MTL782 Assignment - II

Handwritten Digit Recognition

Submitted by:

2021MT60266 Aniket Pandey

2021MT60814 Akshat Chaudhary

2021MT60958 Aditya Arya

Link to Python Notebook: mtl782-assignment-2.ipynb

Part a:

We have implemented five models for the classification in the MNIST dataset. These are (1) Decision Tree, (2) Random Forest, (3) Naïve Bayes Classifier, (4) KNN classifier, and (5) Neural Network classifier. Comparing the performances using k-fold cross-validation (with k=5), we obtain the following:

```
Decision Tree F1 score: 0.866 +/- 0.007
Random Forest F1 score: 0.966 +/- 0.003
Gaussian Naive Bayes F1 score: 0.515 +/- 0.015
Bernoulli Naive Bayes F1 score: 0.828 +/- 0.008
KNN F1 score: 0.969 +/- 0.001
Neural Network F1 score: 0.974 +/- 0.003
```

We obtain thus that in terms of cross-validation performance, we have:
Neural Network > KNN > Random Forest > Decision Tree > Naive Bayes.
Note that in the case of Naive Bayes we have used both Gaussian and Bernoulli Naive Bayes. This is because Bernoulli NB is more suited to the problem statement since the classes are pretty discrete, and hence, a discrete probabilistic distribution works better than a continuous distribution, but Gaussian NB is the more standard one.

(b) Exploration of Different Evaluation Metrics:

The evaluation metrics that we have concentrated on are accuracy, precision, recall, f-1 score and support. This is part of the classification report. We have also produced the confusion Matrix to visualise the models' performances.

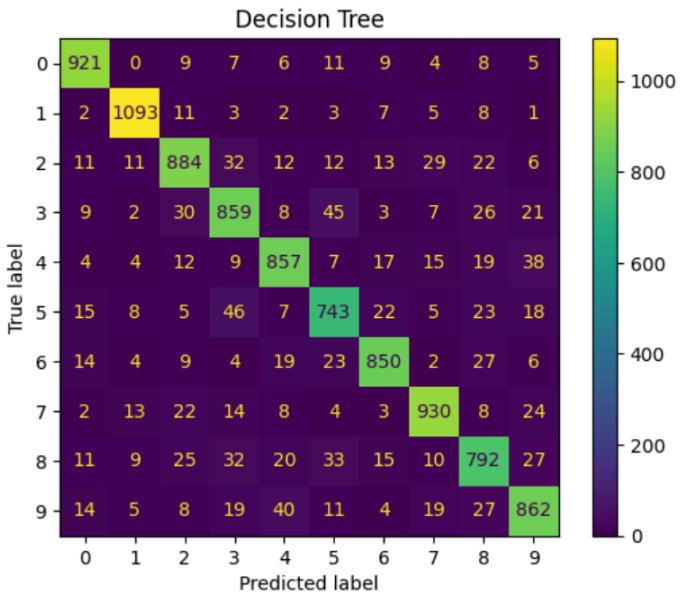
- **Decision Tree**

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

           0       0.92      0.94      0.93       980
           1       0.95      0.96      0.96      1135
           2       0.87      0.86      0.86      1032
           3       0.84      0.85      0.84      1010
           4       0.88      0.87      0.87       982
           5       0.83      0.83      0.83       892
           6       0.90      0.89      0.89       958
           7       0.91      0.90      0.91      1028
           8       0.82      0.81      0.82       974
           9       0.86      0.85      0.85      1009

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

Confusion matrix:



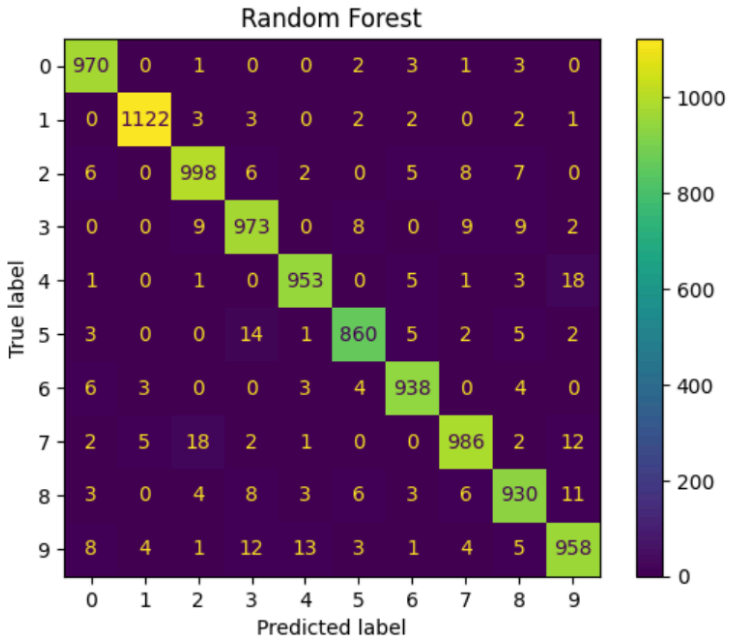
- **Random Forest**

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

           0       0.97      0.99      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.97      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.98      0.97      0.97       982
           5       0.97      0.96      0.97       892
           6       0.98      0.98      0.98       958
           7       0.97      0.96      0.96      1028
           8       0.96      0.95      0.96       974
           9       0.95      0.95      0.95      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

Confusion matrix:



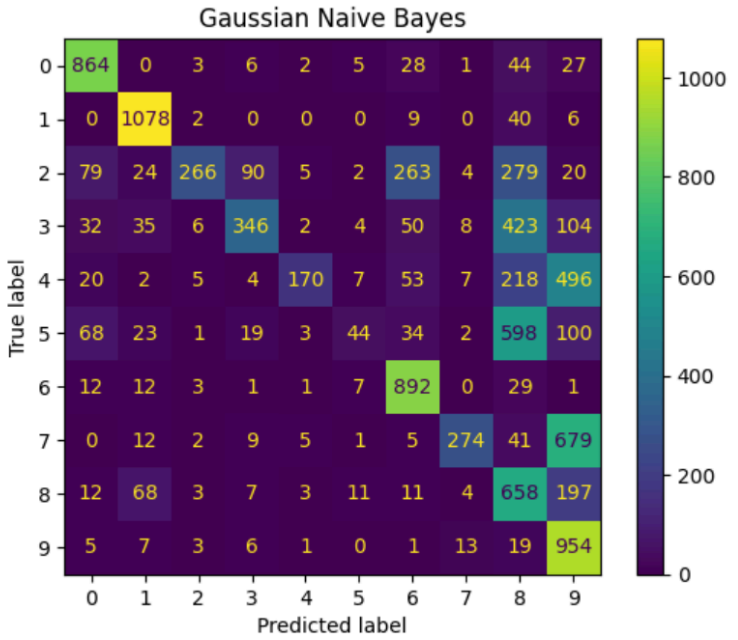
- **Gaussian Naive Bayes**

```
Classification report for classifier Gaussian Naive Bayes:
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       980
           1       0.85      0.95      0.90      1135
           2       0.90      0.26      0.40      1032
           3       0.71      0.34      0.46      1010
           4       0.89      0.17      0.29       982
           5       0.54      0.05      0.09       892
           6       0.66      0.93      0.77       958
           7       0.88      0.27      0.41      1028
           8       0.28      0.68      0.40       974
           9       0.37      0.95      0.53      1009

    accuracy                           0.55     10000
   macro avg       0.69      0.55      0.51     10000
weighted avg       0.69      0.55      0.52     10000
```

Confusion matrix:



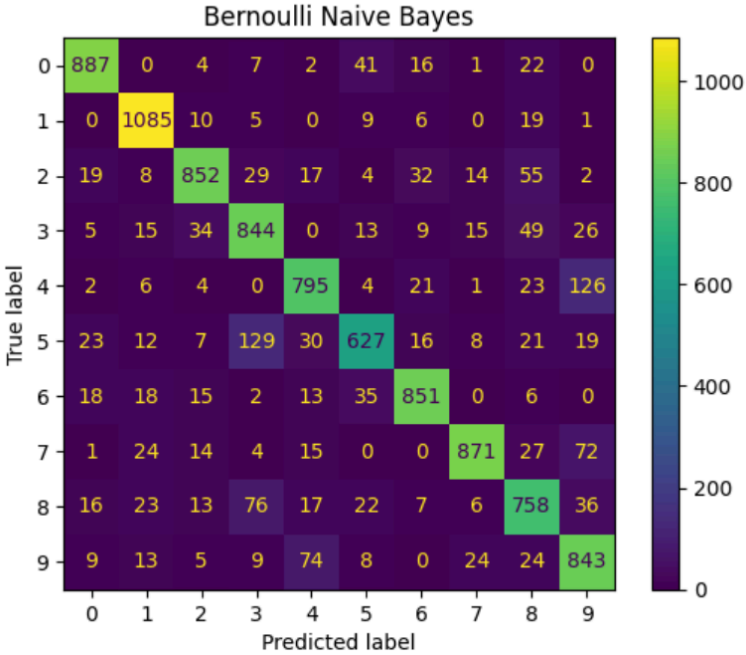
- **Bernoulli Naive Bayes**

```
Classification report for classifier Bernoulli Naive Bayes:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91       980
           1       0.90      0.96      0.93      1135
           2       0.89      0.83      0.86      1032
           3       0.76      0.84      0.80      1010
           4       0.83      0.81      0.82       982
           5       0.82      0.70      0.76       892
           6       0.89      0.89      0.89       958
           7       0.93      0.85      0.89      1028
           8       0.75      0.78      0.77       974
           9       0.75      0.84      0.79      1009

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.84     10000
weighted avg       0.84      0.84      0.84     10000
```

Confusion matrix:

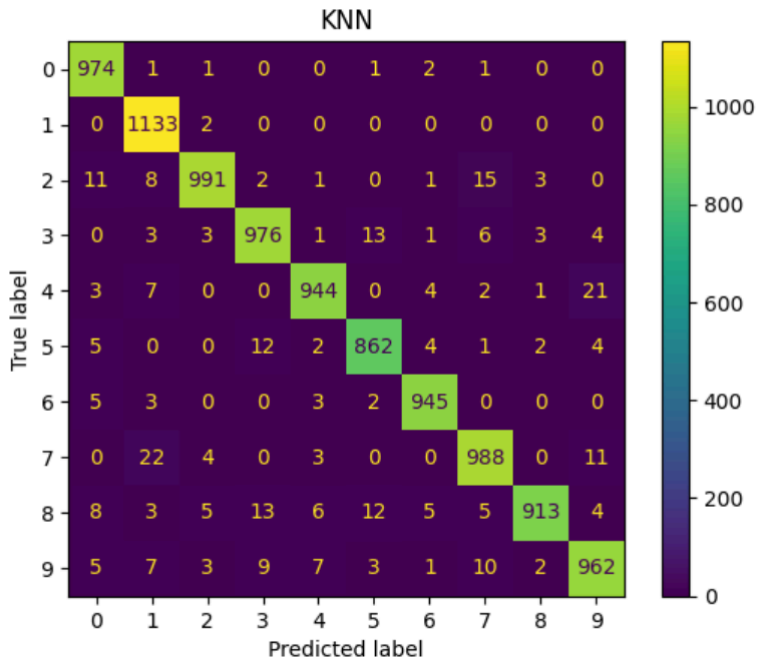


- **Classifier KNN**

Classification report for classifier KNN:

	precision	recall	f1-score	support
0	0.96	0.99	0.98	980
1	0.95	1.00	0.98	1135
2	0.98	0.96	0.97	1032
3	0.96	0.97	0.97	1010
4	0.98	0.96	0.97	982
5	0.97	0.97	0.97	892
6	0.98	0.99	0.98	958
7	0.96	0.96	0.96	1028
8	0.99	0.94	0.96	974
9	0.96	0.95	0.95	1009
accuracy			0.97	10000
macro avg	0.97	0.97	0.97	10000
weighted avg	0.97	0.97	0.97	10000

Confusion matrix:

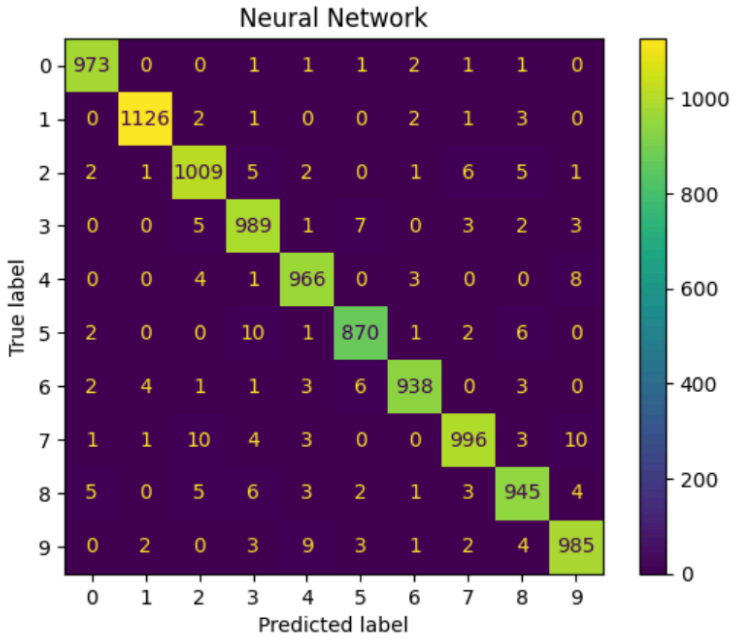


- **Neural Network**

Classification report for classifier Neural Network:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.97	0.98	0.98	1032
3	0.97	0.98	0.97	1010
4	0.98	0.98	0.98	982
5	0.98	0.98	0.98	892
6	0.99	0.98	0.98	958
7	0.98	0.97	0.98	1028
8	0.97	0.97	0.97	974
9	0.97	0.98	0.98	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

Confusion matrix:



(c) Parameter Tuning through Grid Search & Cross Validation

Grid Search

To tune the hyperparameters we have used Grid Search. To use Grid Search, we define a parameter grid, and the algorithm traverses through this whole grid and gives us the best hyperparameters. The results of the same are attached below.

- **Neural Network**

```
param_grid = {
    'hidden_layer_sizes': [(100, 50), (200, 100)],
    'alpha': [0.0001, 0.001]}
```

Best hyperparameters: {'alpha': 0.001, 'hidden_layer_sizes': (200, 100)}
Best validation score: 0.9799166666666667
Test accuracy: 0.9795

- **Decision Tree**

```
param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [ 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

Best hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 5}

Best validation score: 0.8687833333333334

Test accuracy: 0.8795

- **Random Forest**

```
param_grid = {'max_depth': [None, 10, 20],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']}
```

Best hyperparameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1}

Best validation score: 0.9664166666666667

Test accuracy: 0.9694

- **KNN**

```
param_grid = {'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'leaf_size': [10, 20, 30]}
```

Best hyperparameters: {'leaf_size': 10, 'n_neighbors': 3, 'weights': 'distance'}

Best validation score: 0.97115

Test accuracy: 0.9717

- **Bernoulli Naive Bayes**

```
param_grid = {'alpha': [0.1, 0.5, 1.0],
    'fit_prior': [True, False],
    'class_prior': [None,
                    [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
                    [0.05, 0.05, 0.1, 0.1, 0.1, 0.1, 0.1, 0.15, 0.15, 0.1]
                   ]
}
```

Best hyperparameters: {'alpha': 0.1, 'class_prior': None, 'fit_prior': False}

Best validation score: 0.8306666666666667

Test accuracy: 0.8418

- **Gaussian Naive Bayes**

```
param_grid = {"var_smoothing":[1e-10,1e-9,0.01,1]
}
```

Best hyperparameters: {'var_smoothing': 0.01}

Best validation score: 0.7784166666666668

Test accuracy: 0.7893

Randomized Search

In some instances, especially in the case of a robust model like the Neural Network, GridSearch can be very computationally inefficient. To combat this, we can use Randomized Search, which reduces the computational load by instead of traversing the whole parameter grid, taking a random subset of the grid and traversing it. The results of the same are attached below:

- **Neural Network**

```
param_grid = {'hidden_layer_sizes': [(100,),(100, 50), (200, 100)],
    'alpha': [0.0001, 0.001,0.1]
}
```

Best hyperparameters: {'alpha': 0.001, 'hidden_layer_sizes': (100,)}

Best validation score: 0.99699107142857143

- **Decision Tree**

```
param_grid = {'max_depth': [None, 10, 20, 30],
    'min_samples_split': [ 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Best hyperparameters: {'min_samples_split': 10, 'min_samples_leaf': 4, 'max_depth': None}

Best validation score: 0.8683499999999998

- **Random Forest**

```
param_grid = {'max_depth': [None, 10, 20],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}
```

Best hyperparameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1}

Best validation score: 0.9664166666666667

- **KNN**

```
param_grid = {'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'leaf_size': [10, 20, 30]
}
```

Best hyperparameters: {'leaf_size': 20, 'n_neighbors': 3, 'weights': 'distance'}

Best validation score: 0.9700535714285714

- **Gaussian Naive Bayes**

```
param_grid = {"var_smoothing":[1e-10,1e-9,0.01,1]
}
```

Best hyperparameters: {'var_smoothing': 0.01}

Best validation score: 0.0.7784166666666668

- **Bernoulli Naive Bayes**

```
param_grid = {'alpha': [0.1, 0.5, 1.0],
    'fit_prior': [True, False],
    'class_prior': [None,
                    [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
                    [0.05, 0.05, 0.1, 0.1, 0.1, 0.1, 0.1, 0.15, 0.15, 0.1]
                   ]
}
```

Best hyperparameters: {'fit_prior': False, 'class_prior': None, 'alpha': 0.1}

Best validation score: 0.8306666666666667