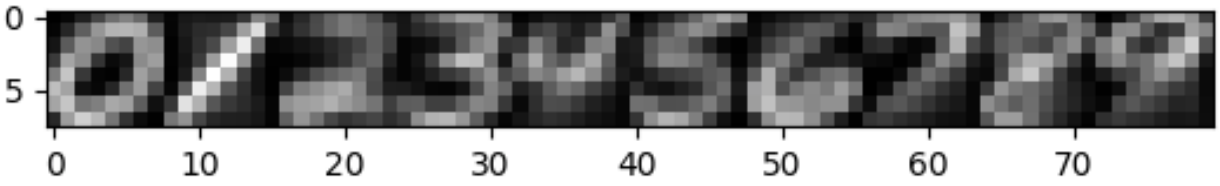
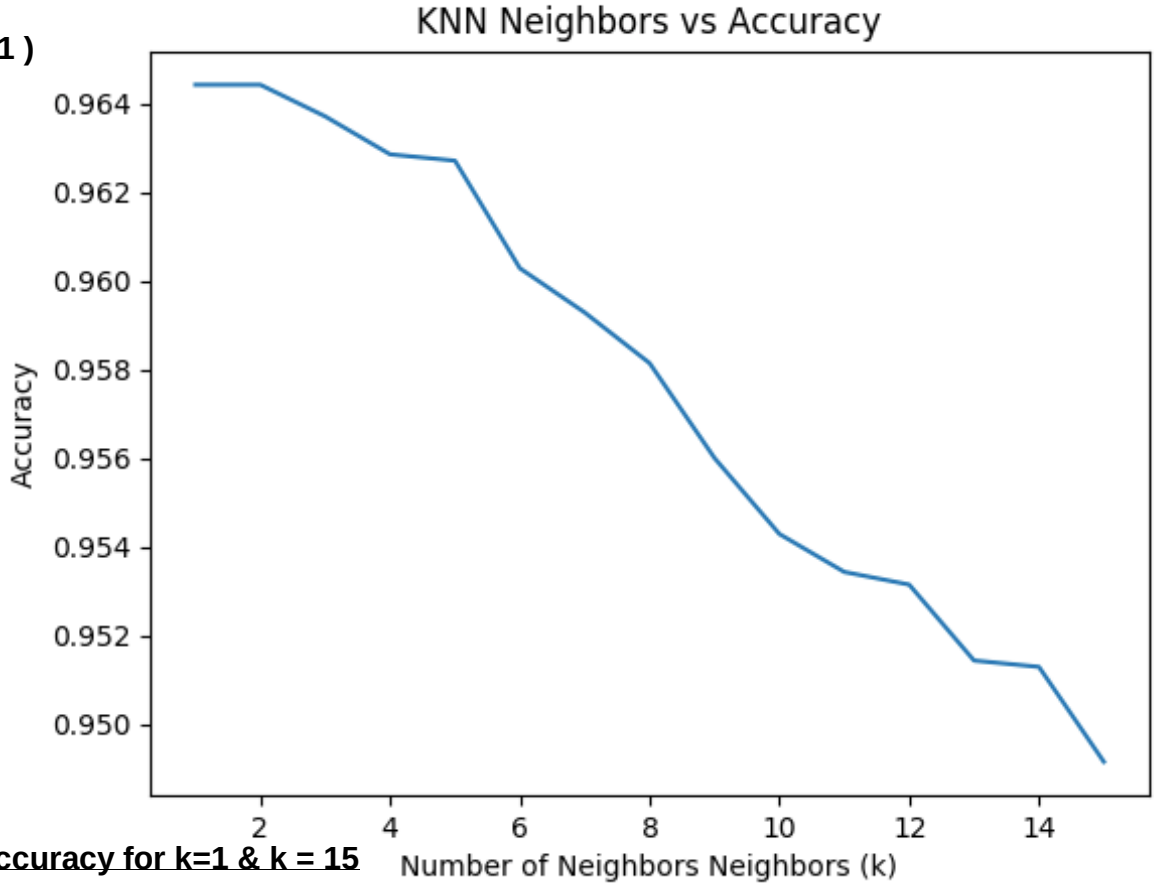
K Nearest Neighbors

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.

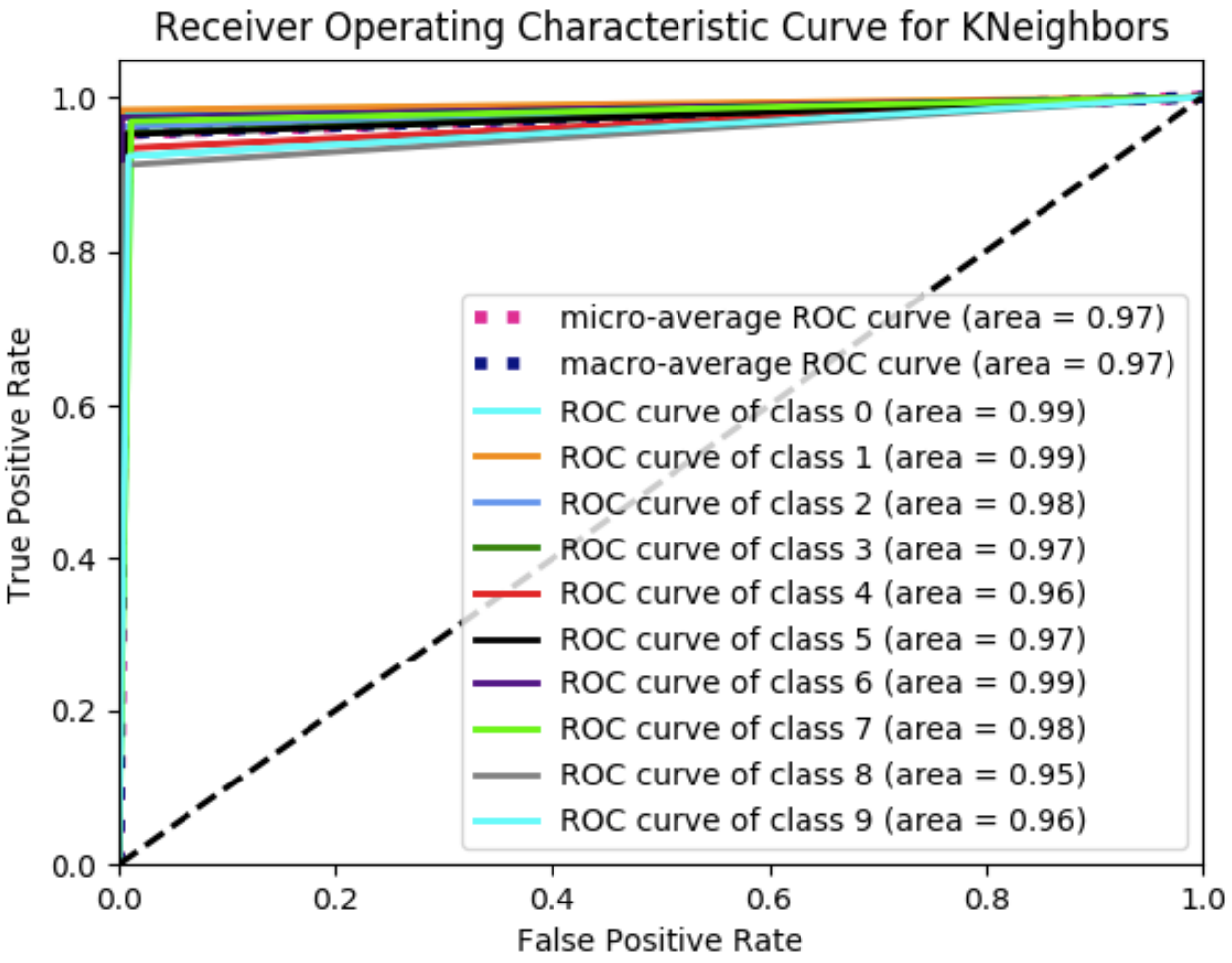
However, after k=2, you could see from the above graph that the model began to overfit, and subsequently decreased in accuracy. However, the decrease in accuracy is negligible, at least in the context of this data set.

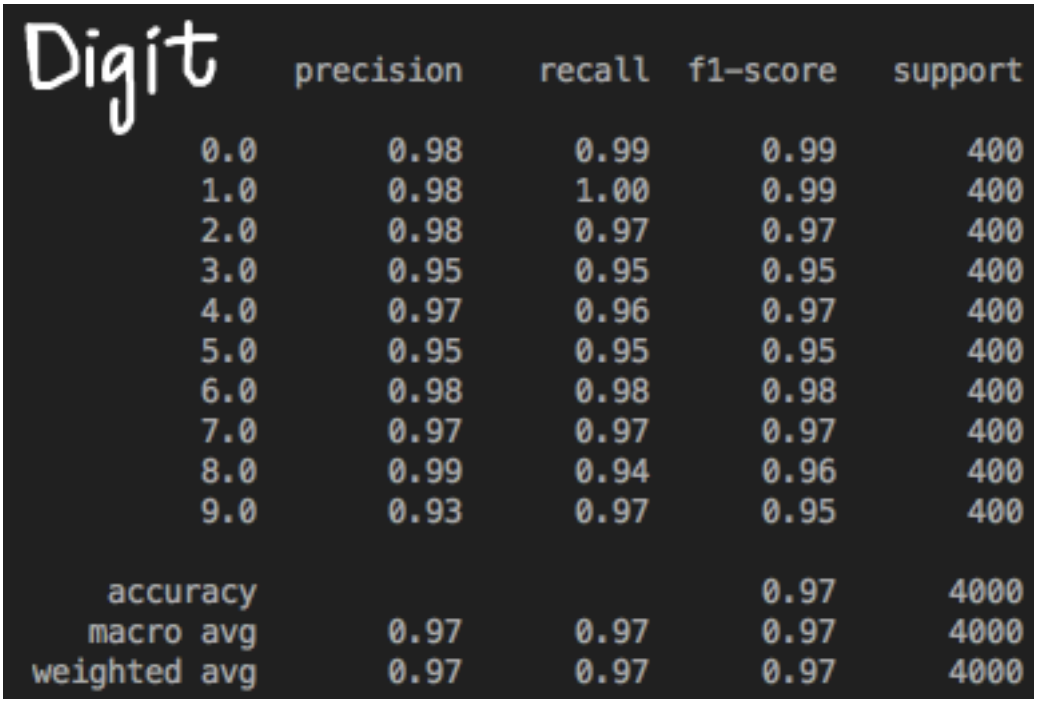
Breaking Ties for k > 1:

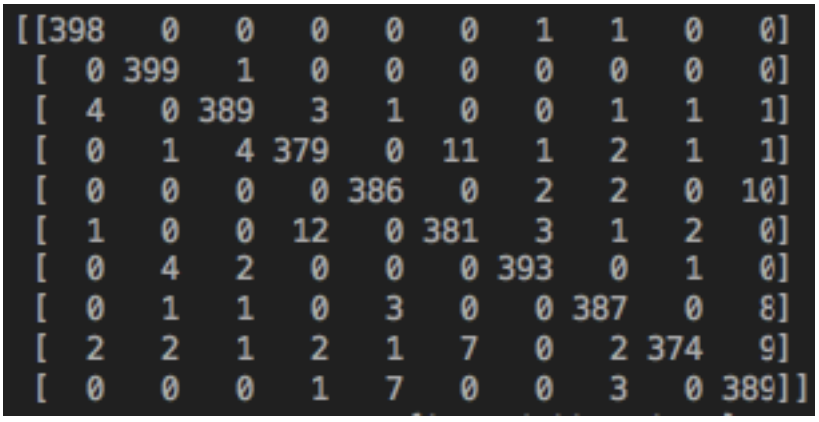
To break ties, I decided to choose the digit with that was most common in the dataset. I chose this method because it was easy to implement, and would be easy to explain to others. This goes well with a KNN classifier because one of its best attributes is ease of explanability. On a more technical side, 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. The accuracy of both was: 0.9644285714285715

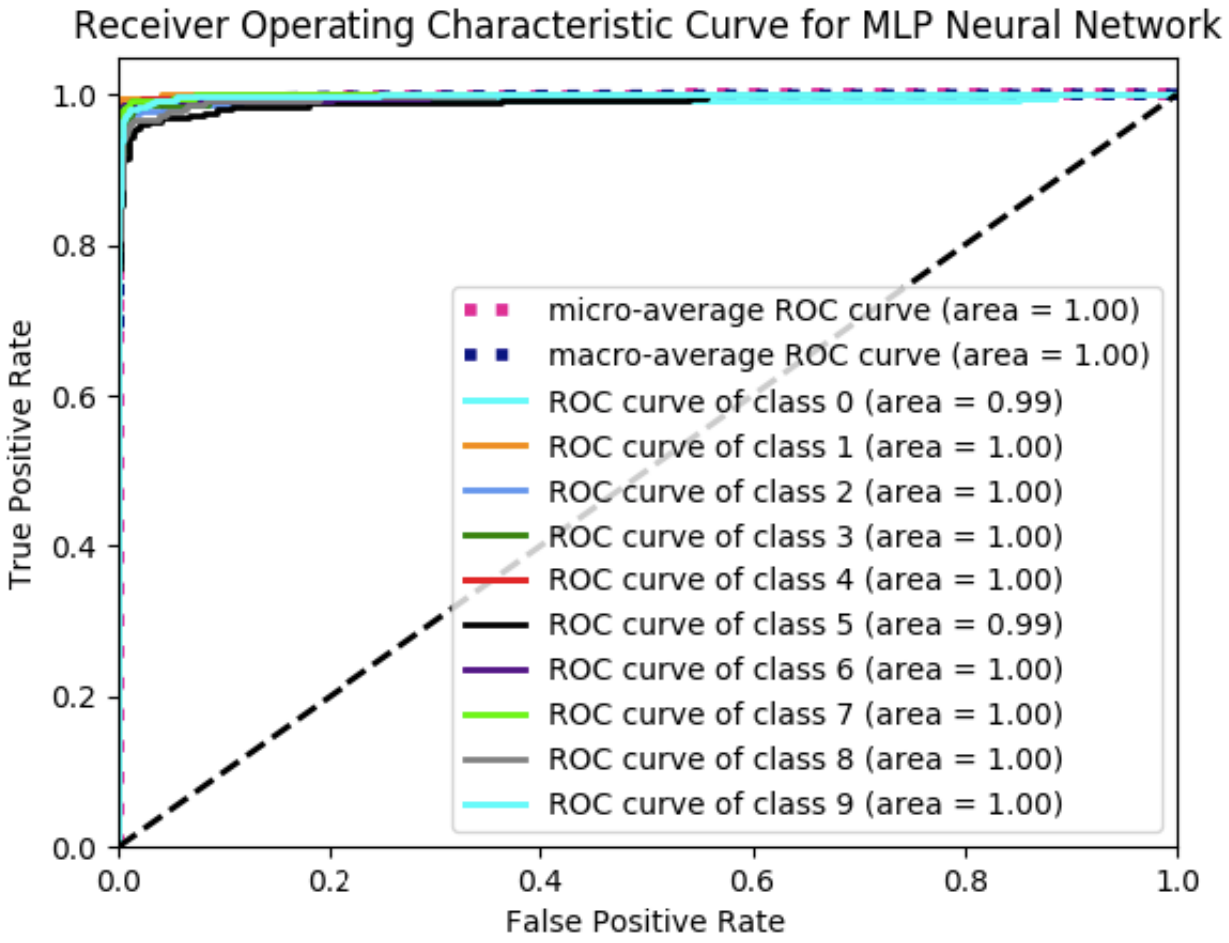


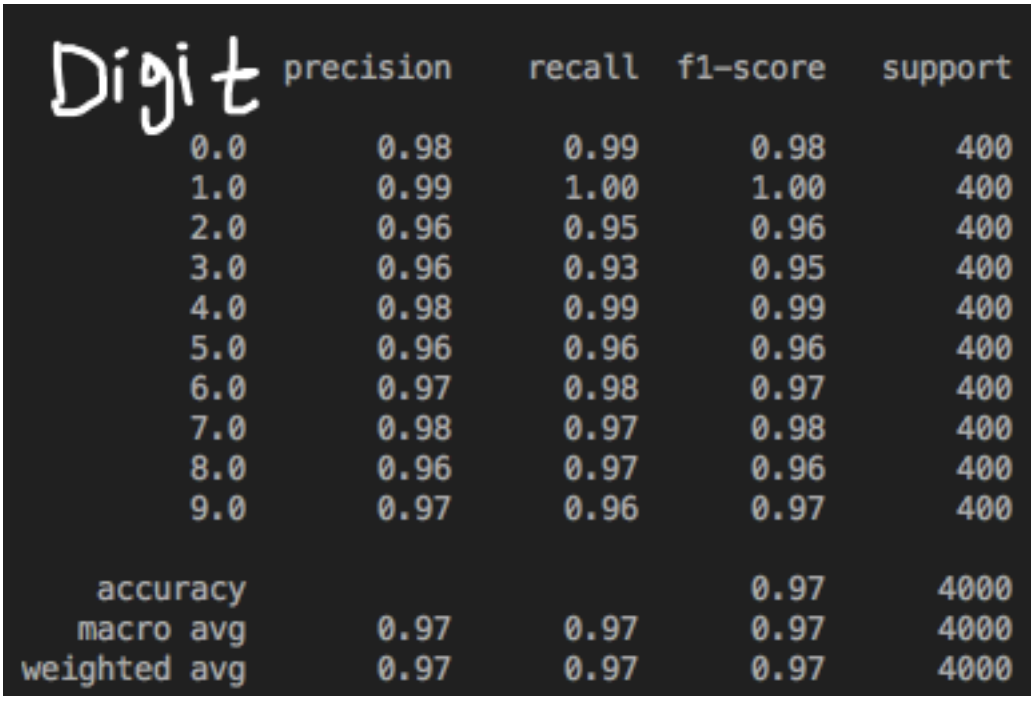


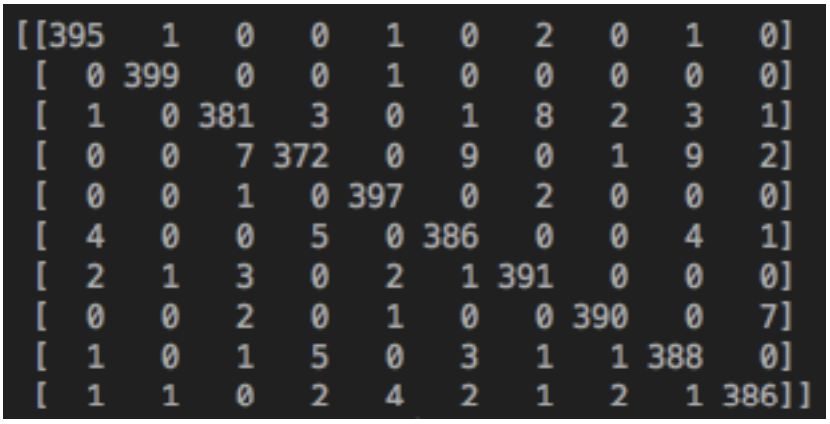


Best Parameters are: {'n_neighbors': 1}

MLP Neural Network

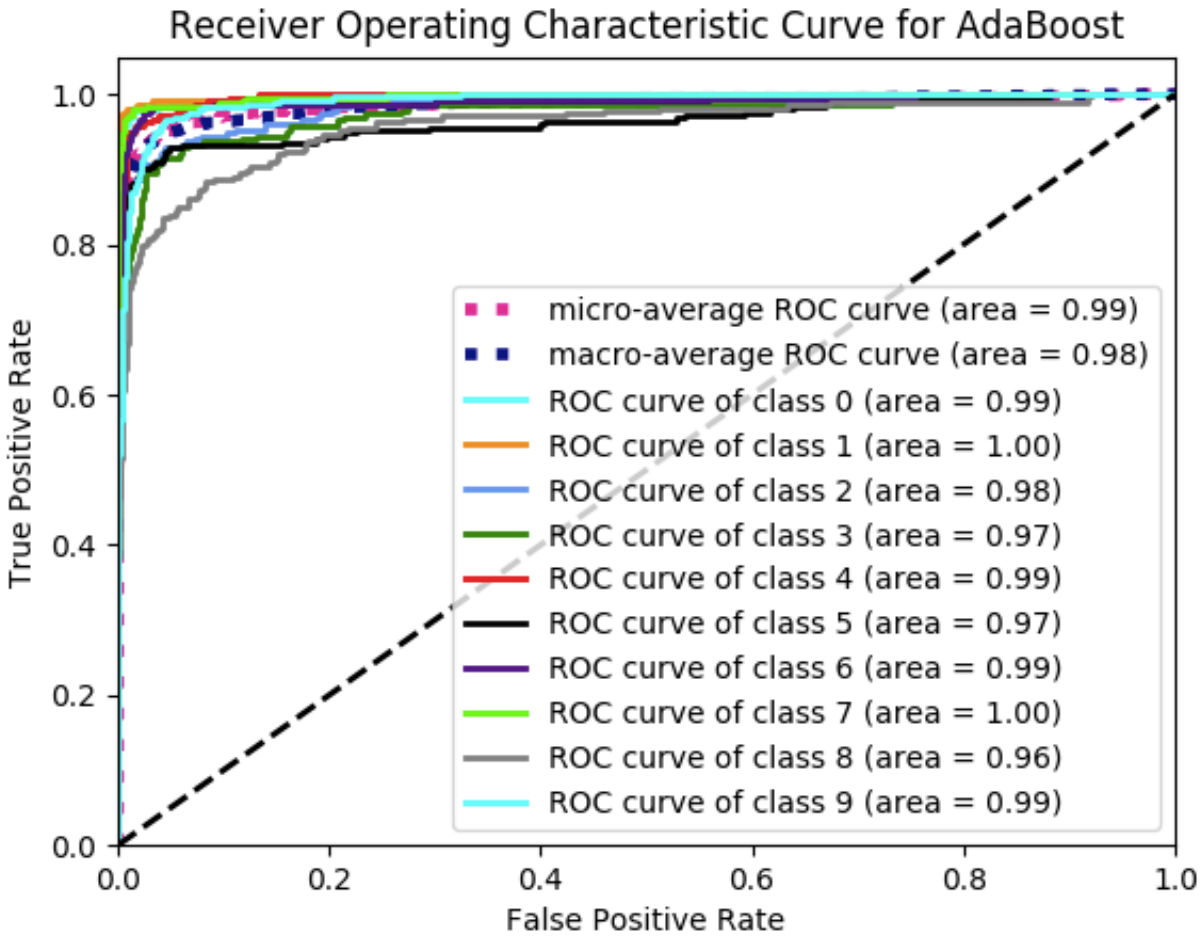


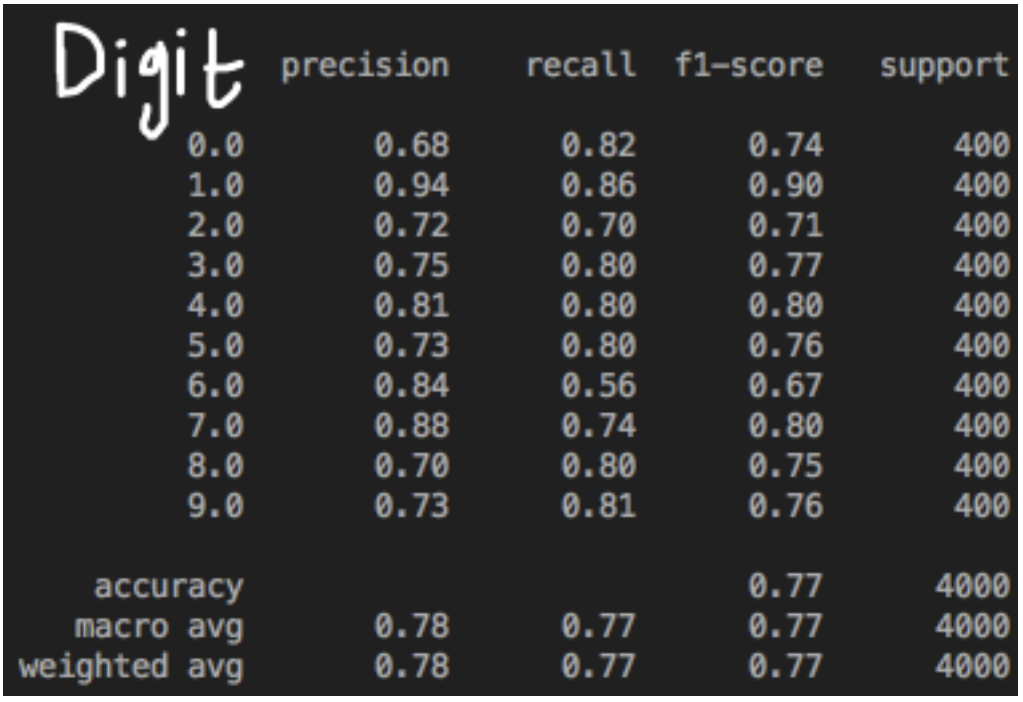




Best Parameters are: {'hidden_layer_sizes': (100,), 'max_iter': 1000, 'random_state': 3, 'solver': 'adam'}

Adaboost


Receiver Operating Characteristic Curve for AdaBoost
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
micro-average ROC curve (area = 0.99)
macro-average ROC curve (area = 0.98)
ROC curve of class 0 (area = 0.99)
ROC curve of class 1 (area = 1.00)
ROC curve of class 2 (area = 0.98)
ROC curve of class 3 (area = 0.97)
ROC curve of class 4 (area = 0.99)
ROC curve of class 5 (area = 0.97)
ROC curve of class 6 (area = 0.99)
ROC curve of class 7 (area = 1.00)
ROC curve of class 8 (area = 0.96)
ROC curve of class 9 (area = 0.99)

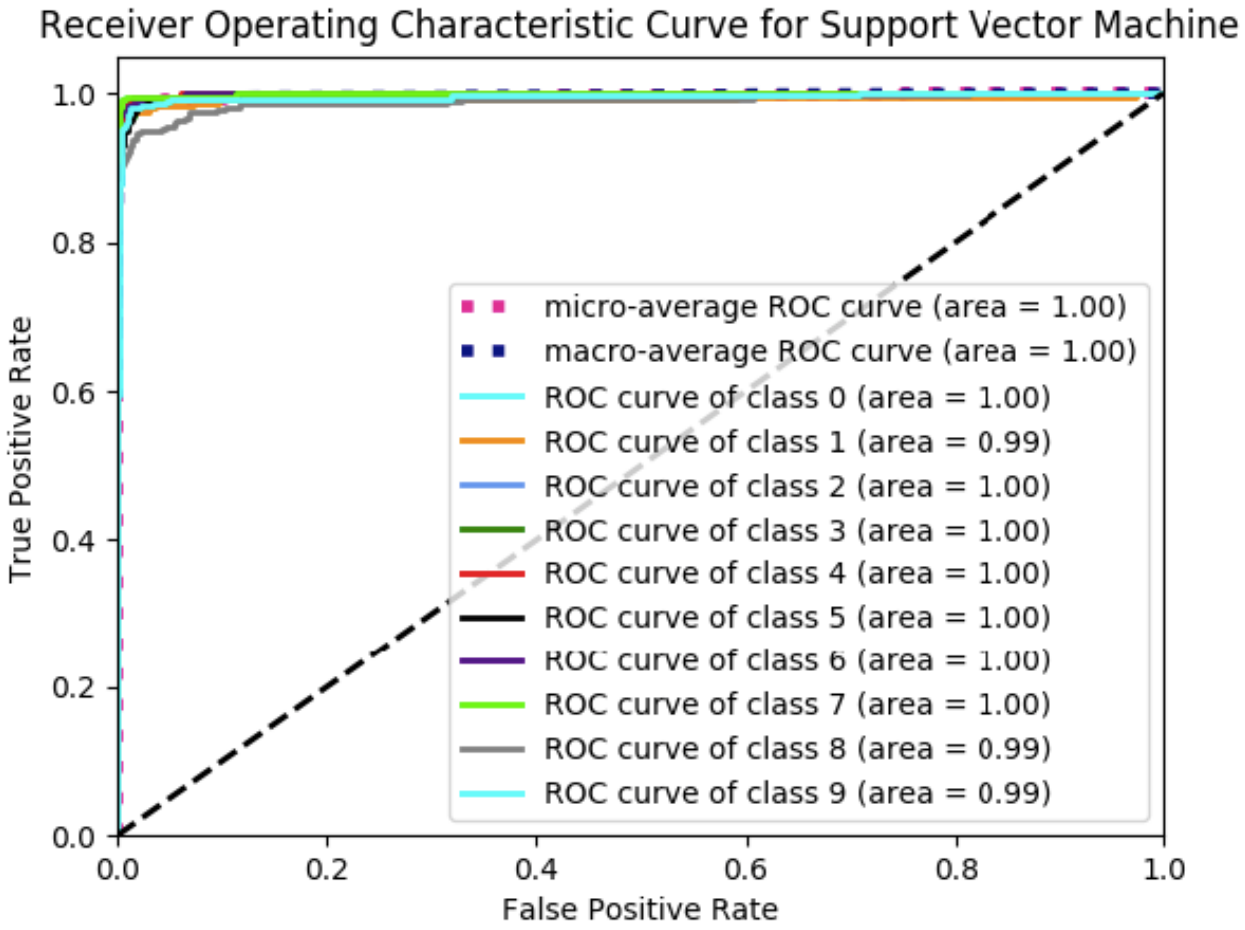


Digit	precision	recall	f1-score	support
0.0	0.68	0.82	0.74	400
1.0	0.94	0.86	0.90	400
2.0	0.72	0.70	0.71	400
3.0	0.75	0.80	0.77	400
4.0	0.81	0.80	0.80	400
5.0	0.73	0.80	0.76	400
6.0	0.84	0.56	0.67	400
7.0	0.88	0.74	0.80	400
8.0	0.70	0.80	0.75	400
9.0	0.73	0.81	0.76	400
accuracy			0.77	4000
macro avg	0.78	0.77	0.77	4000
weighted avg	0.78	0.77	0.77	4000

```
[[328   2  12  10   0  16   2   0  30   0]
 [  0 344  13   7  20   5   0   0  11   0]
 [ 18   2 282  16   8  22  19   0  32   1]
 [  1   0  28 320   0  29   0   0  19   3]
 [  2   6   1   0 319   0  18   5   7  42]
 [  3   4   6  41   8 319   4   1  13   1]
 [115   1  18   0   8  27 226   0   5   0]
 [  0   4   5  18   4   1   0 297   6  65]
 [ 12   3  23  10   4  18   1   0 319  10]
 [  2   1   2   5  22   0   0  35  11 322]]
```

Best Parameters are: {'learning_rate': 1, 'n_estimators': 45, 'random_state': 3}

Support Vector Machine




```
[[396   0   0   1   0   0   2   0   1   0]
 [  0 395   1   0   0   0   1   0   3   0]
 [  0   0 390   2   1   1   0   0   6   0]
 [  0   1   4 377   0   5   0   1  12   0]
 [  0   0   0   0 398   0   2   0   0   0]
 [  1   0   1   7   0 386   2   0   2   1]
 [  0   0   1   0   1   0 396   0   2   0]
 [  0   0   1   0   3   0   0 388   1   7]
 [  1   0   1   4   0   4   0   0 388   2]
 [  0   1   0   3   3   0   0   6   1 386]]
```

	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	398
1.0	0.99	0.99	0.99	397
2.0	0.97	0.98	0.98	399
3.0	0.94	0.96	0.95	394
4.0	0.99	0.98	0.99	406
5.0	0.96	0.97	0.97	396
6.0	0.99	0.98	0.99	403
7.0	0.97	0.98	0.98	395
8.0	0.97	0.93	0.95	416
9.0	0.96	0.97	0.97	396
accuracy			0.97	4000
macro avg	0.97	0.98	0.98	4000
weighted avg	0.98	0.97	0.97	4000

Best Parameters are: {'SVM__C': 0.1, 'SVM__gamma': 0.1}