Kopal Garg, 1003063221

Team Members: Rohan Deepak Ajwani, Yujie Chen

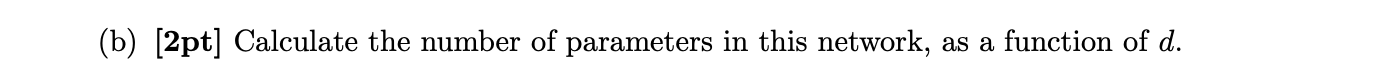
Q1.

Diagram

Description automatically generated

Text, letter

Description automatically generated



Diagram

Description automatically generated

(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 d2+d+1.)

Text

Description automatically generated

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Diagram

Description automatically generated

Q3.Text

Description automatically generated

A black and white checkered background

Description automatically generated with medium confidence

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

Graphical user interface, text

Description automatically generated

For K = 1:

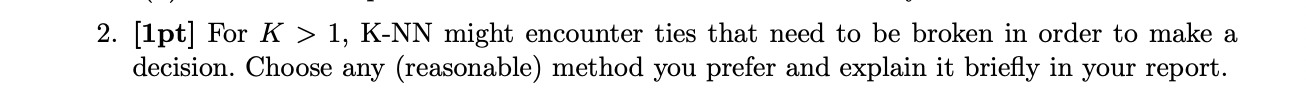
train\_acc\_k1 = 100%

test\_acc\_k1 = 96.9%

For K = 15:

train\_acc\_k15 = 96.4%

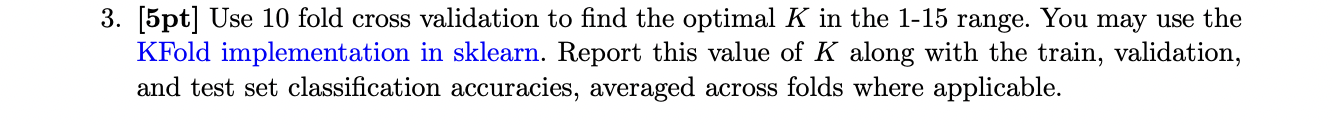
test\_acc\_k15 = 96.1%



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.



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)

Chart, line chart

Description automatically generated

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

Text

Description automatically generated

**MLP**

**Text

Description automatically generated**

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:

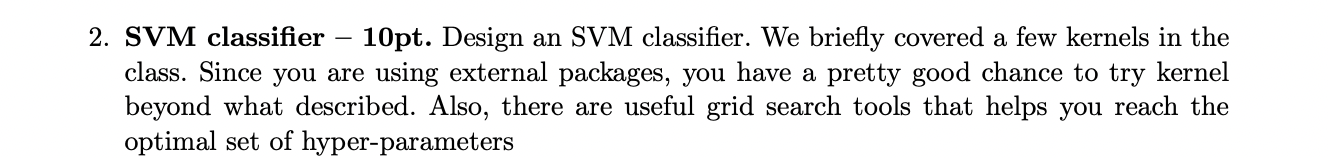
**Diagram

Description automatically generated**

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.

**SVM Classifier**



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]

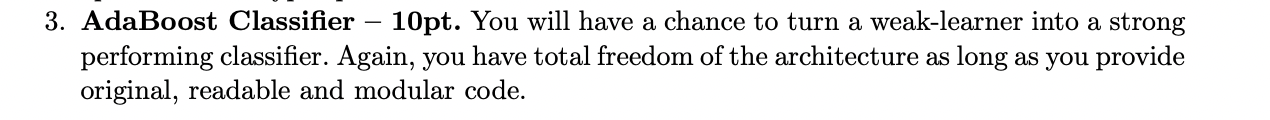
'kernel': ['rbf','poly','linear', 'sigmoid']

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.

**AdaBoost Classifier**



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

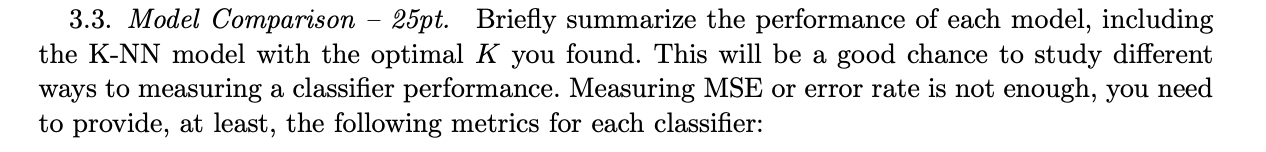
'learning\_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

'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.



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.

Chart

Description automatically generated with medium confidence A picture containing chart

Description automatically generated

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

Diagram

Description automatically generated Chart

Description automatically generated with low confidence

**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

Description automatically generated

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:

**KNN with K = 1 (1-NN)**

Graphical user interface

Description automatically generated with medium confidence

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

**KNN with K = 4 (4-NN)**

**Graphical user interface

Description automatically generated with medium confidence**

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

SVM:

Graphical user interface

Description automatically generated with medium confidence

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

MLP

Graphical user interface

Description automatically generated with medium confidence

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

A picture containing graphical user interface

Description automatically generated

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]

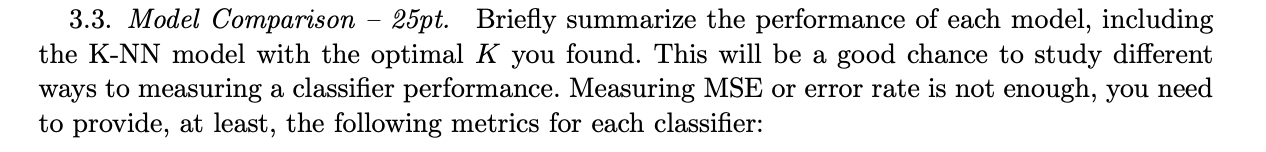
[ 5 2 30 9 1 9 0 1 336 7]

[ 0 2 2 3 12 0 0 37 17 327]]

Accuracy: 0.82325

Precision: 0.8338660191887979

Recall: 0.82325



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:

Worst

Accuracy: 0.82325

Precision: 0.8338660191887979

Recall: 0.82325

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.

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.