# CSC2515 Fall 2022: Introduction to Machine Learning Homework 3

Collaborators: Chen Dan and Hongbo Zhou

Student Name: Yulin Wang ID Number: 1003942326

2022-11-23

## Q1. Backpropagation

### (a)

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

### (b)

Since the number of parameters in this network is the number of total elements in $W^{(1)}$ and $W^{(2)}$, so it can be calculated by:

$$dim(W^{(1)}) + dim(W^{(2)}) = d^2 + d$$

### (c)

$$\begin{aligned}
&\bar{y} = \frac{\partial \mathcal{L}}{\partial y} = y - t \\
&\bar{W}^{(2)} = \bar{y}\frac{\partial y}{\partial W^{(2)}} = \bar{y}z_2^{\intercal} = (y-t)z_2^{\intercal} \\
&\bar{z}_2 = \bar{y}\frac{\partial y}{\partial z_2} = W^{(2)\intercal}\bar{y} = W^{(2)\intercal}(y-t) \\
&\bar{h} = \bar{z}_2\frac{\partial z_2}{\partial h} = \bar{z}_2 = W^{(2)\intercal}(y-t) \\
&\bar{z}_1 = \bar{h}\frac{\partial h}{\partial z_1} = \bar{h} \circ \sigma'(z_1) = W^{(2)\intercal}(y-t) \circ \sigma'(z_1) \quad \text{, where } \circ \text{ means element-wise product} \\
&\bar{W}^{(1)} = \bar{z}_1\frac{\partial z_1}{\partial W^{(1)}} = \bar{z}_1 x^{\intercal} = \bar{h} \circ \sigma'(z_1)x^{\intercal} = W^{(2)\intercal}(y-t) \circ \sigma'(z_1)x^{\intercal} \\
&\bar{x} = \bar{z}_1\frac{\partial z_1}{\partial x} + \bar{z}_2\frac{\partial z_2}{\partial x} = W^{(1)}\bar{z}_1 + \bar{z}_2 = W^{(1)}W^{(2)\intercal}(y-t) \circ \sigma'(z_1) + W^{(2)\intercal}(y-t)
\end{aligned}$$

## Q2. Multi-Class Logistic Regression

### (a)

$$
\begin{aligned}
\frac{\partial y_k}{\partial z_{k'}} &= \frac{\partial}{\partial z_{k'}}\{\frac{e^{z_k}}{\sum_{k'=1}^{K} e^{z_{k'}}}\} \\
&= \frac{\frac{\partial e^{z_k}}{\partial z_{k'}} \cdot \sum_{k'=1}^{K} e^{z_{k'}} - \frac{\partial \sum_{k'=1}^{K} e^{z_{k'}}}{\partial z_{k'}} \cdot e^{z_k}}{\{\sum_{k'=1}^{K} e^{z_{k'}}\}^2}
\end{aligned}
$$

Case 1: When $k = k'$, we have:

$$
\frac{\partial e^{z_k}}{\partial z_{k'}} = \frac{\partial e^{z_k}}{\partial z_k} = e^{z_k}; \frac{\partial \sum_{k'=1}^{K} e^{z_{k'}}}{\partial z_{k'}} = \frac{\partial \sum_{k'=1}^{K} e^{z_{k'}}}{\partial z_k} = \frac{\partial e^{z_k}}{\partial z_k} = e^{z_k}
$$

Thus,

$$
\begin{aligned}
\frac{\partial y_k}{\partial z_{k'}} &= \frac{e^{z_k} \cdot \sum_{k'=1}^{K} e^{z_{k'}} - e^{z_k} \cdot e^{z_k}}{\{\sum_{k'=1}^{K} e^{z_{k'}}\}^2} \\
&= \frac{e^{z_k} \cdot \{\sum_{k'=1}^{K} e^{z_{k'}} - e^{z_k}\}}{\{\sum_{k'=1}^{K} e^{z_{k'}}\}^2} \\
&= \frac{e^{z_k}}{\sum_{k'=1}^{K} e^{z_{k'}}} \cdot \frac{\sum_{k'=1}^{K} e^{z_{k'}} - e^{z_k}}{\sum_{k'=1}^{K} e^{z_{k'}}} \\
&= y_k \cdot (1 - \frac{e^{z_k}}{\sum_{k'=1}^{K} e^{z_{k'}}}) \\
&= y_k \cdot (1 - y_k) \\
&= y_k - y_k^2
\end{aligned}
$$

Case 2: When $k \neq k'$, we have:

$$
\frac{\partial e^{z_k}}{\partial z_{k'}} = 0; \frac{\partial \sum_{k'=1}^{K} e^{z_{k'}}}{\partial z_{k'}} = \frac{\partial e^{z_{k'}}}{\partial z_{k'}} = e^{z_{k'}}
$$

Thus,

$$
\begin{aligned}
\frac{\partial y_k}{\partial z_{k'}} &= \frac{0 - e^{z_{k'}} \cdot e^{z_k}}{\{\sum_{k'=1}^{K} e^{z_{k'}}\}^2} \\
&= -\frac{e^{z_{k'}}}{\sum_{k'=1}^{K} e^{z_{k'}}} \cdot \frac{e^{z_k}}{\sum_{k'=1}^{K} e^{z_{k'}}} \\
&= -y_{k'} y_k
\end{aligned}
$$

## (b)

Since

$$\begin{aligned}\frac{\partial \mathcal{L}_{CE}(t, y(x;W))}{\partial w_k} &= \frac{\partial - \sum_{i=1}^{K} t_i log y_i}{\partial w_k} \\ &= \sum_{i=1}^{K}\{\frac{\partial - t_i log y_i}{\partial y_i} \cdot \frac{\partial y_i}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_k}\}\end{aligned}$$

and we have

$$\frac{\partial - t_i log y_i}{\partial y_i} = -\frac{t_i}{y_i} \quad ; \; i = 1, ..., K$$

$$\frac{\partial y_i}{\partial z_k} = y_k - y_k^2 \quad \text{for } i = k; \quad \frac{\partial y_i}{\partial z_k} = -y_i y_k \quad \text{for } i \neq k \qquad \text{,from part (b)}$$

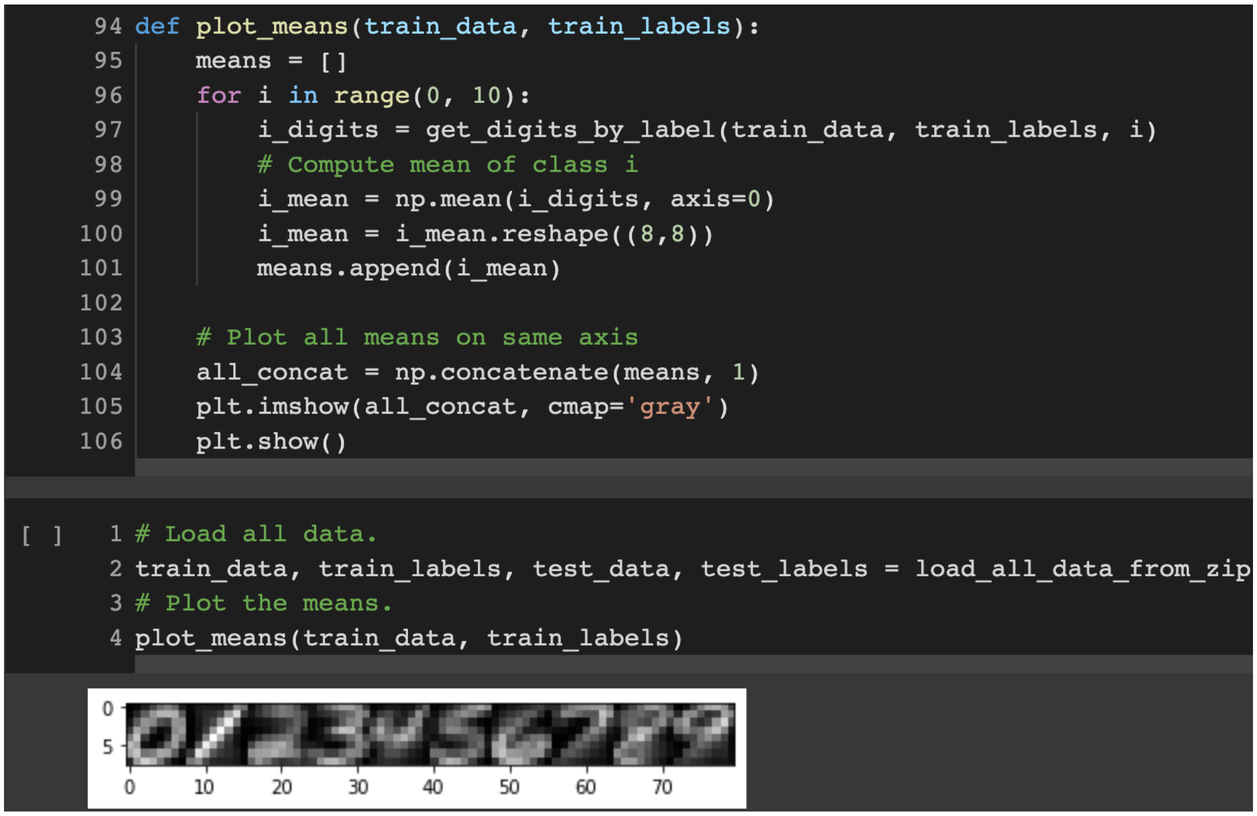
$$\frac{\partial z_k}{\partial w_k} = x$$

Thus,

$$\begin{aligned}\frac{\partial \mathcal{L}_{CE}(t, y(x;W))}{\partial w_k} &= \sum_{i=1}^{K}\{\frac{\partial - t_i log y_i}{\partial y_i} \cdot \frac{\partial y_i}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_k}\} \\ &= \sum_{i\neq k}^{K}\{-\frac{t_i}{y_i} \cdot -y_i y_k \cdot x\} + \{-\frac{t_k}{y_k} \cdot (y_k - y_k^2) \cdot x\} \\ &= \sum_{i\neq k}^{K} t_i y_k x + (-t_k x + t_k y_k x) \\ &= y_k x \sum_{i\neq k}^{K} t_i + t_k y_k x - t_k x \\ &\text{; since t is a one-hot encoding of the output, so } \sum_{i=1}^{K} t_i = 1 \text{ and} \sum_{i\neq k}^{K} t_i = 1 - t_k \\ &= y_k x(1 - t_k) + t_k y_k x - t_k x \\ &= y_k x - t_k y_k x + t_k y_k x - t_k x \\ &= y_k x - t_k x \\ &= (y_k - t_k)x\end{aligned}$$

## Q4. Handwritten Digit Classification

### 4.0



```python
def plot_means(train_data, train_labels):
    means = []
    for i in range(0, 10):
        i_digits = get_digits_by_label(train_data, train_labels, i)
        # Compute mean of class i
        i_mean = np.mean(i_digits, axis=0)
        i_mean = i_mean.reshape((8,8))
        means.append(i_mean)

    # Plot all means on same axis
    all_concat = np.concatenate(means, 1)
    plt.imshow(all_concat, cmap='gray')
    plt.show()
```

```python
[ ]
# Load all data.
train_data, train_labels, test_data, test_labels = load_all_data_from_zip
# Plot the means.
plot_means(train_data, train_labels)
```

### 4.1.1

```
For K = 1, the train classification accuracy is: 1.0.
For K = 1, the test classification accuracy is: 0.96875.
For K = 15, the train classification accuracy is: 0.9594285714285714.
For K = 15, the test classification accuracy is: 0.9585.
```

### 4.1.2

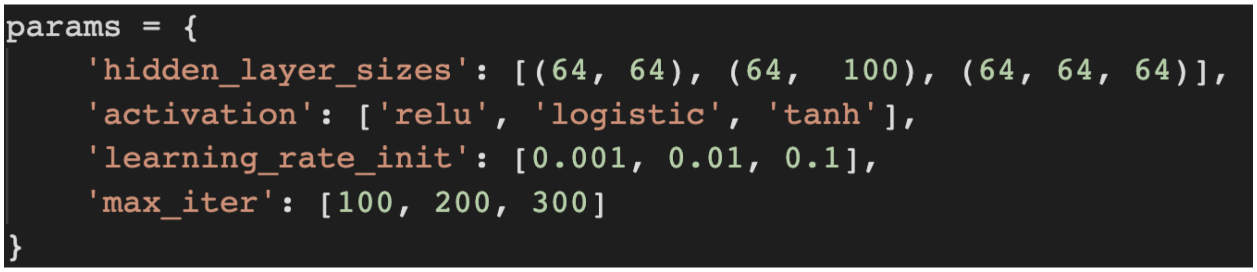
We can choose to weight tied K-NN by distance. Typically we can weight the neighbors so that the nearest points to the unobserved point have a greater weight, and then compare the ties again.

### 4.1.3

```
The optimal K is: 3.
The validation accuracy is: 0.9634285714285713.
For the optimal K = 3, the train classification accuracy is: 0.9834285714285714.
For the optimal K = 3, the test classification accuracy is: 0.96975.
```

### 4.2.1 MLP - Neural Network Classifier

Taking overfitting into consideration, we used the GridSearchCV from sklearn.model_selection to find the optimal parameters.
Here are the potential parameters:

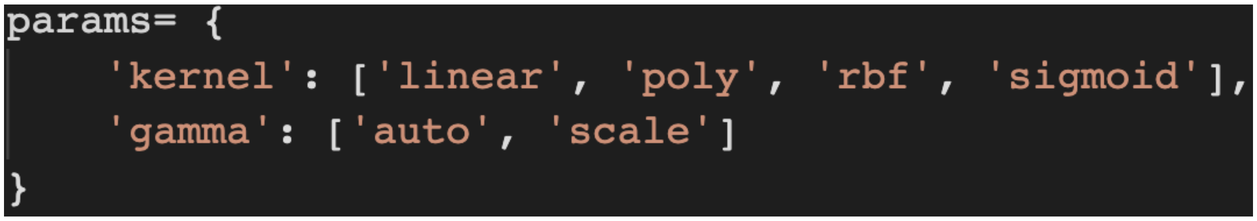


```
params = {
    'hidden_layer_sizes': [(64, 64), (64,  100), (64, 64, 64)],
    'activation': ['relu', 'logistic', 'tanh'],
    'learning_rate_init': [0.001, 0.01, 0.1],
    'max_iter': [100, 200, 300]
}
```

The optimal set of hyper-parameters for MLP-NN Classifier is:
‘activation’: ‘relu’, ‘hidden_layer_sizes’: (64, 64, 64), ‘learning_rate_init’: 0.01, ‘max_iter’:100

### 4.2.2 SVM classifier

Similarly, we used the GridSearchCV from sklearn.model_selection to find the optimal parameters.
Here are the potential parameters:

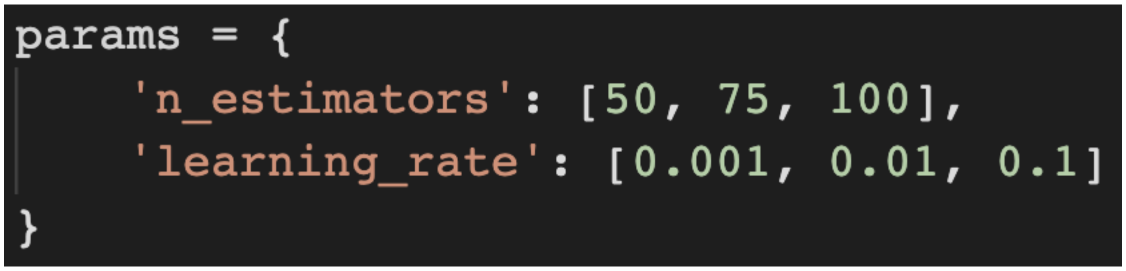


```
params= {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['auto', 'scale']
}
```

The optimal set of hyper-parameters for SVM Classifier is:
‘gamma’: ‘scale’, ‘kernel’: ‘rbf’

### 4.2.3 AdaBoost Classifier

Similarly, we used the GridSearchCV from sklearn.model_selection to find the optimal parameters.
Here are the potential parameters:

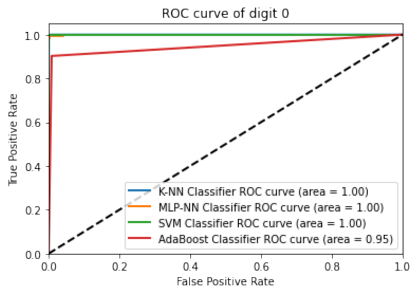


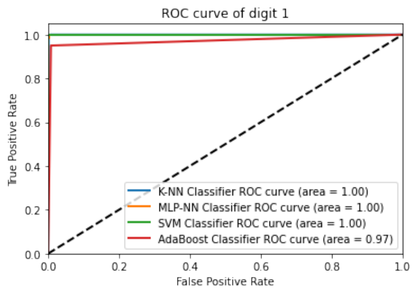
```
params = {
    'n_estimators': [50, 75, 100],
    'learning_rate': [0.001, 0.01, 0.1]
}
```

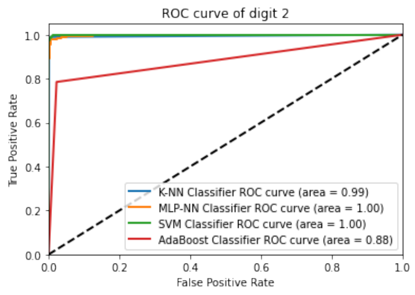
The optimal set of hyper-parameters for AdaBoost Classifier is:
‘learning_rate’: 0.001, ‘n_estimators’: 100

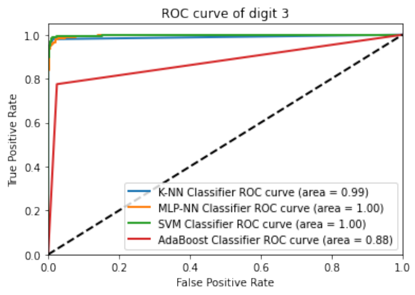
## 4.3 Model Comparison

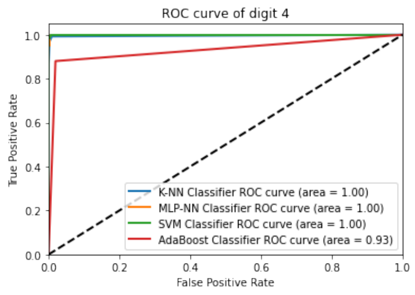
### 4.3.1 ROC (Receiver Operating Characteristics) curve

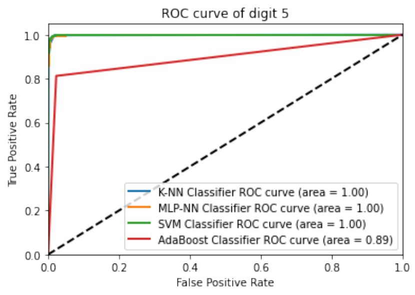


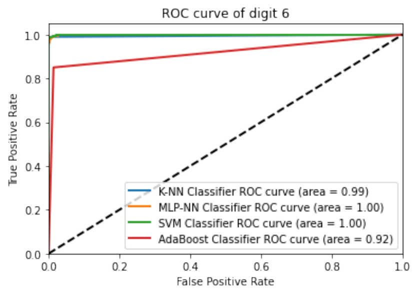


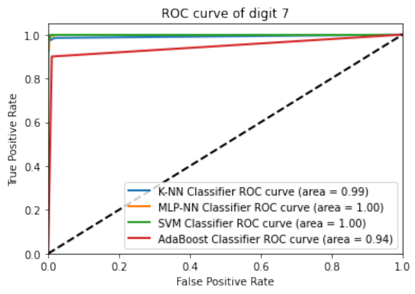


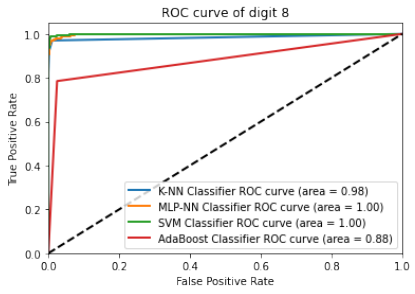


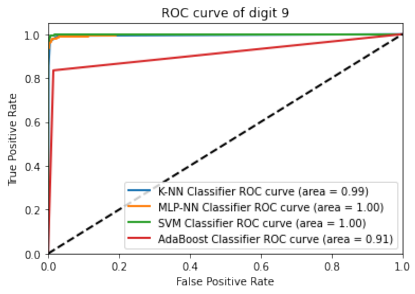




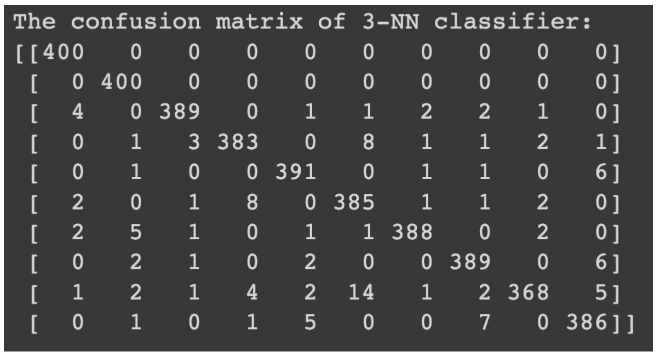




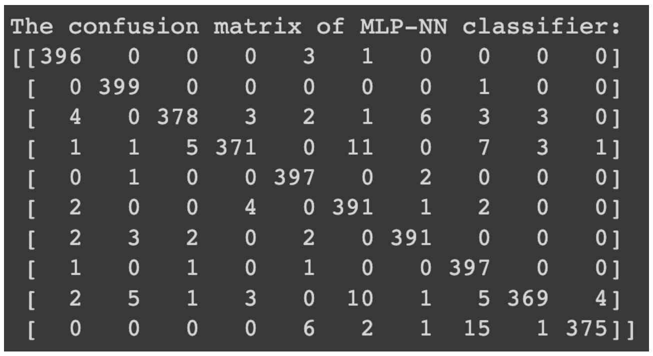




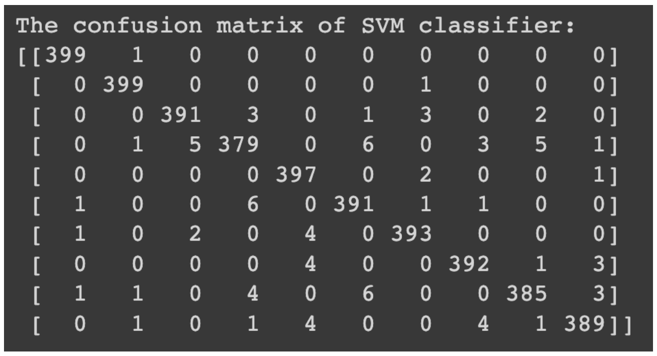
### 4.3.2 Confusion Matrix



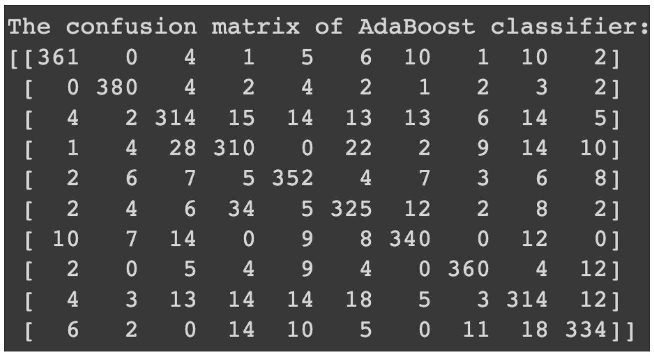
```
The confusion matrix of 3-NN classifier:
[[400   0   0   0   0   0   0   0   0   0]
 [  0 400   0   0   0   0   0   0   0   0]
 [  4   0 389   0   1   1   2   2   1   0]
 [  0   1   3 383   0   8   1   1   2   1]
 [  0   1   0   0 391   0   1   1   0   6]
 [  2   0   1   8   0 385   1   1   2   0]
 [  2   5   1   0   1   1 388   0   2   0]
 [  0   2   1   0   2   0   0 389   0   6]
 [  1   2   1   4   2  14   1   2 368   5]
 [  0   1   0   1   5   0   0   7   0 386]]
```



```
The confusion matrix of MLP-NN classifier:
[[396   0   0   0   3   1   0   0   0   0]
 [  0 399   0   0   0   0   0   1   0   0]
 [  4   0 378   3   2   1   6   3   3   0]
 [  1   1   5 371   0  11   0   7   3   1]
 [  0   1   0   0 397   0   2   0   0   0]
 [  2   0   0   4   0 391   1   2   0   0]
 [  2   3   2   0   2   0 391   0   0   0]
 [  1   0   1   0   1   0   0 397   0   0]
 [  2   5   1   3   0  10   1   5 369   4]
 [  0   0   0   0   6   2   1  15   1 375]]
```

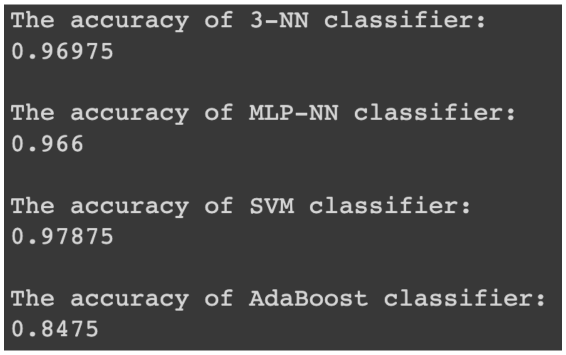


```
The confusion matrix of SVM classifier:
[[399   1   0   0   0   0   0   0   0   0]
 [  0 399   0   0   0   0   1   0   0   0]
 [  0   0 391   3   0   1   3   0   2   0]
 [  0   1   5 379   0   6   0   3   5   1]
 [  0   0   0   0 397   0   2   0   0   1]
 [  1   0   0   6   0 391   1   1   0   0]
 [  1   0   2   0   4   0 393   0   0   0]
 [  0   0   0   0   4   0   0 392   1   3]
 [  1   1   0   4   0   6   0   0 385   3]
 [  0   1   0   1   4   0   0   4   1 389]]
```



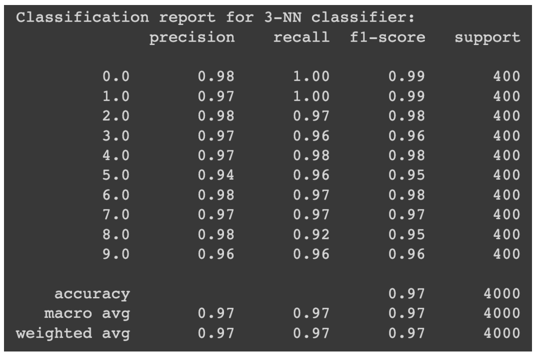
```
The confusion matrix of AdaBoost classifier:
[[361   0   4   1   5   6  10   1  10   2]
 [  0 380   4   2   4   2   1   2   3   2]
 [  4   2 314  15  14  13  13   6  14   5]
 [  1   4  28 310   0  22   2   9  14  10]
 [  2   6   7   5 352   4   7   3   6   8]
 [  2   4   6  34   5 325  12   2   8   2]
 [ 10   7  14   0   9   8 340   0  12   0]
 [  2   0   5   4   9   4   0 360   4  12]
 [  4   3  13  14  14  18   5   3 314  12]
 [  6   2   0  14  10   5   0  11  18 334]]
```

### 4.3.3 Accuracy

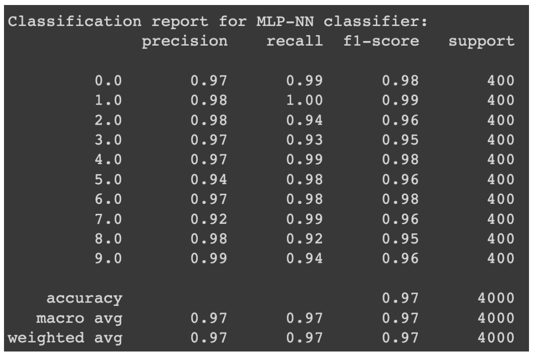


```
The accuracy of 3-NN classifier:
0.96975

The accuracy of MLP-NN classifier:
0.966

The accuracy of SVM classifier:
0.97875

The accuracy of AdaBoost classifier:
0.8475
```

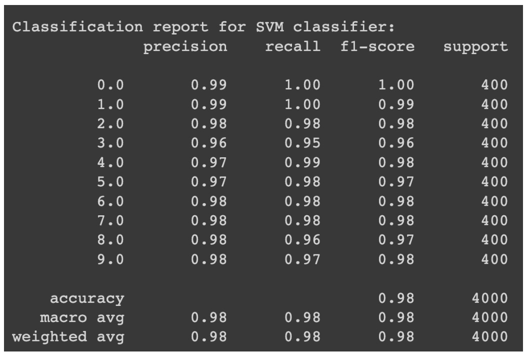
### 4.3.4 Precision & 4.3.5 Recall



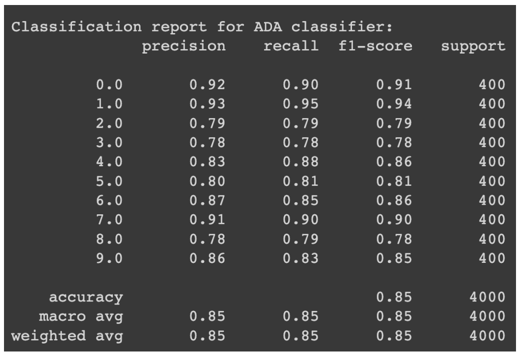
```
Classification report for 3-NN classifier:
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99       400
         1.0       0.97      1.00      0.99       400
         2.0       0.98      0.97      0.98       400
         3.0       0.97      0.96      0.96       400
         4.0       0.97      0.98      0.98       400
         5.0       0.94      0.96      0.95       400
         6.0       0.98      0.97      0.98       400
         7.0       0.97      0.97      0.97       400
         8.0       0.98      0.92      0.95       400
         9.0       0.96      0.96      0.96       400

    accuracy                           0.97      4000
   macro avg       0.97      0.97      0.97      4000
weighted avg       0.97      0.97      0.97      4000
```



```
Classification report for MLP-NN classifier:
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98       400
         1.0       0.98      1.00      0.99       400
         2.0       0.98      0.94      0.96       400
         3.0       0.97      0.93      0.95       400
         4.0       0.97      0.99      0.98       400
         5.0       0.94      0.98      0.96       400
         6.0       0.97      0.98      0.98       400
         7.0       0.92      0.99      0.96       400
         8.0       0.98      0.92      0.95       400
         9.0       0.99      0.94      0.96       400

    accuracy                           0.97      4000
   macro avg       0.97      0.97      0.97      4000
weighted avg       0.97      0.97      0.97      4000
```



```
Classification report for SVM classifier:
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00       400
         1.0       0.99      1.00      0.99       400
         2.0       0.98      0.98      0.98       400
         3.0       0.96      0.95      0.96       400
         4.0       0.97      0.99      0.98       400
         5.0       0.97      0.98      0.97       400
         6.0       0.98      0.98      0.98       400
         7.0       0.98      0.98      0.98       400
         8.0       0.98      0.96      0.97       400
         9.0       0.98      0.97      0.98       400

    accuracy                           0.98      4000
   macro avg       0.98      0.98      0.98      4000
weighted avg       0.98      0.98      0.98      4000
```



```
Classification report for ADA classifier:
              precision    recall  f1-score   support

         0.0       0.92      0.90      0.91       400
         1.0       0.93      0.95      0.94       400
         2.0       0.79      0.79      0.79       400
         3.0       0.78      0.78      0.78       400
         4.0       0.83      0.88      0.86       400
         5.0       0.80      0.81      0.81       400
         6.0       0.87      0.85      0.86       400
         7.0       0.91      0.90      0.90       400
         8.0       0.78      0.79      0.78       400
         9.0       0.86      0.83      0.85       400

    accuracy                           0.85      4000
   macro avg       0.85      0.85      0.85      4000
weighted avg       0.85      0.85      0.85      4000
```

### 4.3.5 Summary

Among the four classifiers with their optimal set of hyper-parameters, the SVM classifier performed the best and the AdaBoost classifier performed the worst. This does not exactly match my expectation. Since I expected the MLP-NN classifier performed the best and the K-NN classifier performed the worst.

The SVM classifier performed the best, which might because that we tuned an appropriate kernel functions o that a non-linear decision surface is able to transformed to a linear equation in high dimensions. While the AdaBoost classifier performed the worst, which might because that our tuned parameters did not perform well in turning a weak-learner into a stronger one. Moreover, the K-NN and MLP-NN had a good performance, with a very close accuracy, which is only slightly smaller than that of SVM.

The computation cost ranking is:
MLP-NN classifier > SVM classifier > AdaBoost classifier > K-NN classifier