Q1

1a

$$W^{(1)} : d \times d$$
$$z_1 1 \times d$$
$$W^{(2)} : d \times 1$$
$$z_2 \ : 1 \times d$$

1b

Assuming bias is zero, then numbers of parameters = $d \times d + d \times 1$

1c

$$\bar{y} = y - t$$
$$\bar{W}^{(2)} = (y - t)z_2$$
$$\bar{z}_2 = (y - t)W^{(2)}$$
$$\bar{h} = (y - t)W^{(2)}$$
$$\bar{z}_1 = (y - t)W^{(2)}\sigma'(z_1)$$
$$\bar{W}^{(1)} = (y - t)W^{(2)}\sigma'(z_1)x$$
$$\bar{x} = (y - t)W^{(2)}(\sigma'(z_1)W^{(1)} + 1)$$

Q2

2a

if $k = k'$

$$\frac{\partial y_k}{\partial z_{k'}} = \frac{\partial y_k}{\partial z_k} = \frac{\left(\frac{\partial}{\partial z_k} \exp(z_k)\right)\left(\sum_{k'=1}^{k} \exp(z_{k'})\right) - \left(\exp(z_k)\right)\left(\frac{\partial}{\partial z_k} \sum^{k} \exp(z_{k'})\right)}{\left(\sum_{k'=1}^{k} \exp(z_{k'})\right)^2}$$

$$= \frac{\exp(z_k)\left(\sum_{k'=1}^{k} \exp(z_{k'})\right) - \left(\exp(z_k)\right)\exp(z_k)}{\left(\sum_{k'=1}^{k} \exp(z_{k'})\right)^2}$$

$$= \frac{\exp(z_k)}{\left(\sum_{k'=1}^{k} \exp(z_{k'})\right)} - \frac{\exp(z_k)^2}{\left(\sum_{k'=1}^{k} \exp(z_{k'})\right)^2}$$

$$= y_k - y_k^2 = y_k(1 - y_k)$$

if $k \neq k'$

$$\frac{\partial y_k}{\partial z_{k'}} = \frac{0 - \exp(z_k)\exp(z_{k'})}{\left(\sum_{k'=1}^{k} \exp(z_{k'})\right)^2}$$

$$= \frac{-\exp(z_k)}{\sum_{k'=1}^{k} \exp(z_{k'})} \cdot \frac{\exp(z_{k'})}{\sum_{k'=1}^{k} \exp(z_{k'})^2}$$

$$= -y_k \cdot y_{k'}$$

$$\frac{\partial y_k}{\partial z_{k'}} \begin{cases} y_k(1 - y_k) & \text{when } k = k' \\ -y_k \cdot y_{k'} & \text{when } k \neq k' \end{cases}$$

2b

$$\frac{\partial \mathcal{L}_{CE}(t, y(x;W))}{\partial W_k} = \frac{\partial \mathcal{L}_{CE}}{\partial y_k} \cdot \frac{\partial y_k}{\partial z_k'} \cdot \frac{\partial z_k'}{\partial w_k}$$

$$\frac{\partial z_k'}{\partial W_k} = x$$

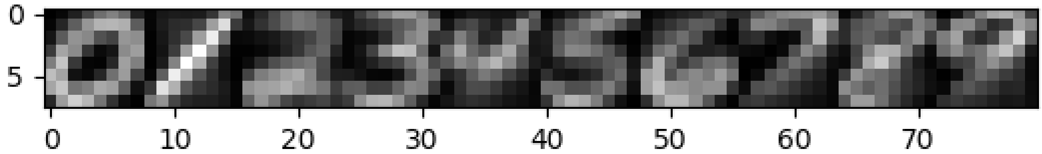
$$\frac{\partial y_k}{\partial z_k'} = y_k(1-y_k)$$

$$\frac{\partial \mathcal{L}_{CE}}{\partial y_k} = \frac{-t_k}{y_k}$$

$$\frac{\partial \mathcal{L}_{CE}(t, y(x;w))}{\partial W_k} = \frac{-t_k}{y_k} \cdot y_k(1-y_k) \cdot x$$

Q3

3.0



3.1

3.1.1

In [1]:

```
from q3_1 import KNearestNeighbor, classification_accuracy, cross_validation
import data

train_data, train_labels, test_data, test_labels = data.load_all_data('data')
```

In [2]:

```
knn = KNearestNeighbor(train_data, train_labels)

print('K = 1 Train Accuarcy: ', classification_accuracy(knn, 1, train_data, train_labels))
print('K = 1 Test Accuarcy: ', classification_accuracy(knn, 1, test_data, test_labels))

print('K = 15 Train Accuarcy: ', classification_accuracy(knn, 15, train_data, train_labels))
print('K = 15 Test Accuarcy: ', classification_accuracy(knn, 15, test_data, test_labels))
```

```
K = 1 Train Accuarcy:  1.0
K = 1 Test Accuarcy:  0.96875
K = 15 Train Accuarcy:  0.9597142857142857
K = 15 Test Accuarcy:  0.95875
```

3.1.2

When ties occur, the alogrithm increase k by 1 recursively until the ties is broke. This method is easy to implement but it increses the computional stress and change the optimal k.

3.1.3

In [4]:

```
opt_k = cross_validation(train_data, train_labels)
print('Test Accuracy: ', classification_accuracy(knn, opt_k, test_data, test_labels))
```

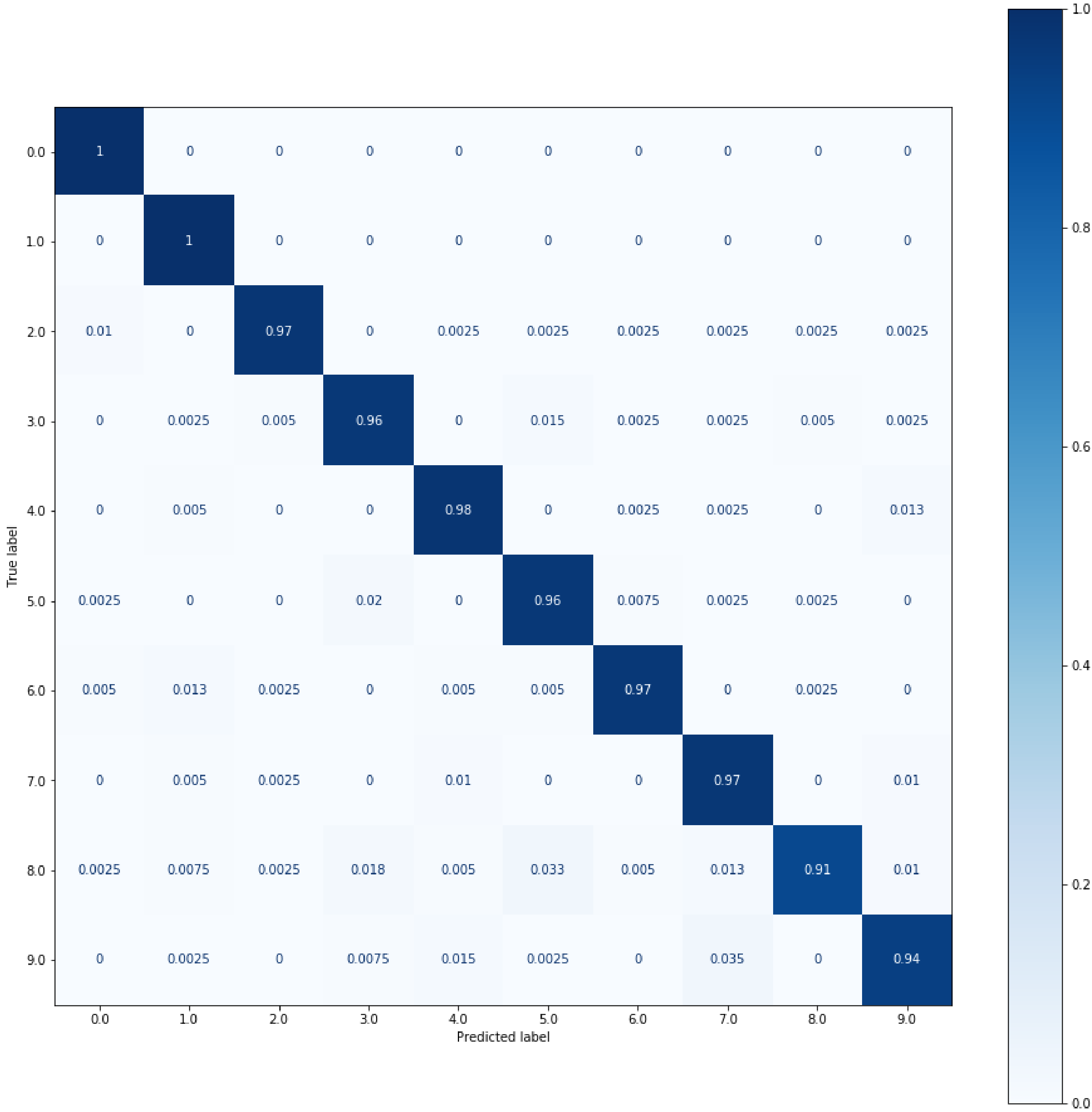
```
Optimal K:  1
Average Accuracy across folds:  0.9644285714285715
Test Accuracy:  0.96875
```

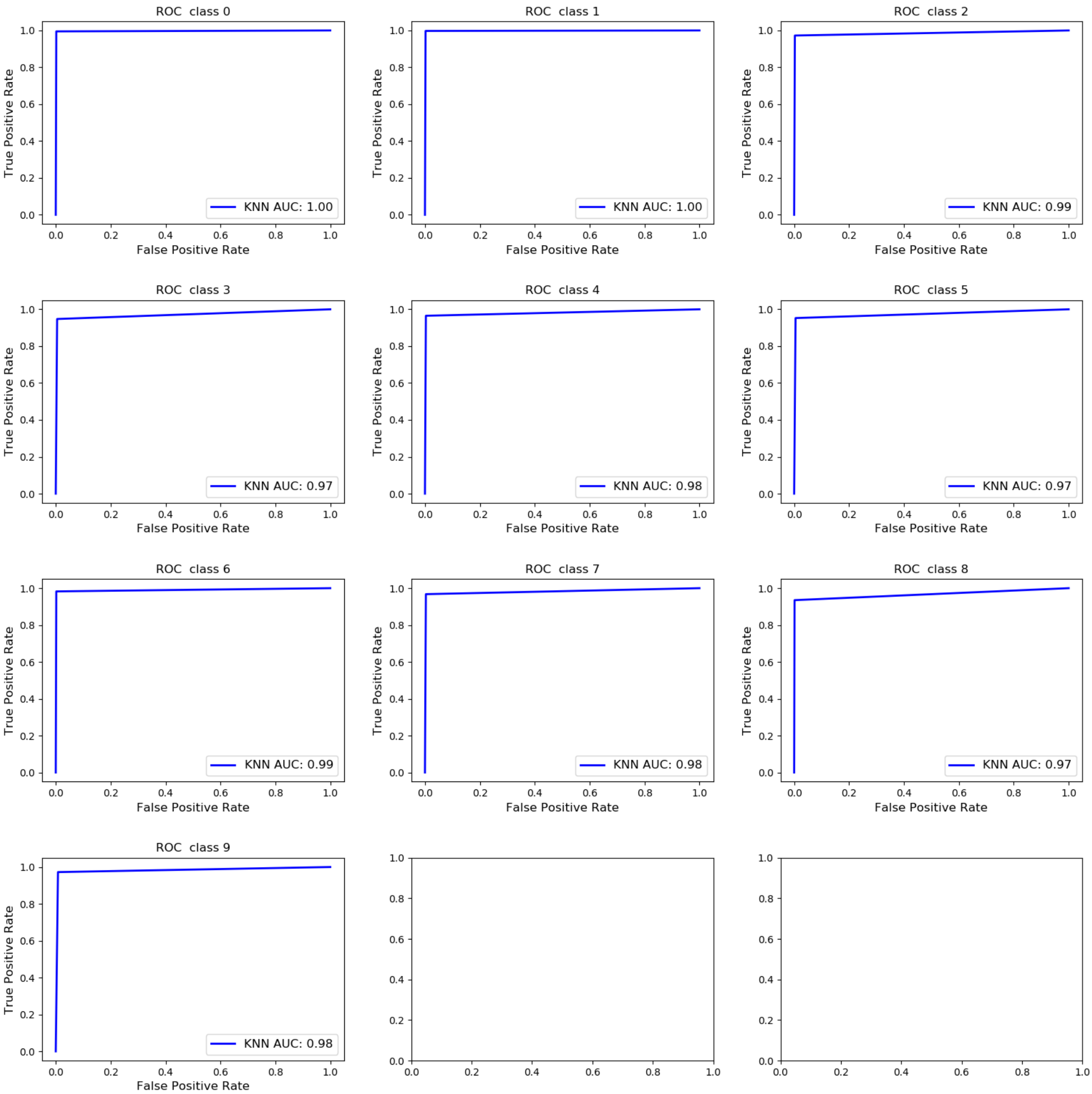
3.2

Models fitted in q3_3.py file.

3.3

KNN


1 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0.01 0 0.97 0 0.0025 0.0025 0.0025 0.0025 0.0025 0.0025
0 0.0025 0.005 0.96 0 0.015 0.0025 0.0025 0.005 0.0025
0 0.005 0 0 0.98 0 0.0025 0.0025 0 0.013
0.0025 0 0 0.02 0 0.96 0.0075 0.0025 0.0025 0
0.005 0.013 0.0025 0 0.005 0.005 0.97 0 0.0025 0
0 0.005 0.0025 0 0.01 0 0 0.97 0 0.01
0.0025 0.0075 0.0025 0.018 0.005 0.033 0.005 0.013 0.91 0.01
0 0.0025 0 0.0075 0.015 0.0025 0 0.035 0 0.94
True label
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
Predicted label
0.0
0.2
0.4
0.6
0.8
1.0



	Recall	Precision
0	1.0000	0.980392
1	1.0000	0.966184
2	0.9750	0.987342
3	0.9650	0.955446
4	0.9775	0.963054
5	0.9650	0.943765
6	0.9675	0.979747
7	0.9725	0.944175
8	0.9050	0.986376
9	0.9375	0.961538

Accuracy: 0.9665

KNN perform extremely well in this classification problem. Its True Positive rate reaches almost 1.0 with False Positive Rate is nearly 0. Each classes also have a distinguish auc, whic h tells they have precise classification on prediciting the true value.

The precision from each class are mostly higher than 0.95, which is extremely higher than other models.

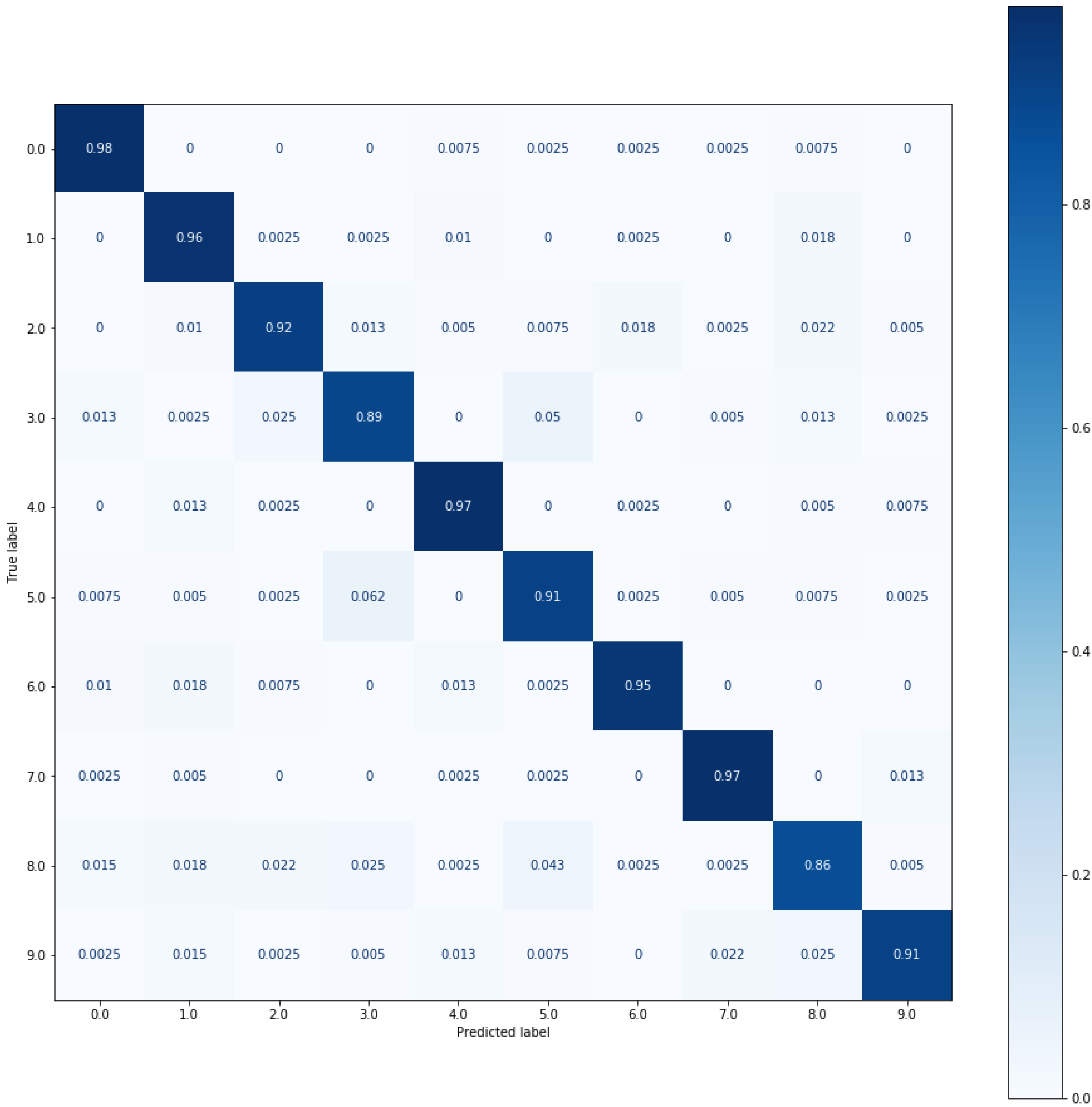
MLP

```
Normalize Confusion Matrix

![](mlp_conf.png)
![](mlp_roc.png)
```

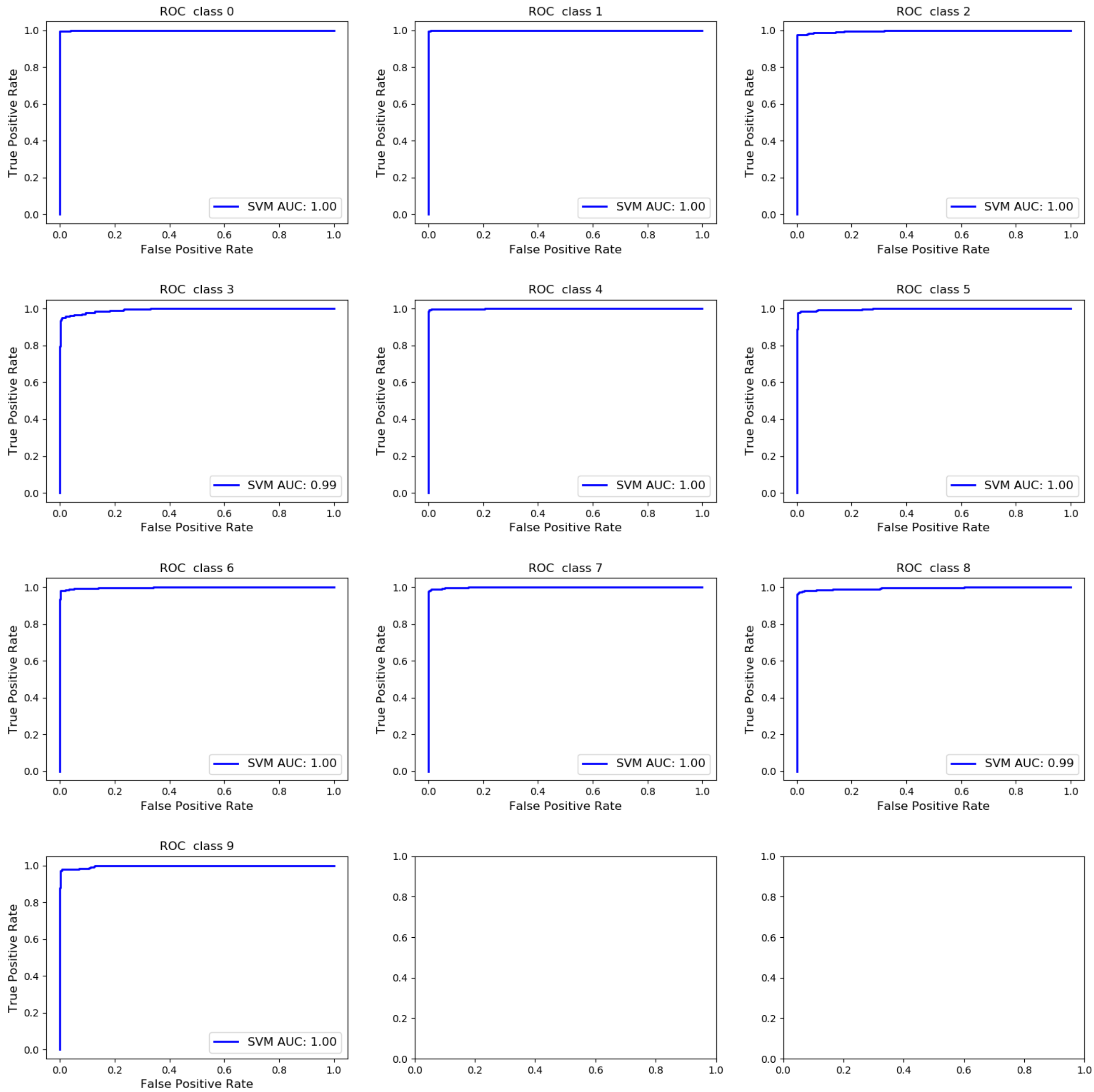
```
        Recall  Precision
0       0.9850  0.668930
1       0.9425  0.971649
2       0.9000  0.952381
3       0.8650  0.925134
4       0.9675  0.969925
5       0.8675  0.940379
6       0.9150  0.955614
7       0.9425  0.971649
8       0.8700  0.969359
9       0.9050  0.970509

Accuracy:  0.916
```

From Confusion Matrix, the predict probabilities are consist at diagonal direction, which means most of the prediction are correct. From our Accuracy and Precision, the scores are also pretty high, so this model has a relatively good performance. In roc curves for each class, True Positive Rates are in a high level with a low False Positive Rate.

SVM



True label
Predicted label
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
0.0
0.2
0.4
0.6
0.8
0.98 0 0 0 0.0075 0.0025 0.0025 0.0025 0.0075 0
0 0.96 0.0025 0.0025 0.01 0 0.0025 0 0.018 0
0 0.01 0.92 0.013 0.005 0.0075 0.018 0.0025 0.022 0.005
0.013 0.0025 0.025 0.89 0 0.05 0 0.005 0.013 0.0025
0 0.013 0.0025 0 0.97 0 0.0025 0 0.005 0.0075
0.0075 0.005 0.0025 0.062 0 0.91 0.0025 0.005 0.0075 0.0025
0.01 0.018 0.0075 0 0.013 0.0025 0.95 0 0 0
0.0025 0.005 0 0 0.0025 0.0025 0 0.97 0 0.013
0.015 0.018 0.022 0.025 0.0025 0.043 0.0025 0.0025 0.86 0.005
0.0025 0.015 0.0025 0.005 0.013 0.0075 0 0.022 0.025 0.91

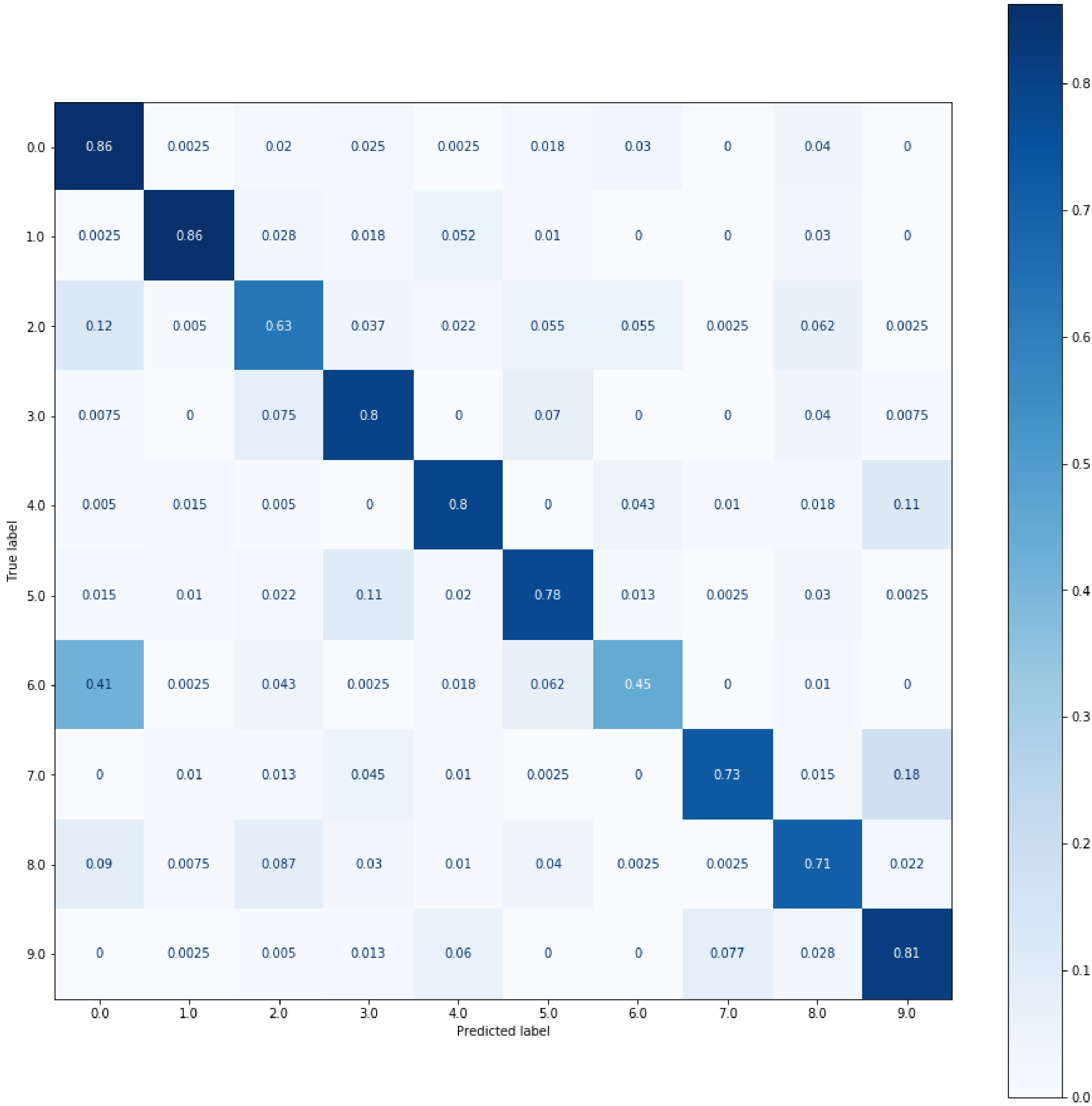


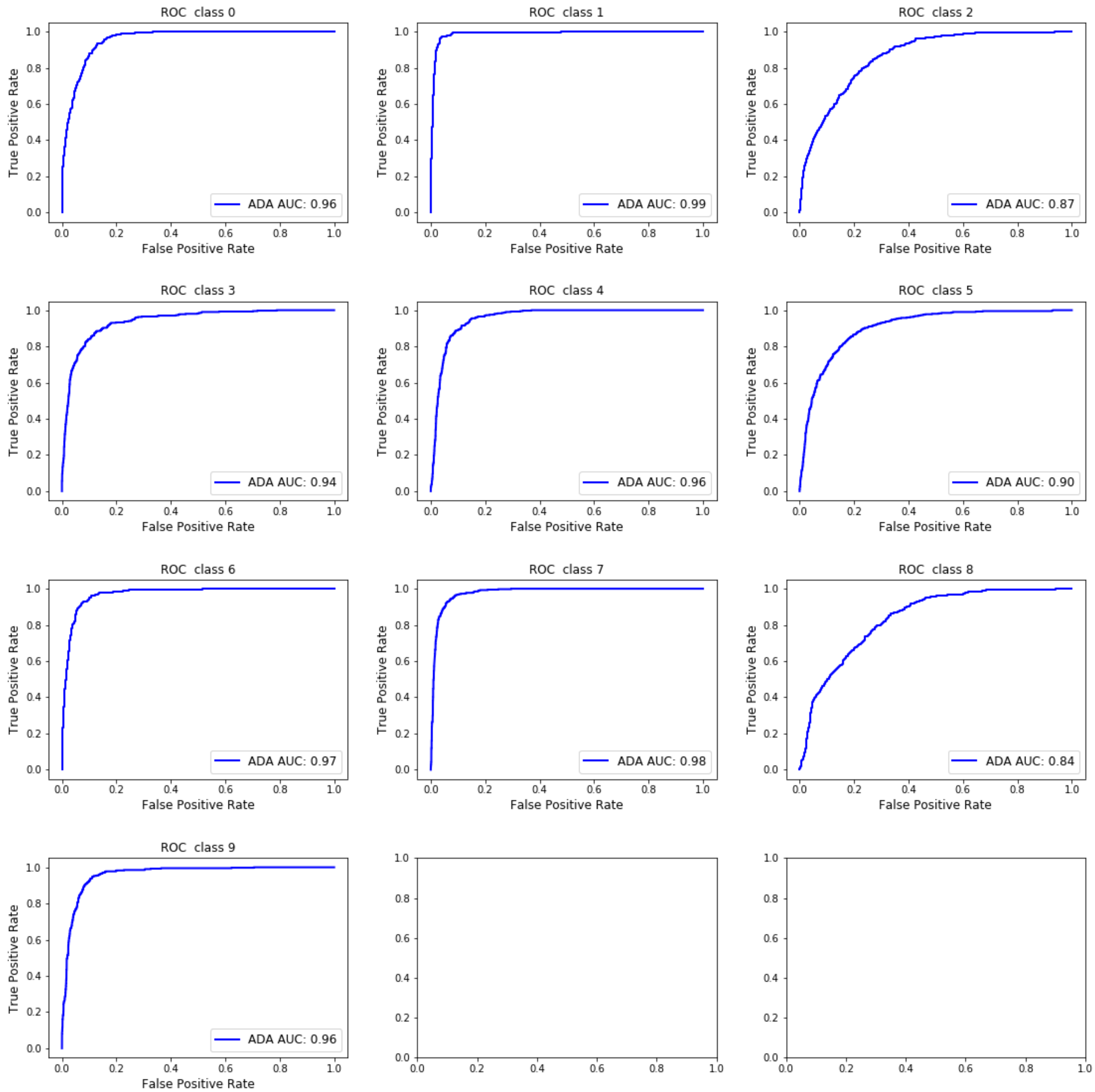
	Precision	Recall
0	0.992537	0.9975
1	0.990074	0.9975
2	0.982412	0.9775
3	0.964377	0.9475
4	0.970660	0.9925
5	0.967822	0.9775
6	0.982500	0.9825
7	0.980000	0.9800
8	0.977157	0.9625
9	0.979849	0.9725

Accuracy: 0.97875

In this case, we used Radial basis function kernel, which has slightly better performance than Linear Kernel. Because with less features and large datasets, RBF kernel is better than Linear kernel. From it ROC curve, True Positive reach to a high level with a extremely low False Positive rate, which supported by its high Accuarcy result too.

Adaboost Classifier





	Recall	Precision
0	0.8025	0.877049
1	0.8850	0.975207
2	0.6900	0.707692
3	0.7575	0.703016
4	0.8275	0.844388
5	0.7300	0.656180
6	0.7025	0.831361
7	0.7550	0.937888
8	0.7700	0.618474
9	0.8350	0.734066

Accuracy: 0.7755

From Accuracy result, the performance seems to be relatively weaker than other models. Additionally, from Roc curve, a high True Positive Rate will consist weith a relatively high False Positive Rate. In Normalize confusion matrix plot, there are few point with deep color, which means there are few false prediction on some specific points. This result is within my expection, because Adaboost Classifier is more successful in binary classification instead of MultiClassification. However, this low accuaracy result may possible cause by an appropriate weak classifier too. Moreover, in Multi-Class case, the error for weak classifier needs to be 1/k, which is hard to achieve compare to 1/2 in binary case.

In [ ]:

```

```