PROJECT REPORT

Assignment Project Report Handwritten Digit Recognition
NAME: RISHI RAJ SINGH
AI AND ML B-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:
- Software: Python 3

Tools:

- Pandas
- Numpy
- Matplotlib
- Seaborn
- Sklearn

Digit recognition system is the working of a machine to train itself for recognizing the digits from different sources like emails, bank cheque, papers, images, etc. and in different realworld scenarios for online handwriting recognition on computer tablets or system. Developing such a system includes a machine to understand and classify the images of handwritten digits as 10 digits (0–9). Handwritten digits from the MNIST database has been one of the most famous databases among the machine learning community for many recent decades.

Implementation:

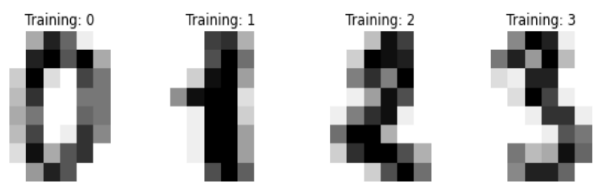
- Load all required libraries

```
In [1]:  import matplotlib.pyplot as plt
         import pandas as pd
         import numpy as np
         import seaborn as sns
         import cv2
         from sklearn.datasets import load_digits
         from sklearn import preprocessing
         from collections import Counter
         from skimage.feature import hog
```

- Loading Dataset

```
In [2]:  digits =load_digits()

         _, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
         for ax, image, label in zip(axes, digits.images, digits.target):
             ax.set_axis_off()
             ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
             ax.set_title('Training: %i' % label)
```



To apply a classifier on this data, we need to flatten the images, turning each 2-D array of grayscale values from shape (8, 8) into shape (64,). Subsequently, the entire dataset will be of shape (n_samples, n_features), where n_samples is the number of images and n_features is the total number of pixels in each image.

We can then split the data into train and test subsets and fit a classifier on the train samples. The fitted classifier can subsequently be used to predict the value of the digit for the samples in the test subset.

- Training And testing

In [5]:
```python
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
```

In [6]:
```python
# Split data into 50% train and 50% test subsets
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.5, shuffle=False)
```

- ## Classifier And Accuracy

In [7]:
```python
Model = MLPClassifier(activation='relu', hidden_layer_sizes=(200, 200), alpha = 0.3)
Model.fit(X_train, y_train)
```

```
C:\Users\dell\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:692: ConvergenceWarning: Stochast
ic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Out[7]:
```
MLPClassifier(alpha=0.3, hidden_layer_sizes=(200, 200))
```

In [8]:
```python
print("Training Score :: {}\n".format(Model.score(X_train, y_train)))
print("Testing Score :: {}\n".format(Model.score(X_test, y_test)))
```

```
Training Score :: 1.0

Testing Score :: 0.949944382647386
```

- ## Performance

In [12]:
```python
print(classification_report(y_test,predicted))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98        88
           1       0.95      0.91      0.93        91
           2       0.97      1.00      0.98        86
           3       0.96      0.86      0.91        91
           4       0.99      0.96      0.97        92
           5       0.92      0.96      0.94        91
           6       0.98      0.99      0.98        91
           7       0.97      0.97      0.97        89
           8       0.92      0.92      0.92        88
           9       0.87      0.97      0.92        92

    accuracy                           0.95       899
   macro avg       0.95      0.95      0.95       899
weighted avg       0.95      0.95      0.95       899
```

In [13]:
```python
    axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
```

- Applying SVC

In [15]:
```
from sklearn.svm import SVC
```

In [16]:
```

clf = SVC(gamma=0.001)
```

In [17]:
```
clf.fit(X_train,y_train)
pred = clf.predict(X_test)
```

- SVC Performance

In [18]:
```

print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        88
           1       0.99      0.97      0.98        91
           2       0.99      0.99      0.99        86
           3       0.98      0.87      0.92        91
           4       0.99      0.96      0.97        92
           5       0.95      0.97      0.96        91
           6       0.99      0.99      0.99        91
           7       0.96      0.99      0.97        89
           8       0.94      1.00      0.97        88
           9       0.93      0.98      0.95        92

    accuracy                           0.97       899
   macro avg       0.97      0.97      0.97       899
weighted avg       0.97      0.97      0.97       899
```

- Decision Tree And Performace

```
In [20]:  from sklearn.tree import DecisionTreeClassifier

In [21]:  clf2 = DecisionTreeClassifier()

In [22]:  clf2.fit(X_train,y_train)
          pred2 = clf2.predict(X_test)

In [23]:  print(classification_report(y_test,pred2))
```

```
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        88
           1       0.78      0.65      0.71        91
           2       0.86      0.74      0.80        86
           3       0.65      0.75      0.69        91
           4       0.83      0.87      0.85        92
           5       0.61      0.74      0.67        91
           6       0.90      0.89      0.90        91
           7       0.86      0.64      0.74        89
           8       0.61      0.67      0.64        88
           9       0.69      0.75      0.72        92

    accuracy                           0.76       899
   macro avg       0.78      0.76      0.77       899
weighted avg       0.77      0.76      0.76       899
```

```
In [24]:  from sklearn.ensemble import RandomForestClassifier
```

- Logistic Regression and performance

```
In [28]:  from sklearn.linear_model import LogisticRegression

In [29]:  clf4 = LogisticRegression()
          clf4.fit(X_train,y_train)
          pred4 = clf4.predict(X_test)
```

```
C:\Users\dell\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converg
e (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
In [30]:  print(classification_report(y_test,pred4))
```

```
              precision    recall  f1-score   support

           0       0.99      0.95      0.97        88
           1       0.94      0.90      0.92        91
           2       0.99      0.98      0.98        86
           3       0.94      0.84      0.88        91
           4       0.98      0.91      0.94        92
           5       0.88      0.95      0.91        91
           6       0.91      0.99      0.95        91
           7       0.98      0.96      0.97        89
           8       0.89      0.90      0.89        88
           9       0.84      0.93      0.89        92

    accuracy                           0.93       899
   macro avg       0.93      0.93      0.93       899
weighted avg       0.93      0.93      0.93       899
```

- O/P Image