Kopal Garg, 1003063221
Team Members: Rohan Deepak Ajwani, Yujie Chen

Q1.

1. Backpropagation – 20pts.

The goal of this exercise is to help you practice how backpropagation works. We consider a simple variation of the feedforward fully-connected network. In the usual feedforward fully-connected network, each layer is connected to its previous layer. The main difference here is that one of the hidden layers in this network is connected to the input too. The computation graph and how each computation is performed is as follows:

$$
\begin{tikzcd}
x \arrow[r, "W^{(1)}"] \arrow[rrr, bend right] & z_1 \arrow[r, "\sigma"] & h \arrow[r] & z_2 \arrow[r, "W^{(2)}"] & y \arrow[r] & \mathcal{L}
\end{tikzcd}
$$

$$
\begin{aligned}
z_1 &= W^{(1)}x \quad \text{with } x \in \mathbb{R}^d \\
h &= \sigma(z_1) \quad \text{with } h \in \mathbb{R}^d \\
z_2 &= h + x \\
y &= W^{(2)}z_2 \\
\mathcal{L} &= \frac{1}{2}(y-t)^2 \text{ with } t \in \mathbb{R}.
\end{aligned}
$$

Here σ is the activation function, and you can assume that it is differentiable. Answer the following questions:

(a) **[4pt]** Determine the dimensions of $W^{(1)}$, $W^{(2)}$, z_1, and z_2.

x is an input vector of shape $(d,1)$
h is the first hidden layer of shape $(d,1)$
$W^{(1)}$ is a weight matrix of shape (d,d) $\Rightarrow$ $W^{(1)}_{d\times d}$
$z_1 = W^{(1)}_{d\times d}\, x_{d\times 1} \Rightarrow z_{1\, d\times 1}$
$W^{(2)}$ is a weight matrix of shape $(1,d)$ because, assuming y is 1×1 and given that $y_{1\times 1} = W^{(2)}_{1\times d}\, z_{2\, d\times 1}$ $\therefore$ $W^{(2)}_{1\times d}$
$z_2 = h_{d\times 1} + x_{d\times 1} \Rightarrow z_{2\, d\times 1}$

Kopal Garg, 1003063221

(b) **[2pt]** Calculate the number of parameters in this network, as a function of d.

The total # parameters would be equal to the sum of total weight matrix elements.

$W^{(1)}_{d \times d}$ $\therefore$ total # parameters $= d^2$

$W^{(2)}_{1 \times d}$ $\therefore$ total # parameters $= d$

overall parameters in this network $= d^2 + d$

(Note: if we had a bias term, then the number of parameters would include one weight of connection with bias, and the total number of parameters would be d^2+d+1.)

Kopal Garg, 1003063221

(c) **[14pt]** Compute the gradient of loss $\mathcal{L}$ with respect to all variables. That is, compute

- $\bar{y} = \frac{\partial \mathcal{L}}{\partial y} = \ldots$
- $\bar{W}^{(2)} = \frac{\partial \mathcal{L}}{\partial W^{(2)}} = \ldots$
- $\bar{z}_2 = \ldots$
- $\bar{h} = \ldots$
- $\bar{z}_1 = \ldots$
- $\bar{W}^{(1)} = \ldots$
- $\bar{x} = \ldots$

$$\bar{y} = \frac{\partial L}{\partial y} = \frac{\partial}{\partial y}\left(\frac{1}{2}(y-t)^2\right) = y - t$$

$$\bar{W}^{(2)} = \frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial W^{(2)}} = \bar{y}\frac{\partial}{\partial W^{(2)}}\left(W^{(2)} z_2\right) = \bar{y} z_2$$

$$\bar{z}_2 = \frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_2} = \bar{y}\frac{\partial}{\partial z_2}\left(W^{(2)} z_2\right) = \bar{y} W^{(2)}$$

$$\bar{h} = \frac{\partial L}{\partial h} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial h} = \bar{z}_2 \frac{\partial}{\partial h}\left(h + x\right) = \bar{z}_2$$

$$\bar{z}_1 = \frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial z_1} = \bar{h}\frac{\partial}{\partial z_1}\left(\sigma(z_1)\right) = \bar{h}\sigma'(z_1)$$

$$\bar{W}^{(1)} = \frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial W^{(1)}} = \bar{z}_1 \frac{\partial}{\partial W^{(1)}} W^{(1)} x = \bar{z}_1 x$$

$$\begin{aligned}
\bar{x} = \frac{\partial L}{\partial x} &= \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial x} + \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial x} \\
&= \bar{z}_2 \frac{\partial h + x}{\partial x} + \bar{z}_1 \frac{\partial}{\partial x} W^{(1)} x \\
&= \bar{z}_2 + \bar{z}_1 W^{(1)} \quad \text{condensed answer}
\end{aligned}$$

Expanding this out, we get:

$$\begin{aligned}
&= \bar{z}_2 + \bar{z}_1 W^{(1)} \\
&= \bar{y} W^{(2)} + \bar{h}\sigma'(z_1) W^{(1)} \\
&= (y-t) W^{(2)} + \bar{z}_2 \sigma'(z_1) W^{(1)} \\
&= (y-t) W^{(2)} + \bar{y} W^{(2)} \sigma'(z_1) W^{(1)} \\
&= (y-t) W^{(2)} + (y-t) W^{(2)} \sigma'(z_1) W^{(1)} \\
&= (y-t) W^{(2)} \left[1 + \sigma'(z_1) W^{(1)}\right] \quad \text{expanded answer}
\end{aligned}$$

Kopal Garg, 1003063221

2. Multi-Class Logistic Regression – 10pts.
The goal of this exercise is to verify the formula on Slide 91 of Lecture 3. Consider

$$\mathbf{z} = W\mathbf{x}$$
$$\mathbf{y} = \text{softmax}(\mathbf{z})$$
$$\mathcal{L}_{\text{CE}}(\mathbf{t}, \mathbf{y}) = -\mathbf{t}^\top \log \mathbf{y} = -\sum_{k=1}^{K} t_k \log y_k$$

Note that if $x \in \mathbb{R}^d$, the dimension of W is $K \times d$. We denote its k-th row by $\mathbf{w}_k$. The vector $\mathbf{y}$ is a function of W and x. And the output $\mathbf{t}$ is a one-hot encoding of the output.

Recall that the k-th component of $\mathbf{y}$ is

$$y_k = \text{softmax}(z_1, \ldots, z_K)_k = \frac{\exp(z_k)}{\sum_{k'=1}^{K} \exp(z_{k'})}.$$

(a) [**5pt**] Compute

$$\frac{\partial y_k}{\partial z_{k'}},$$

for any $k, k' = 1, \ldots, K$ (note that k and k' may or may not be the same). Try to write it in a compact form (no $\exp(\cdots)$ would be needed).

$$y_K = \frac{\exp(z_K)}{\sum_{K'=1}^{K} \exp(z_{K'})} = \frac{e^{z_K}}{\sum_{K'=1}^{K} e^{z_{K'}}}$$

[Piazza @165]

Let $j = K'$ and $z_{K'} = z_j$ in order to avoid confusion.

$$y_K = \frac{e^{z_K}}{\sum_{j=1}^{K} e^{z_j}}$$

Taking the derivative of both sides, w.r.t z_j:

$$\frac{\partial y_K}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{e^{z_K}}{\sum_{j=1}^{K} e^{z_j}}\right)$$

<u>case 1 if $j = K$</u>

Using the quotient rule for derivatives:

$$= \frac{\left(\frac{\partial}{\partial z_j} e^{z_K}\right)\left(\sum_{j=1}^{K} e^{z_j}\right) - \left(e^{z_K}\right)\left(\frac{\partial}{\partial z_j}\sum_{j=1}^{K} e^{z_j}\right)}{\left(\sum_{j=1}^{K} e^{z_j}\right)^2}$$

$$= \frac{\left(e^{z_K}\right)\left(\sum_{j=1}^{K} e^{z_j}\right) - \left(e^{z_K}\right)\left(e^{z_j}\right)}{\left(\sum_{j=1}^{K} e^{z_j}\right)^2}$$

$$= \frac{e^{z_k}\left[\sum_{j=1}^{K} e^{z_j} - e^{z_j}\right]}{\left(\sum_{j=1}^{K} e^{z_j}\right)^2}$$

$$= \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \cdot \left[\frac{\sum_{j=1}^{K} e^{z_j} - e^{z_j}}{\left(\sum_{j=1}^{K} e^{z_j}\right)}\right]$$

Its given that $y_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$

$$= y_k \left[\underbrace{\cancel{\frac{\sum_{j=1}^{K} e^{z_j}}{\sum_{j=1}^{K} e^{z_j}}}}_{1} - \frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_j}}\right]$$

$$= y_k \left[1 - y_j\right]$$

Case 2 if $j \neq k$

$$\frac{\partial y_k}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}\right)$$

Using the quotient rule for derivatives:

$$= \frac{\left(\frac{\partial}{\partial z_j} e^{z_k}\right)\left(\sum_{j=1}^{K} e^{z_j}\right) - \left(e^{z_k}\right)\left(\frac{\partial}{\partial z_j}\sum_{j=1}^{K} e^{z_j}\right)}{\left(\sum_{j=1}^{K} e^{z_j}\right)^2}$$

$$= \frac{0 \cdot \left(\sum_{j=1}^{K} e^{z_j}\right) - \left(e^{z_k}\right)\left(e^{z_j}\right)}{\left(\sum_{j=1}^{K} e^{z_j}\right)^2}$$

$$= \frac{-e^{z_j}}{\sum_{j=1}^{K} e^{z_j}} \cdot \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} = -y_j \, y_k$$

Therefore,

$$\boxed{\begin{array}{lll} \text{If } j = k \; (\text{or } k' = k) & \frac{\partial y_k}{\partial z_j} = y_k(1 - y_j) \text{ or } y_k(1 - y_{k'}) \\ \text{If } j \neq k \; (\text{or } k' \neq k) & \frac{\partial y_k}{\partial z_j} = -y_j\, y_k = -y_{k'}\, y_k \end{array}}$$

(b) **[5pt]** Compute

$$\frac{\partial \mathcal{L}_{\mathrm{CE}}(\mathbf{t}, \mathbf{y}(\mathbf{x}; W))}{\partial \mathbf{w}_k}.$$

You need to show all the derivations in order to get the full mark. The final solution alone will not give you any mark, as it is already shown on the slide.

Switching k to j :

$$\frac{\partial L_{CE}}{\partial w_j} = \underbrace{\frac{\partial L_{CE}}{\partial z_j}}_{①} \cdot \underbrace{\frac{\partial z_j}{\partial w_j}}_{②}$$

Computing ① :

$$\frac{\partial L_{CE}}{\partial z_j} = \frac{\partial}{\partial z_j}\left(-\sum_{k=1}^{K} t_k \log y_k\right)$$

$$= -\sum_{k=1}^{K} t_k \frac{\partial(\log y_k)}{\partial y_k} \frac{\partial y_k}{\partial z_j}$$

$$= -\sum_{k=1}^{K} t_k \left(\frac{1}{y_k}\right) \frac{\partial y_k}{\partial z_j}$$

From 2a:

If $j=k$ $\quad \frac{\partial y_k}{\partial z_j} = y_k(1-y_j)$

If $j \neq k$ $\quad \frac{\partial y_k}{\partial z_j} = -y_j y_k$

$$= \frac{-t_j}{y_k} y_k (1-y_j) - \sum_{k\neq j}^{K} \frac{t_k}{y_k}(-y_k \cdot y_j)$$

$$= -t_j + t_j y_j + \sum_{k\neq j}^{K} t_k y_i$$

$$= y_j \sum_{k\neq j}^{K} t_k + y_j t_j - t_j$$

$$= y_j \left(\sum_{k\neq j}^{K} t_k + t_j\right) - t_j$$

$$= y_j (1) - t_j = y_j - t_j \quad — Ⓐ$$

Computing ② : $\frac{\partial z_j}{\partial w_j}$ since $z = wx$

$$z_j = \sum_{i=1}^{D} w_{ji} x_i + b_j \quad \text{for } j = 1, 2, 3, \ldots, J$$

and D = input dim. J = output dim.

$$\frac{\partial\left(\sum_{i=1}^{D} w_{ji} x + b_j\right)}{\partial w_j} = \frac{\partial \sum_{i=1}^{D} w_{ji} x_i}{\partial w_j} + \frac{\partial b_j}{\partial w_j}$$

$$= \sum_{i=1}^{D} x_i + \phi = x \quad — Ⓑ$$

Using Ⓐ and Ⓑ :

$$\frac{\partial L_{CE}}{\partial w_j} = \frac{\partial L_{CE}}{\partial w_j} \cdot \frac{\partial z_j}{\partial w_j} = (y_j - t_j) \cdot x$$

Switching back to the original symbol, we have:

$$\boxed{\frac{\partial L_{CE}}{\partial w_k} = (y_k - t_k) \cdot x}$$

Kopal Garg, 1003063221

Q3.

0. **[3pt]** Load the data and plot the means for each of the digit classes in the training data (include these in your report). Given that each image is a vector of size 64, the mean will be a vector of size 64 which needs to be reshaped as an 8×8 2D array to be rendered as an image. Plot all 10 means side by side using the same scale.

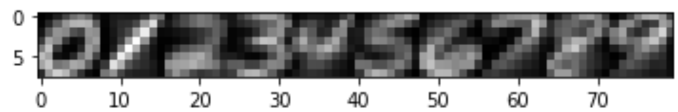


Figure 1. Means of each digit class in the training data represented by 8x8 2D images

Kopal Garg, 1003063221

3.1. *K-NN Classifier – 12pt.*

1. **[6pt]** Build a simple K nearest neighbour classifier using Euclidean distance on the raw pixel data.
   (a) For $K = 1$ report the train and test classification accuracy.
   (b) For $K = 15$ report the train and test classification accuracy.

For K = 1:

```
train_acc_k1 = 100%
test_acc_k1 = 96.9%
```

For K = 15:

```
train_acc_k15 = 96.4%
test_acc_k15 = 96.1%
```

2. **[1pt]** For $K > 1$, K-NN might encounter ties that need to be broken in order to make a decision. Choose any (reasonable) method you prefer and explain it briefly in your report.

We attempted the following 2 tie-breaking strategies:

1. In the event of a tie, we query the test point using an updated K value of K-1. For example, if K is an even number, then using K-1 would help us break the tie by picking the majority.
2. In the event of a tie, pick the digit with the lowest mean distance. For example, if there are 5 votes for digit 8 and 5 votes for digit 3, then we compute the average of the distance for all digit 8 events and compute the average of the distance for all digit 3 events and pick the digit with the lowest mean distance.

We compared the performance of both tie-breaking strategies and found that the second one performed better. Only the results obtained using the second strategy are presented in subsequent sections.

Kopal Garg, 1003063221

3. **[5pt]** Use 10 fold cross validation to find the optimal K in the 1-15 range. You may use the KFold implementation in sklearn. Report this value of K along with the train, validation, and test set classification accuracies, averaged across folds where applicable.

In general, KNN with K=1 implies over-fitting. When K=1 we estimate the probability based on a single sample, i.e., the closest neighbor. This is sensitive to the intricacies (mislabelling, outliers, noise) in the training set. Using a higher value of K tends to lead to a model that is robust to these. We report the train and validation accuracies for all K values in the 1-15 range, and the testing accuracy for K=1 and the next most optimal K value of 4.

	train	validation	K
0	100.000000	96.457143	1
1	100.000000	96.457143	2
2	98.601587	96.571429	3
3	98.650794	96.800000	4
4	98.052381	96.514286	5
5	98.087302	96.557143	6
6	97.484127	96.157143	7
7	97.582540	96.128571	8
8	97.138095	95.842857	9
9	97.136508	95.857143	10
10	96.728571	95.671429	11
11	96.717460	95.685714	12
12	96.390476	95.385714	13
13	96.414286	95.342857	14
14	96.100000	95.171429	15

`Test accuracy using K=1:  96.875 %`

`Optimized test accuracy using K=4:  97.2 %` (based on validation accuracy)

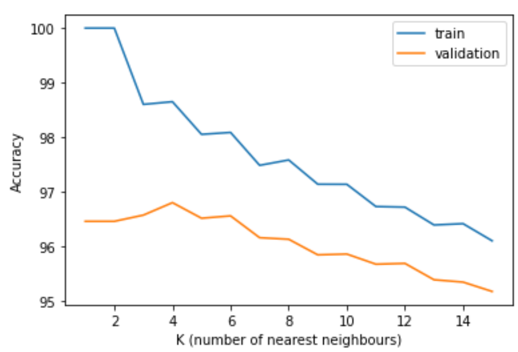


Figure 2. K-means clustering training and validation results for different K values

3.2. *Classifiers comparison – 30pt.* In this section, you will design three different classifiers for the provided hand-written digits data set. You are free to implement your own classifier or to use any package you prefer as long as you will provide readable modular code. We recommend exploring well-known packages such as PyTorch, TensorFlow and scikit-learn. Here is the list of classifiers you are to implement

MLP

1. **MLP - Neural Network Classifier – 10pt.** Design a Multi-Layer Perceptron Neural Network. We are asking for a fully connected network with one-hot encoding output. Your input layer needs one unit for each pixel; given that your are distinguishing among 10 classes, the output layer will need 10 units. Other than the input and the output layers, you are free to deign the best network that provide the least possible error rate, taking overfitting into consideration.

I one-hot encoded the labels using the to_categorical function by keras. I reshaped the input from 64*1 pixels to 8*8. The MLP – NN model has the following architecture:

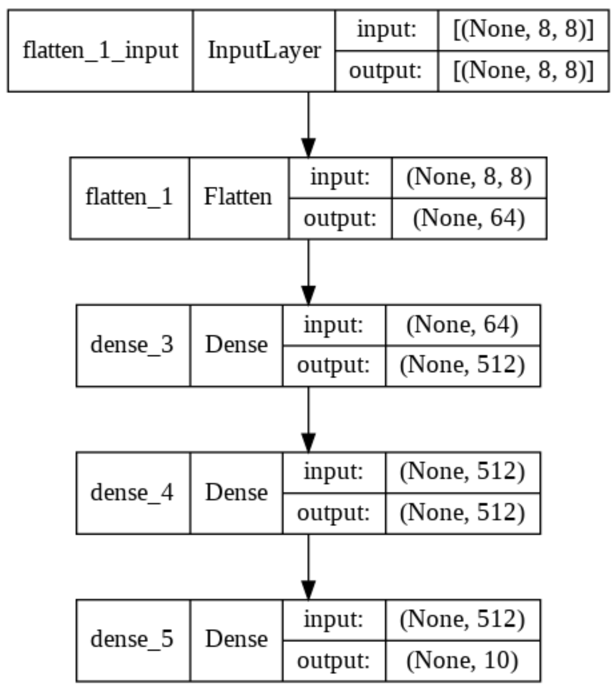


Figure 3. NN Architecture

The output layer has 10 units which represents the number of classes. Finally, I used the np.argmax function to return indices with max. values.

Kopal Garg, 1003063221

SVM Classifier

2. **SVM classifier – 10pt.** Design an SVM classifier. We briefly covered a few kernels in the class. Since you are using external packages, you have a pretty good chance to try kernel beyond what described. Also, there are useful grid search tools that helps you reach the optimal set of hyper-parameters

I performed a 5-fold cross validation grid search on three hyperparameters with sensible values:

```
'gamma': [0.01, 0.001,0.0001]
'C': [1, 10, 100]
'kernel': ['rbf','poly','linear', 'sigmoid']
```

The best SVM model had the following parameters:

```
best_hyperparams {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
```

Because this is a multi-class classification task, I used `OneVsRestClassifier`, which fits one classifier per class. As per the documentation, for each classifier, the class is fitted against all the other classes. I then output the decision_function to get the distance of each sample from the decision boundary for each class. I passed the output through argmax, and finally used label_binarizer to transform multiclass labels to binary labels.

Kopal Garg, 1003063221

AdaBoost Classifier

3. **AdaBoost Classifier – 10pt.** You will have a chance to turn a weak-learner into a strong performing classifier. Again, you have total freedom of the architecture as long as you provide original, readable and modular code.

I performed a 5-fold cross validation grid search on two hyperparameters with sensible values:

```
'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
'n_estimators': list(range(2, 102, 2))
```

The best AdaBoost classifier had the following parameters:

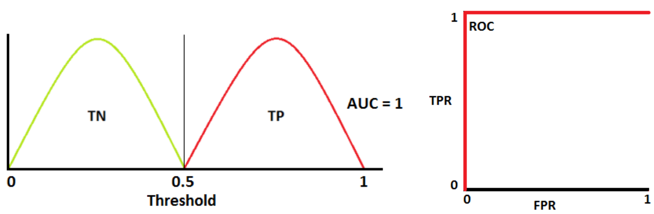
```
best_hyperparams {'learning_rate': 0.3, 'n_estimators': 90}
```

Using predict_proba, I obtained the probability estimates of belonging to each class. I passed the output through argmax, and finally used label_binarizer to transform multiclass labels to binary labels.

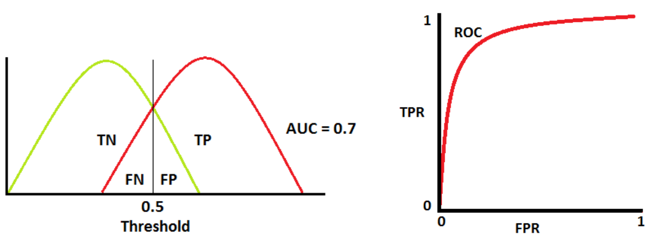
3.3. *Model Comparison – 25pt.* Briefly summarize the performance of each model, including the K-NN model with the optimal K you found. This will be a good chance to study different ways to measuring a classifier performance. Measuring MSE or error rate is not enough, you need to provide, at least, the following metrics for each classifier:

In terms of performance metrics, I used multi-class ROC curves, confusion matrices, accuracy, recall score and precision score.

ROC curve: Since this s a multi-class problem, the idea was to carry out pairwise comparison (one class vs. all other classes). ROC curves are created by plotting the true positive rate against the false positive rate at various threshold settings. In an AUC-ROC curve, a good model has AUC near to the 1 meaning it has a good measure of separability.



A poor model has an AUC near 0 which means it has the poor measure of separability.



Confusion matrix: The data has 10 classes, so our confusion matrix would be a 10×10 matrix, with the left axis showing the true class and the top axis showing the class assigned to an item with that true class. Each element i,j of the matrix is the number of items with true class i that were classified as being in class j.

Accuracy: Categorical accuracy is the percentage of predicted values that match with actual values for one-hot encoded labels.

Precision and Recall: For a given confusion matrix, M:

$$\text{Precision}_i = \frac{M_{ii}}{\sum_j M_{ji}}$$

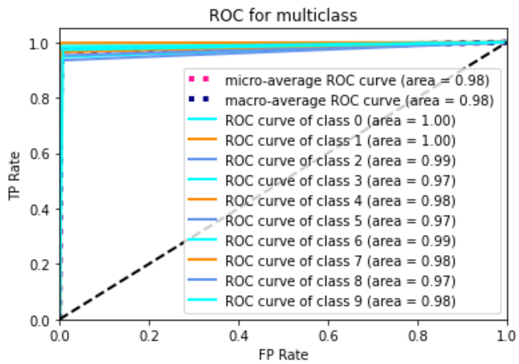
$$\text{Recall}_i = \frac{M_{ii}}{\sum_j M_{ij}}$$

Precision represents the proportion of events where we correctly declared i out of all instances where the algorithm declared i. Recall is the proportion of events where we correctly declared i out of all the cases where the true class is i.

Performance for each classifier:

Kopal Garg, 1003063221

KNN with K = 1 (1-NN)

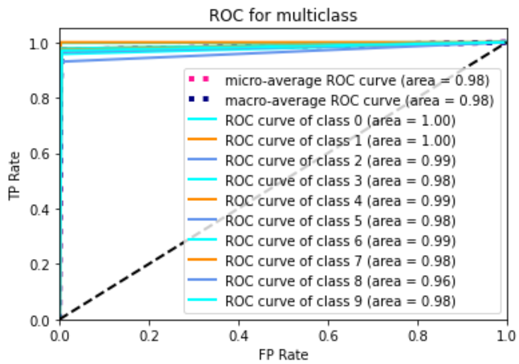


	precision	recall	f1-score	support
0	0.98	0.99	0.99	400
1	0.98	1.00	0.99	400
2	0.98	0.97	0.97	400
3	0.95	0.95	0.95	400
4	0.97	0.96	0.97	400
5	0.95	0.95	0.95	400
6	0.98	0.98	0.98	400
7	0.97	0.97	0.97	400
8	0.99	0.94	0.96	400
9	0.93	0.97	0.95	400
accuracy			**0.97**	4000
macro avg	0.97	0.97	0.97	4000
weighted avg	0.97	0.97	0.97	4000

```
Confusion matrix:
[[398   0   0   0   0   0   1   1   0   0]
 [  0 399   1   0   0   0   0   0   0   0]
 [  4   0 389   3   1   0   0   1   1   1]
 [  0   1   4 379   0  11   1   2   1   1]
 [  0   0   0   0 386   0   2   2   0  10]
 [  1   0   0  12   0 381   3   1   2   0]
 [  0   4   2   0   0   0 393   0   1   0]
 [  0   1   1   0   3   0   0 387   0   8]
 [  2   2   1   2   1   7   0   2 374   9]
 [  0   0   0   1   7   0   0   3   0 389]]


Accuracy:  0.96875
Precision:  0.9689697587760747
Recall:  0.96875
```

Kopal Garg, 1003063221

KNN with K = 4 (4-NN)

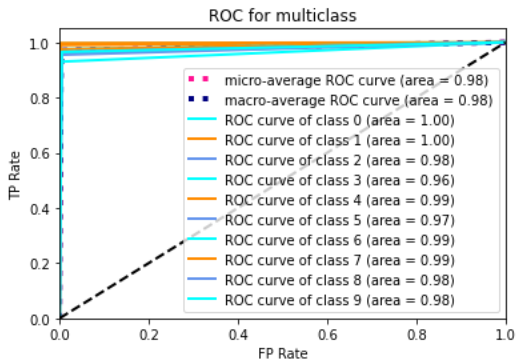


	precision	recall	f1-score	support
0	0.98	1.00	0.99	400
1	0.97	1.00	0.99	400
2	0.99	0.97	0.98	400
3	0.98	0.96	0.97	400
4	0.98	0.98	0.98	400
5	0.95	0.96	0.96	400
6	0.98	0.98	0.98	400
7	0.96	0.97	0.97	400
8	0.97	0.93	0.95	400
9	0.95	0.97	0.96	400
accuracy			0.97	4000
macro avg	0.97	0.97	0.97	4000
weighted avg	0.97	0.97	0.97	4000

```
Confusion matrix:
[[399   0   0   0   0   0   0   1   0   0]
 [  0 400   0   0   0   0   0   0   0   0]
 [  4   0 389   0   1   1   1   2   1   1]
 [  0   1   2 383   0   7   1   2   3   1]
 [  0   1   0   0 391   0   1   1   0   6]
 [  1   0   0   5   0 386   3   1   4   0]
 [  1   4   1   0   0   1 391   0   2   0]
 [  0   2   1   0   2   0   0 389   0   6]
 [  1   2   1   3   1  10   1   3 372   6]
 [  0   1   0   1   4   0   0   6   0 388]]


Accuracy:  0.972
Precision:  0.9721089655006059
Recall:  0.972
```

Kopal Garg, 1003063221

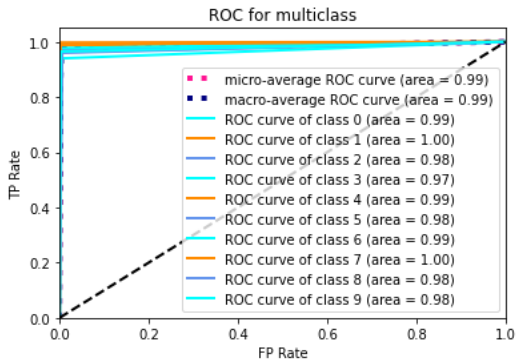
SVM:




```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       400
           1       0.99      1.00      0.99       400
           2       0.96      0.96      0.96       400
           3       0.96      0.93      0.95       400
           4       0.97      0.99      0.98       400
           5       0.95      0.95      0.95       400
           6       0.98      0.97      0.98       400
           7       0.98      0.97      0.98       400
           8       0.96      0.96      0.96       400
           9       0.95      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


Confusion matrix:
[[398   0   0   0   1   0   1   0   0   0]
 [  0 399   0   0   1   0   0   0   0   0]
 [  0   0 384   3   0   2   5   0   5   1]
 [  0   0   9 372   0  10   0   1   6   2]
 [  0   0   1   0 396   0   1   0   0   2]
 [  1   1   0   6   0 382   2   2   2   4]
 [  1   2   3   0   4   0 390   0   0   0]
 [  0   0   1   0   1   0   0 390   0   8]
 [  2   1   1   3   0   5   0   0 386   2]
 [  0   1   1   2   4   1   0   4   1 386]]

Accuracy:  0.97075
Precision:  0.970727484350322
Recall:  0.97075
```

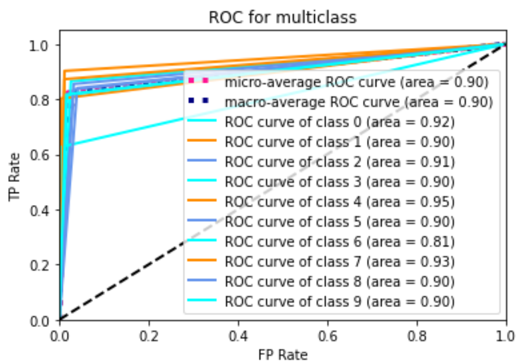
Kopal Garg, 1003063221

MLP




```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       400
           1       1.00      1.00      1.00       400
           2       0.98      0.96      0.97       400
           3       0.96      0.94      0.95       400
           4       0.98      0.99      0.99       400
           5       0.95      0.97      0.96       400
           6       0.98      0.98      0.98       400
           7       0.98      0.99      0.99       400
           8       0.96      0.96      0.96       400
           9       0.97      0.96      0.97       400

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


Confusion matrix:
[[396   0   0   0   2   0   1   0   1   0]
 [  0 399   0   0   0   0   0   0   1   0]
 [  1   0 384   4   0   2   5   1   2   1]
 [  0   0   5 376   0  10   0   1   6   2]
 [  0   0   0   0 395   0   1   0   0   4]
 [  2   0   0   4   0 389   1   2   2   0]
 [  1   1   2   0   1   2 391   0   2   0]
 [  0   0   0   0   1   0   0 397   0   2]
 [  0   0   0   5   0   7   0   1 385   2]
 [  0   1   1   2   3   1   0   4   2 386]]

Accuracy:  0.9745
Precision:  0.9745474547409659
Recall:  0.9745
```

Kopal Garg, 1003063221



	precision	recall	f1-score	support
0	0.79	0.86	0.82	400
1	0.96	0.80	0.88	400
2	0.75	0.85	0.80	400
3	0.84	0.81	0.83	400
4	0.90	0.90	0.90	400
5	0.75	0.83	0.79	400
6	0.92	0.63	0.75	400
7	0.89	0.87	0.88	400
8	0.70	0.84	0.76	400
9	0.85	0.82	0.83	400
accuracy			0.82	4000
macro avg	0.83	0.82	0.82	4000
weighted avg	0.83	0.82	0.82	4000

```
Confusion matrix:
[[346   0   9   2   0  22   3   0  18   0]
 [  0 321   3   8  12   6   0   0  50   0]
 [  5   1 342  10   4  10  11   0  16   1]
 [  2   0  40 326   0  20   0   0   9   3]
 [  0   2   3   0 361   1   7   2   8  16]
 [  5   1  13  27   5 334   1   1  12   1]
 [ 77   2  15   0   3  46 251   0   6   0]
 [  0   2   1   5   5   0   0 349   9  29]
 [  5   2  30   9   1   9   0   1 336   7]
 [  0   2   2   3  12   0   0  37  17 327]]

Accuracy:  0.82325
Precision:  0.8338660191887979
Recall:  0.82325
```

3.3. *Model Comparison – 25pt.* Briefly summarize the performance of each model, including the K-NN model with the optimal K you found. This will be a good chance to study different ways to measuring a classifier performance. Measuring MSE or error rate is not enough, you need to provide, at least, the following metrics for each classifier:

The performance rankings (from best to worst) match my expectations:

Best

MLP:
```
Accuracy:  0.9745
Precision:  0.9745474547409659
Recall:  0.9745
```

4-NN:
```
Accuracy:  0.972
Precision:  0.9721089655006059
Recall:  0.972
```

SVM:
```
Accuracy:  0.97075
Precision:  0.970727484350322
Recall:  0.97075
```

1-NN:
```
Accuracy:  0.96875
Precision:  0.9689697587760747
Recall:  0.96875
```

AdaBoost:
```
Accuracy:  0.82325
Precision:  0.8338660191887979
Recall:  0.82325
```

Worst

I had expected that a neural network architecture would outperform any traditional ML classifier, as it has a more complicated architecture which can handle non-linearities.

It was also expected that KNN with K = 4 would outperform KNN with K = 1, as it would generalize better on the test set. KNN with K = 1 had superior performance on the KNN with K = 4, because the MLP displays the best performance on the testing dataset because we're estimating the probability based on a single sample, i.e., the closest neighbor. This is sensitive to the intricacies (mislabelling, outliers, noise) in the training set. Using a higher value of K tends to lead to a model that is robust to these.

I also expected for SVM to outperform 1-NN on the test set for similar reasons as before. We performed an extensive grid search CV and trained the model using the best set of hyperparameters, which is why SVM outperformed 1-NN. We also used the OneVsRest classifier.

Kopal Garg, 1003063221

It was unexpected that AdaBoost had such poor performance despite implementing a grid search CV on its two hyperparameters. It is an ensemble learning method that combines weak classifiers into a strong classifier to minimize errors. One reason for poor performance may have been the fact that we used OneVsRest classification in SVM, but not in AdaBoost, so it may not have been optimal for multi-class classification tasks. The images could have also been noisy or low resolution (only 8x8) so it may perform better if we use higher resolution images.