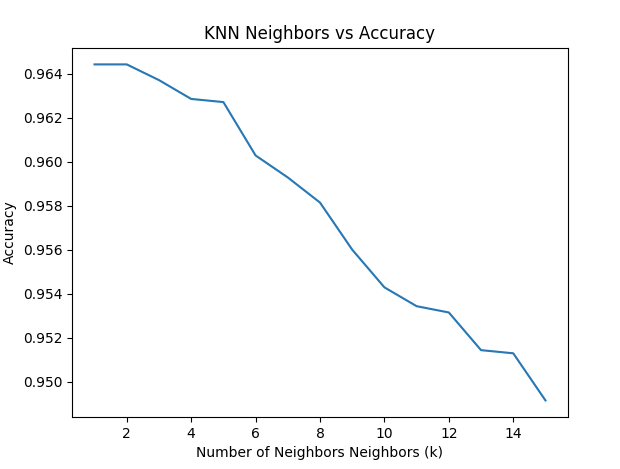
**K Nearest Neighbors**

**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.

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

The average accuracies for k=[1,15] was 0.9576380952380953.

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

**Breaking Ties for k > 1:**

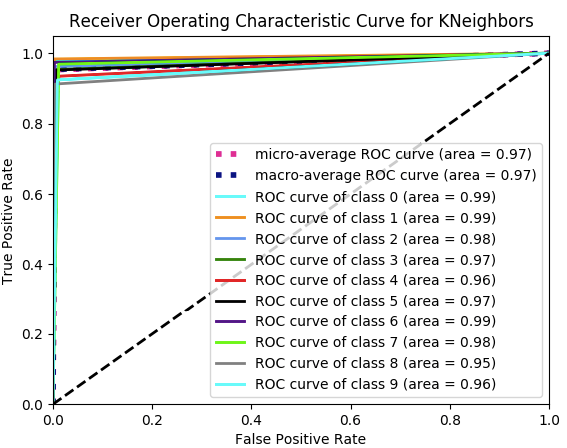
To break ties, I decided to choose the digit that was most common in the dataset. I chose this method because it was easy to implement, and would be easy to explain to others. This goes well with a KNN classifier because one of its best attributes is ease of explainability. On a more technical side, I found that without breaking ties, my optimal k was k=3, and I had a lower accuracy score of 0.94. Thus, I conclude that the strategy of choosing the most common label improved my model’s performance.

**Optimal k:**

As shown in the graph, the optimal k values were: k =1 & k=2. The accuracy of both was: 0.9644285714285715

**ROC:**

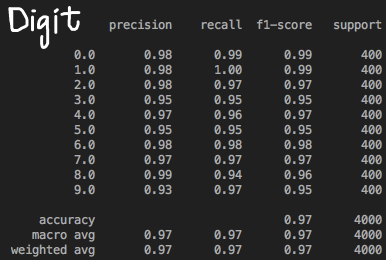
As shown in the ROC curves for KNN, the graphs below depicts that the KNN Algorithm was quite good at distinguishing between whether or not a class was a digit or not. Notice that many of the areas are close to 1, which further tells us that the prediction rate was almost perfect.



**Precision, recall, f1-score, support, accuracy:**

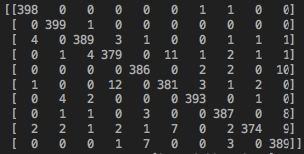
Accuracy score: 0.96875

For KNN, recall and precision are close. However, this was expected because the accuracy of KNN was fairly high at just over 0.96. Thus, precision would not have many false positives to penalize the algorithm for. Similarly, most of the digits were classified correctly, and thus I would indeed have a high recall score. F1-score is also high, which is expected, because our precision and recall scores are robust.



KNN Confusion Matrix:

Blow is a printout of my confusion matrix.

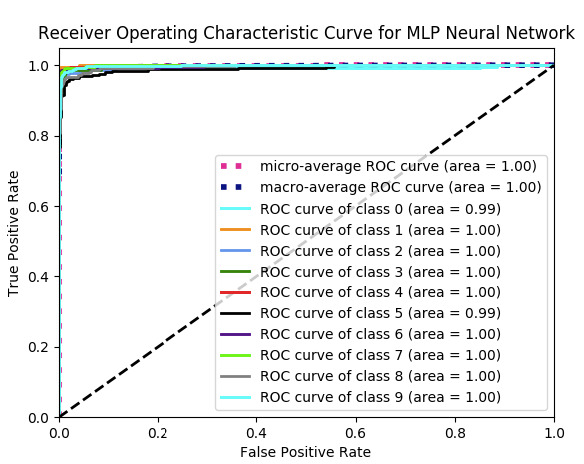


Best Parameters are: {'n\_neighbors': 1}

**MLP Neural Network**

**ROC:**

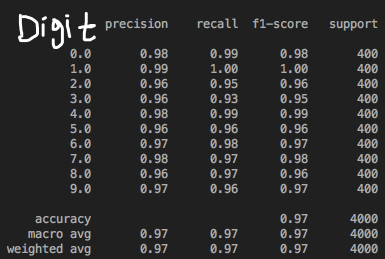
Shown in the below graph is the ROC curve for my MLP Neural Network. As we can see by the curve, our True positive rates were high, and our false positive rates were low for all digits. In fact, many of our areas for the classes are 1.0, meaning the algorithm correctly classified those classes as themselves well.

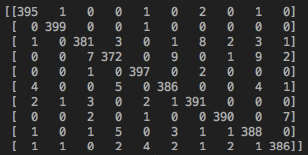


**Precision, recall, f1-score, support, accuracy:**

Accuracy score: 0.97125

For all the digit classifications, both precision and recall performed well. In fact, for digit 1.0, MLP was able to correctly identify all of the 1.0’s! However, it did manage to get some false positives, which it was penalized for in the precision score. Overall the classifier performed well. We can thank a large dataset, and cross validation for the tuning of parameters for that.



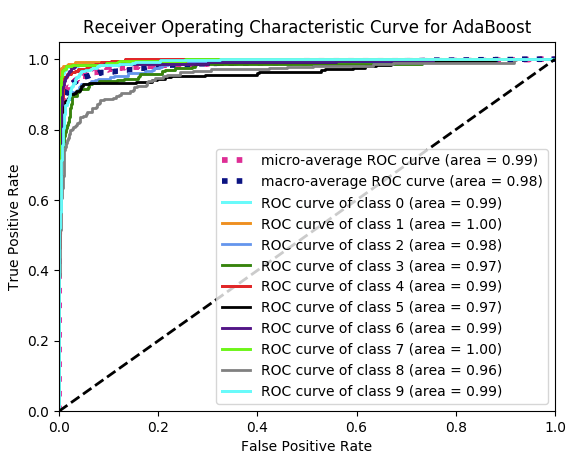


Best Parameters are: {'hidden\_layer\_sizes': (100,), 'max\_iter': 1000, 'random\_state': 3, 'solver': 'adam'}

**Adaboost**

**ROC:**

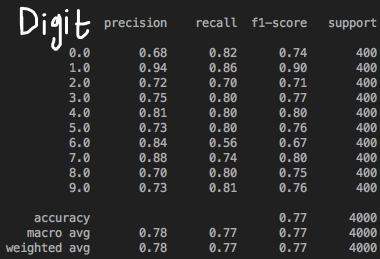
Shown in the below graph is the ROC curve for the Adaboost Classifier. As we can see by the curve, our True positive rates were high, and our false positive rates were low. Not as good as the other classifier, but still, good results.

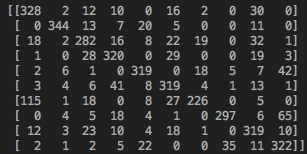


**Precision, recall, f1-score, support, accuracy:**

Accuracy score: 0.76875

This classifier classified correctly classified most digits, with the exception of digit 6.0, this led to a poor recall score for that digit. However, since it did not classify many other digits as a 6.0, when they were not, it ended up having a good precision score. The scores for this classifier are a bit all over the place, but the average accuracy score of 0.76875 tells us it is a reliable classifier in this context. Typ1 1 and Type 2 errors are not that detrimental in the context of our dataset.





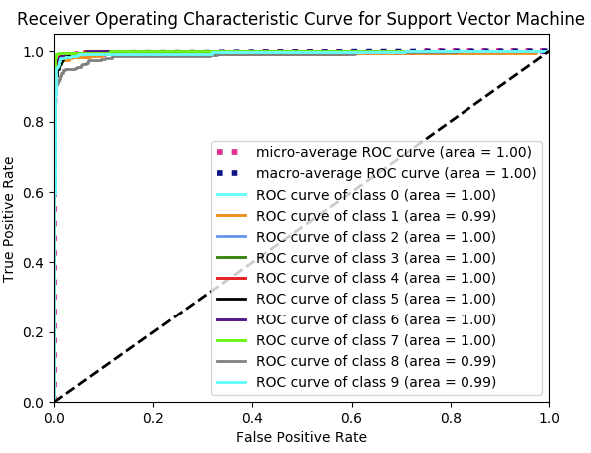
Best Parameters are: {'learning\_rate': 1, 'n\_estimators': 45, 'random\_state': 3}

**Support Vector Machine**

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

**ROC:**

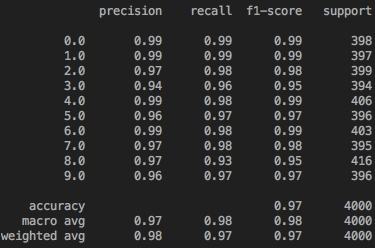
SVM performed well in this graph, it has many a majority of its area curves being 1. It had a really low false positive rate, and with its worst being digit 8.0, which was still a 0.99 area.



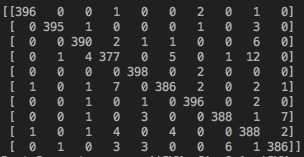
**Precision, recall, f1-score, accuracy:**

Accuracy score: 0.975

SVM was the most consistent in its variance within classification accuracies for all of these metrics. It correctly identified 99% of the digits, with its lowest precision score being for the digit 3.0.



**Confuion Matrix:**



**Which classifier performed best? Which performed worst?**

Best: KNN

Worst: AdaBoost