Capstone Project Report

Name: **Pooja Ramdas Kadam**
Course: AI & ML (Batch - 4)

Problem Statement

Use MNIST dataset to create a classifier for all the 10 digits. First implement the classifier by squeezing the image into a vector and then using a MLP. Now, try the same task using a different machine learning classifier such as an SVM to check the gain in performance by using perceptrons as compared to conventional machine learning techniques.

Prerequisites

Along with Python below packages needed to be installed

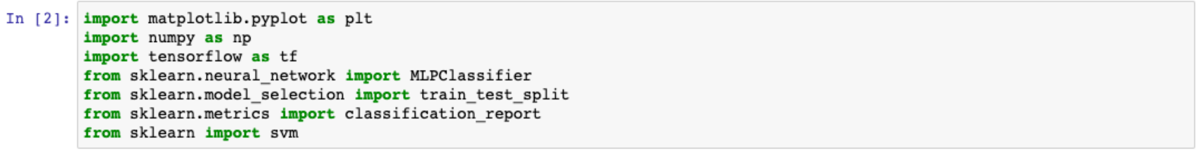
Numpy
Matplotlib

Dataset Used

MNist from tensorflow

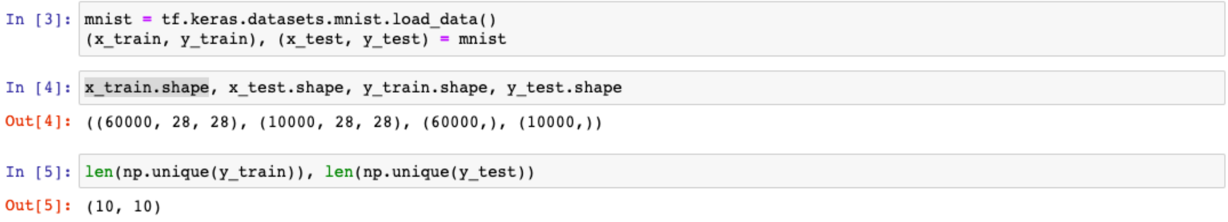
Implementation

Import required libraries



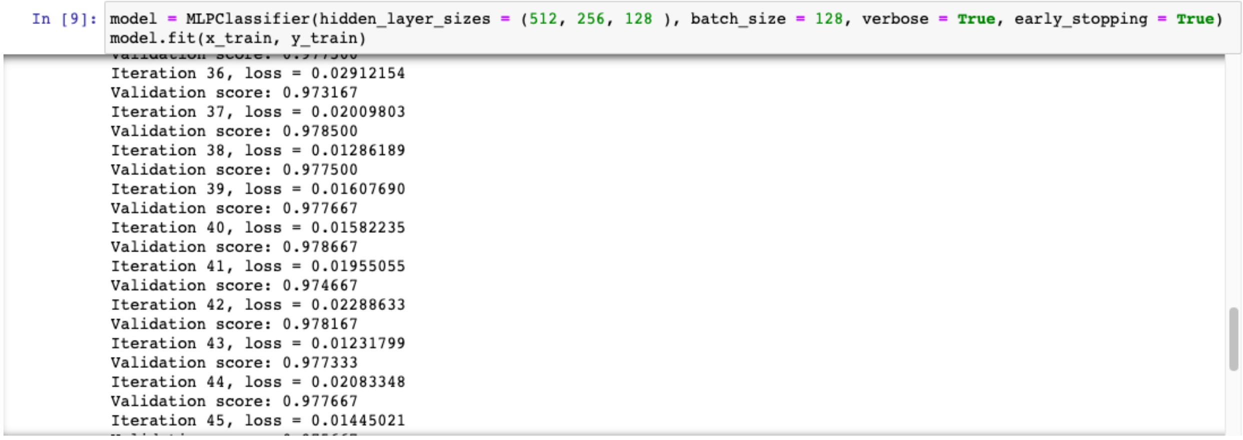
```
In [2]: import matplotlib.pyplot as plt
        import numpy as np
        import tensorflow as tf
        from sklearn.neural_network import MLPClassifier
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import classification_report
        from sklearn import svm
```

Load dataset



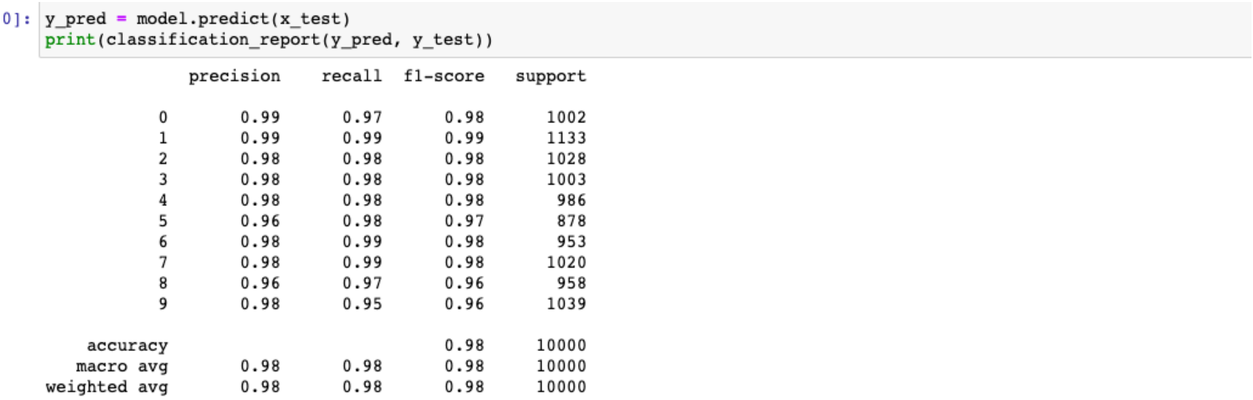
```
In [3]: mnist = tf.keras.datasets.mnist.load_data()
        (x_train, y_train), (x_test, y_test) = mnist

In [4]: x_train.shape, x_test.shape, y_train.shape, y_test.shape

Out[4]: ((60000, 28, 28), (10000, 28, 28), (60000,), (10000,))

In [5]: len(np.unique(y_train)), len(np.unique(y_test))

Out[5]: (10, 10)
```

Apply MLP



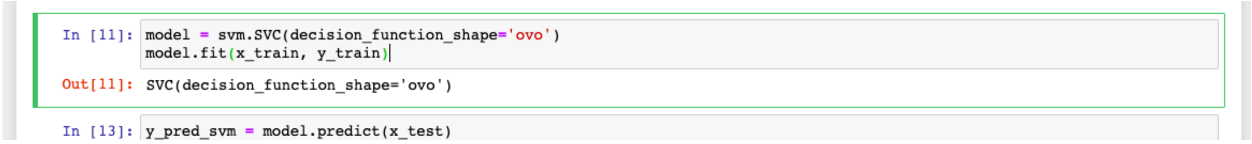
```
In [9]: model = MLPClassifier(hidden_layer_sizes = (512, 256, 128 ), batch_size = 128, verbose = True, early_stopping = True)
        model.fit(x_train, y_train)

        Validation score: 0.977500
        Iteration 36, loss = 0.02912154
        Validation score: 0.973167
        Iteration 37, loss = 0.02009803
        Validation score: 0.978500
        Iteration 38, loss = 0.01286189
        Validation score: 0.977500
        Iteration 39, loss = 0.01607690
        Validation score: 0.977667
        Iteration 40, loss = 0.01582235
        Validation score: 0.978667
        Iteration 41, loss = 0.01955055
        Validation score: 0.974667
        Iteration 42, loss = 0.02288633
        Validation score: 0.978167
        Iteration 43, loss = 0.01231799
        Validation score: 0.977333
        Iteration 44, loss = 0.02083348
        Validation score: 0.977667
        Iteration 45, loss = 0.01445021
```

Accuracy

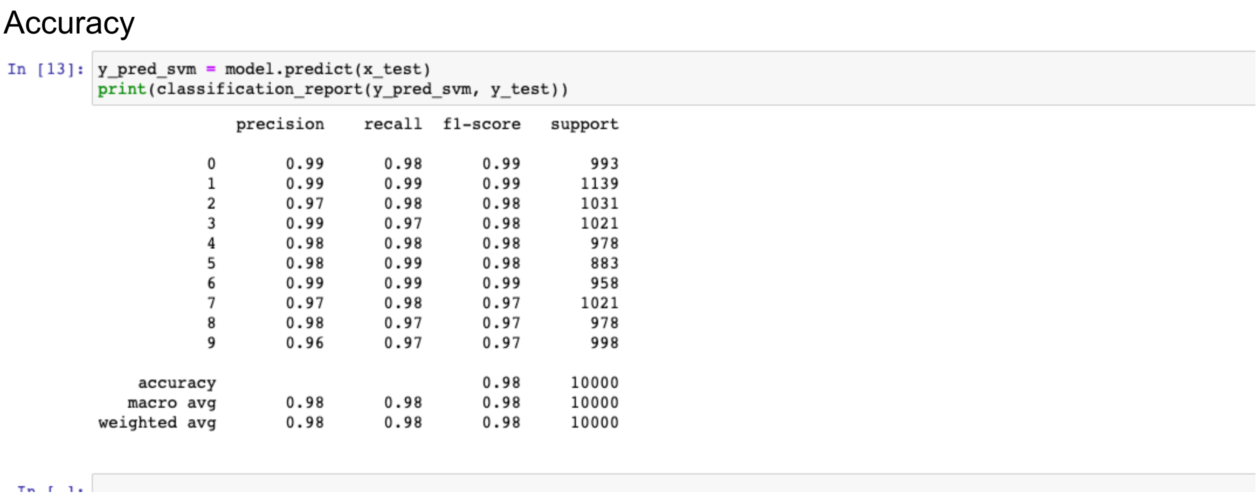


```
0]: y_pred = model.predict(x_test)
    print(classification_report(y_pred, y_test))

              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1002
           1       0.99      0.99      0.99      1133
           2       0.98      0.98      0.98      1028
           3       0.98      0.98      0.98      1003
           4       0.98      0.98      0.98       986
           5       0.96      0.98      0.97       878
           6       0.98      0.99      0.98       953
           7       0.98      0.99      0.98      1020
           8       0.96      0.97      0.96       958
           9       0.98      0.95      0.96      1039

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

Apply SVM



```
 In [11]: model = svm.SVC(decision_function_shape='ovo')
          model.fit(x_train, y_train)

Out[11]: SVC(decision_function_shape='ovo')

 In [13]: y_pred_svm = model.predict(x_test)
```



Accuracy

```
In [13]: y_pred_svm = model.predict(x_test)
         print(classification_report(y_pred_svm, y_test))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       993
           1       0.99      0.99      0.99      1139
           2       0.97      0.98      0.98      1031
           3       0.99      0.97      0.98      1021
           4       0.98      0.98      0.98       978
           5       0.98      0.99      0.98       883
           6       0.99      0.99      0.99       958
           7       0.97      0.98      0.97      1021
           8       0.98      0.97      0.97       978
           9       0.96      0.97      0.97       998

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

```
In [ ]:
```