## EXPERIMENT NO. 3 A

- **Aim**: Implementation of Support Vector Machines (SVM) for classifying images of handwritten digits into their respective numerical classes (0 to 9).

- **Hardware Requirement:**

▪ 6 GB free disk space.
▪ 2 GB RAM.
▪ 2 GB of RAM, plus additional RAM for virtual machines.
▪ 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
▪ Virtualization is available with the KVM hypervisor
▪ Intel 64 and AMD64 architectures

- **Software Requirement:**
  Jypiter Nootbook/Ubuntu

- **Theory:**

Classification Analysis: Definition

This analysis is a data mining technique used to determine the structure and categories within a given dataset. Classification analysis is commonly used in machine learning, text analytics, and statistical modelling. Above all, it can help identify patterns or groupings between individual observations, enabling researchers to understand their datasets better and make more accurate predictions.

Classification analysis is used to group or classify objects according to shared characteristics. Moreover, this analysis can be used in many applications, from segmenting customers for marketing campaigns to forecasting stock market trends.

Classification Analysis Example

- Classifying images

One example of a classification analysis is the use of supervised learning algorithms to classify images. In this case, the algorithm is provided with an image dataset (the training set) that contains labelled images.
The algorithm uses labels to learn how to distinguish between different types of objects in the picture. Once trained, it can then be used to classify new images as belonging to one category or another.

- Customer Segmentation

Another example of classification analysis would be customer segmentation for marketing campaigns. Classification algorithms group customers into segments based on their characteristics and behaviours.
This helps marketers target specific groups with tailored content, offers, and promotions that are more likely to appeal to them.

- Stock Market Prediction

Finally, classification analysis can also be used for stock market prediction. Classification algorithms can identify patterns between past stock prices and other economic indicators, such as interest rates or unemployment figures. By understanding these correlations, analysts can better predict future market trends and make more informed investment decisions.

These are just some examples of how classification analysis can be applied to various scenarios. Unquestionably, classification algorithms can be used to analyse datasets in any domain, from healthcare and finance to agriculture and logistics.

Classification Analysis Techniques

This analysis is a powerful technique used in data science to analyse and categorise data. Classification techniques are used in many areas, from predicting customer behaviours to finding patterns and trends in large datasets.
This analysis can help businesses make informed decisions about marketing strategies, product development, and more. So, let’s delve into the various techniques

1. Supervised Learning

Supervised learning algorithms require labelled data. This means the algorithm is provided with a dataset that has already been categorised or labelled with class labels. The algorithm then uses this label to learn how to distinguish between different class objects in the data. Once trained, it can use its predictive power to classify new datasets.

2. Unsupervised Learning

Unsupervised learning algorithms do not require labelled data. Instead, they use clustering and dimensionality reduction techniques to identify patterns in the dataset without any external guidance. These algorithms help segment customers or identify outlier items in a dataset.

3. Deep Learning

Deep learning is a subset/division of machine learning technologies that use artificial neural networks. These algorithms are capable of learning from large datasets and making complex decisions. Deep learning can be used for tasks such as image classification, natural language processing, and predictive analytics.
Classification algorithms can help uncover patterns in the data that could not be detected using traditional methods. By using classification analysis, businesses can gain valuable insights into their customers’ behaviours and preferences, helping them make more informed decisions.

**Implementation:**

```
# Import Libraries

import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

**Handwritten Digit Recognition**

Use the sklearn.dataset load_digits() method. It loads the handwritten digits dataset. The returned data is in the form of a Dictionary. The 'data' attribute contains a flattenned array of 64 (each digit image is of 8*8 pixels) elements representing the digits.

The 'target' attribute is the 'class' of Digit (0-9) Each individual digit is represented through a flattendded 64 digit array numbers of Greyscale values. There are 1797 samples in total and each class or digit has roughly 180 samples.

```
from sklearn.datasets import load_digits
digits = load_digits(n_class=10)

digits

{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  2., ..., 12.,  0.,  0.],
        [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
'target': array([0, 1, 2, ..., 8, 9, 8]),
'frame': None,
'feature_names': ['pixel_0_0',
 'pixel_0_1',
 'pixel_0_2',
 'pixel_0_3',
 'pixel_0_4',
 'pixel_0_5',
 'pixel_0_6',
 'pixel_0_7',
 'pixel_1_0',
 'pixel_1_1',
 'pixel_1_2',
 'pixel_1_3',
 'pixel_1_4',
 'pixel_1_5',
 'pixel_1_6',
 'pixel_1_7',
 'pixel_2_0',
 'pixel_2_1',
 'pixel_2_2',
 'pixel_2_3',
 'pixel_2_4',
 'pixel_2_5',
 'pixel_2_6',
 'pixel_2_7',
 'pixel_3_0',
 'pixel_3_1',
 'pixel_3_2',
 'pixel_3_3',
 'pixel_3_4',
 'pixel_3_5',
 'pixel_3_6',
 'pixel_3_7',
 'pixel_4_0',
 'pixel_4_1',
 'pixel_4_2',
 'pixel_4_3',
 'pixel_4_4',
 'pixel_4_5',
 'pixel_4_6',
 'pixel_4_7',
 'pixel_5_0',
 'pixel_5_1',
 'pixel_5_2',
 'pixel_5_3',
 'pixel_5_4',
 'pixel_5_5',
 'pixel_5_6',
 'pixel_5_7',
 'pixel_6_0',
 'pixel_6_1',
 'pixel_6_2',
 'pixel_6_3',
 'pixel_6_4',
 'pixel_6_5',
 'pixel_6_6',
 'pixel_6_7',
 'pixel_7_0',
 'pixel_7_1',
 'pixel_7_2',
 'pixel_7_3',
 'pixel_7_4',
 'pixel_7_5',
 'pixel_7_6',
 'pixel_7_7'],

'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
         [ 0.,  0., 13., ..., 15.,  5.,  0.],
         [ 0.,  3., 15., ..., 11.,  8.,  0.],
         ...,
         [ 0.,  4., 11., ..., 12.,  7.,  0.],
         [ 0.,  2., 14., ..., 12.,  0.,  0.],
        [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

       [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
        [ 0.,  0.,  0., ...,  9.,  0.,  0.],
        [ 0.,  0.,  3., ...,  6.,  0.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

       [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
        [ 0.,  0.,  3., ..., 14.,  0.,  0.],
        [ 0.,  0.,  8., ..., 16.,  0.,  0.],
        ...,
        [ 0.,  9., 16., ...,  0.,  0.,  0.],
        [ 0.,  3., 13., ..., 11.,  5.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

       ...,

       [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
        [ 0.,  0., 13., ...,  2.,  1.,  0.],
        [ 0.,  0., 16., ..., 16.,  5.,  0.],
        ...,
        [ 0.,  0., 16., ..., 15.,  0.,  0.],
        [ 0.,  0., 15., ..., 16.,  0.,  0.],
        [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

       [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
        [ 0.,  0., 14., ..., 15.,  1.,  0.],
        [ 0.,  4., 16., ..., 16.,  7.,  0.],
        ...,
        [ 0.,  0.,  0., ..., 16.,  2.,  0.],
        [ 0.,  0.,  4., ..., 16.,  2.,  0.],
        [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

       [[ 0.,  0., 10., ...,  1.,  0.,  0.],
        [ 0.,  2., 16., ...,  1.,  0.,  0.],
        [ 0.,  0., 15., ..., 15.,  0.,  0.],
        ...,
        [ 0.,  4., 16., ..., 16.,  6.,  0.],
        [ 0.,  8., 16., ..., 16.,  8.,  0.],
        [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),
 'DESCR': ".. _digits_dataset:\n\nOptical recognition of handwritten digits dataset\n--------------------------------------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 1797\n    :Number of Attributes: 64\n    :Attribute Information: 8x8 image of integer pixels in the range 0..16.\n    :Missing Attribute Values: None\n    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n    :Date: July; 1998\n\nThis is a copy of the test set of the UCI ML hand-written digits
```

```
datasets\nhttps://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits\n\nThe data set contains images of hand-written digits: 10 classes where\neach class refers to a digit.\n\nPreprocessing programs made available by NIST were used to extract\nnormalized bitmaps of handwritten digits from a preprinted form. From a\ntotal of 43 people, 30 contributed to the training set and different 13\nto the test set. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x4 and the number of on pixels are counted in each block. This generates\nan input matrix of 8x8 where each element is an integer in the range\n0..16. This reduces dimensionality and gives invariance to small\ndistortions.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,\n1994.\n\n.. topic:: References\n\n  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their\n    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of\n    Graduate Studies in Science and Engineering, Bogazici University.\n  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.\n Linear dimensionalityreduction using relevance weighted LDA. School of\n    Electrical and Electronic Engineering Nanyang Technological University.\n    2005.\n  - Claudio Gentile. A New Approximate Maximal Margin Classification\n    Algorithm. NIPS. 2000.\n"}
```

```
digits['data'][0].reshape(8,8)
```

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

```
digits['data'][0]
```

```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

```
digits['images'][1]
```

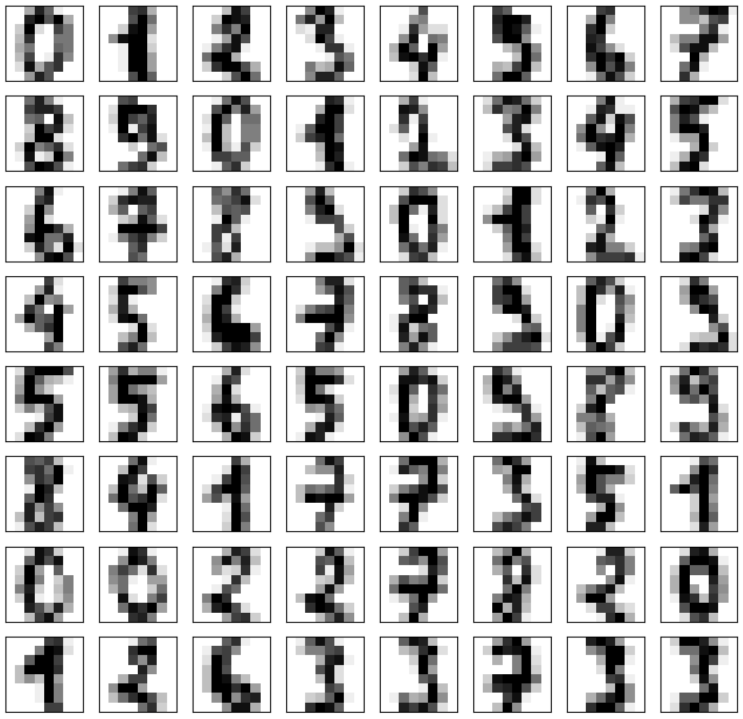
```
array([[ 0.,  0.,  0., 12., 13.,  5.,  0.,  0.],
       [ 0.,  0.,  0., 11., 16.,  9.,  0.,  0.],
       [ 0.,  0.,  3., 15., 16.,  6.,  0.,  0.],
       [ 0.,  7., 15., 16., 16.,  2.,  0.,  0.],
       [ 0.,  0.,  1., 16., 16.,  3.,  0.,  0.],
       [ 0.,  0.,  1., 16., 16.,  6.,  0.,  0.],
       [ 0.,  0.,  1., 16., 16.,  6.,  0.,  0.],
       [ 0.,  0.,  0., 11., 16., 10.,  0.,  0.]])
```

```
digits['target'][0:9]
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
digits['target'][0]

0

digits.images[0]

array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

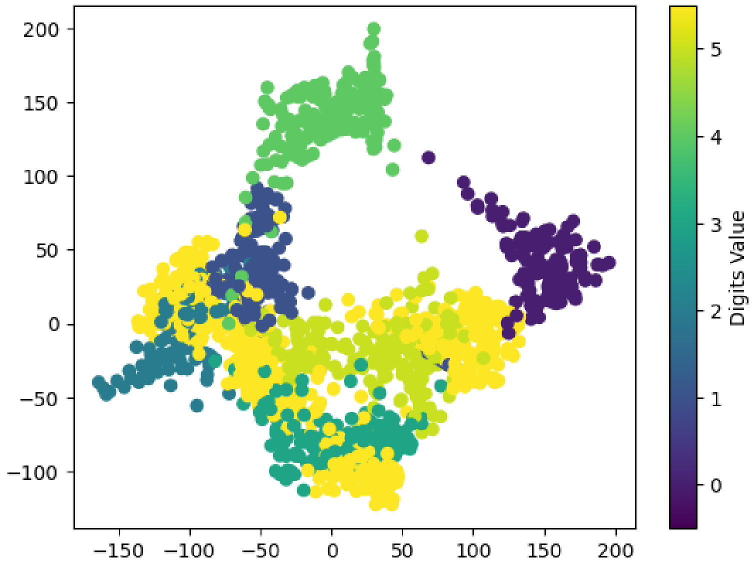
```
# Each Digit is represented in digits.images as a matrix of 8x8 = 64 pixels. Each of the 64 values represent
# a greyscale. The Greyscale are then plotted in the right scale by the imshow method.

fig, ax = plt.subplots(8,8, figsize=(10,10))
for i, axi in enumerate(ax.flat):
    axi.imshow(digits.images[i], cmap='binary')
    axi.set(xticks=[], yticks=[])
```



```
# Plotting - Clustering the data points after using Manifold Learning

from sklearn.manifold import Isomap

iso = Isomap(n_components=2)
```

```
projection = iso.fit_transform(digits.data)     # digits.data - 64 dimensions to 2 dimensions

plt.scatter(projection[:, 0], projection[:, 1], c=digits.target, cmap="viridis")

plt.colorbar(ticks=range(10), label='Digits Value')
plt.clim(-0.5, 5.5)
```

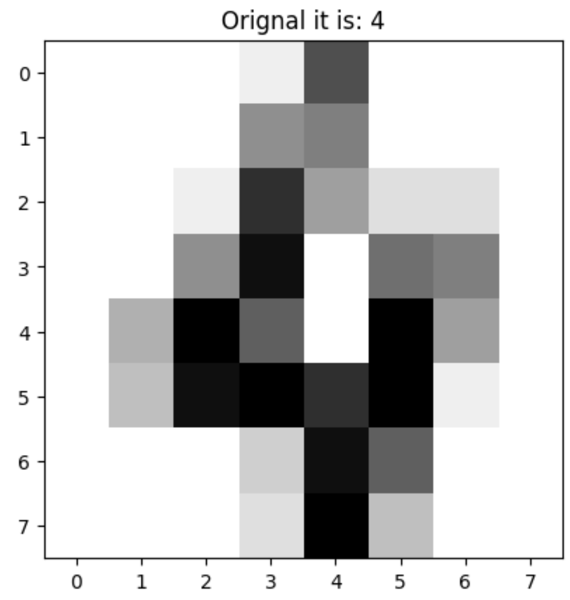
```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_isomap.py:373: UserWarning: The number of connected components of the neighbors graph is 2 > 1. Completing the graph to fit Isomap might be slow. Increase the number of neighbors to avoid this issue.
  self._fit_transform(X)
/usr/local/lib/python3.10/dist-packages/scipy/sparse/_index.py:103: SparseEfficiencyWarning: Changing the sparsity structure of a csr_matrix is expensive. lil_matrix is more efficient.
  self._set_intXint(row, col, x.flat[0])
```



```
print(projection[:, 0][70], projection[:, 1][70])
```

```
-56.60683580684862 61.95022367117501
```

```
def view_digit(index):
    plt.imshow(digits.images[index] , cmap = plt.cm.gray_r)
    plt.title('Orignal it is: '+ str(digits.target[index]))
    plt.show()

view_digit(4)
```



## Use the Support Vector Machine Classifier to train the Data

Use part of the data for train and part of the data for test (predicion)

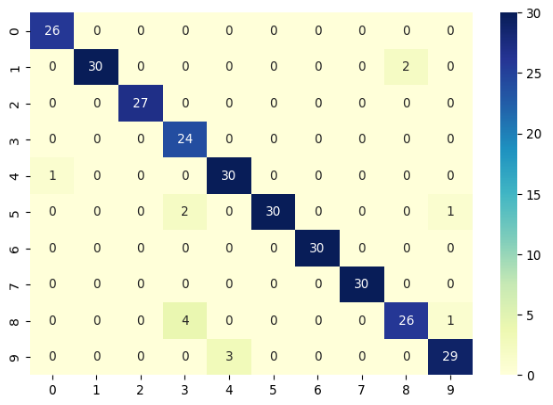
```
main_data = digits['data']
targets = digits['target']

from sklearn import svm

svc = svm.SVC(gamma=0.001 , C = 100)

# GAMMA is a parameter for non linear hyperplanes.
# The higher the gamma value it tries to exactly fit the training data set
# C is the penalty parameter of the error term.
# It controls the trade off between smooth decision boundary and classifying the training points correctly.

svc.fit(main_data[:1500] , targets[:1500])

predictions = svc.predict(main_data[1501:])

list(zip(predictions , targets[1501:]))
```

```
[(7, 7),        (8, 8),        (6, 6),
 (4, 4),        (4, 4),        (1, 1),
 (6, 6),        (3, 3),        (7, 7),
 (3, 3),        (1, 1),        (5, 5),
 (1, 1),        (4, 4),        (4, 4),
 (3, 3),        (0, 0),        (4, 4),
 (9, 9),        (5, 5),        (7, 7),
 (1, 1),        (3, 3),        (2, 2),
 (7, 7),        (6, 6),        (8, 8),
 (6, 6),        (9, 9),        (2, 2),
(2, 2),        (0, 0),        (7, 7),
(5, 5),        (9, 9),        (9, 4),
(7, 7),        (8, 8),        (6, 6),
(9, 9),        (9, 9),        (3, 3),
(5, 5),        (8, 8),        (1, 1),
(4, 4),        (4, 4),        (3, 3),
(8, 8),        (1, 1),        (9, 9),
(8, 8),        (7, 7),        (1, 1),
(4, 4),        (7, 7),        (7, 7),
(9, 9),        (3, 3),        (6, 6),
(0, 0),        (5, 5),        (8, 8),
(8, 8),        (1, 1),        (4, 4),
(9, 9),        (0, 0),        (3, 3),
(8, 8),        (0, 0),        (1, 1),
(0, 0),        (2, 2),        (4, 4),
(1, 1),        (2, 2),        (0, 0),
(2, 2),        (7, 7),        (5, 5),
(3, 3),        (8, 8),        (3, 3),
(4, 4),        (2, 2),        (6, 6),
(5, 5),        (0, 0),        (9, 9),
(6, 6),        (1, 1),        (6, 6),
(7, 7),        (2, 2),        (1, 1),
(1, 8),        (6, 6),        (7, 7),
(9, 9),        (8, 3),        (5, 5),
(0, 0),        (3, 3),        (4, 4),
(1, 1),        (7, 7),        (4, 4),
(2, 2),        (3, 3),        (7, 7),
(3, 3),        (3, 3),        (2, 2),
(4, 4),        (4, 4),        (2, 2),
(5, 5),        (6, 6),        (5, 5),
(6, 6),        (6, 6),        (7, 7),
(9, 9),        (6, 6),        (8, 9),
(0, 0),        (9, 4),        (5, 5),
(1, 1),        (9, 9),        (9, 4),
(2, 2),        (1, 1),        (4, 4),
(3, 3),        (5, 5),        (5, 9),
(4, 4),        (0, 0),        (0, 0),
(5, 5),        (9, 9),        (8, 8),
(6, 6),        (5, 5),        (9, 9),
(7, 7),        (2, 2),        (8, 8),
(1, 8),        (8, 8),        (0, 0),
(9, 9),        (0, 0),        (1, 1),
(4, 0),        (1, 1),        (2, 2),
(9, 9),        (7, 7),        (3, 3),
(5, 5),        (6, 6),        (4, 4),
(5, 5),        (3, 3),        (5, 5),
(6, 6),        (2, 2),        (6, 6),
(5, 5),        (1, 1),        (7, 7),
(8, 8),
(9, 9),
(0, 0),
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5),
(6, 6),
(7, 7),
(8, 8),
(9, 9),
(0, 0),
(1, 1),
(2, 2),
(8, 3),
(4, 4),
(5, 5),
(6, 6),
(7, 7),
(8, 8),
(9, 9),
(0, 0),
(9, 9),
(5, 5),
(5, 5),
(6, 6),
(5, 5),
(0, 0),
(9, 9),
(8, 8),
(9, 9),
(8, 8),
(4, 4),
(1, 1),
(7, 7),
(7, 7),
(3, 3),
(5, 5),
(1, 1),
(0, 0),
(0, 0),
(2, 2),
(2, 2),
(7, 7),
(8, 8),
(2, 2),
(0, 0),
(1, 1),
(2, 2),
(6, 6),
(8, 3),
(8, 3),
(7, 7),
(5, 3),
(3, 3),
(4, 4),
(6, 6),
(6, 6),
(6, 6),
(4, 4),
(9, 9),
(1, 1),
(5, 5),
(0, 0),
(9, 9),
(5, 5),
(2, 2),
(8, 8),
(2, 2),
(0, 0),
(0, 0),
(1, 1),
(7, 7),
(6, 6),
(3, 3),
(2, 2),
(1, 1),
(7, 7),
(4, 4),
(6, 6),
(3, 3),
(1, 1),
(3, 3),
(9, 9),
(1, 1),
(7, 7),
(6, 6),
(8, 8),
(4, 4),
(5, 3),
(1, 1),
(4, 4),
(0, 0),
(5, 5),
(3, 3),
(6, 6),
(9, 9),
(6, 6),
(1, 1),
(7, 7),
(5, 5),
(4, 4),
(4, 4),
(7, 7),
(2, 2),
(8, 8),
(2, 2),
(2, 2),
(5, 5),
(7, 7),
(9, 9),
(5, 5),
(4, 4),
(8, 8),
(8, 8),
(4, 4),
(9, 9),
(0, 0),
(8, 8),
(9, 9),
(8, 8)]
```

**Create the Confusion Matric for Performance Evaluation**

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(predictions, targets[1501:])

conf_matrix = pd.DataFrame(data = cm)
```

```
plt.figure(figsize = (8,5))

sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu");
```



```
cm

array([[26,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 30,  0,  0,  0,  0,  0,  0,  2,  0],
       [ 0,  0, 27,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 24,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0, 30,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  2,  0, 30,  0,  0,  0,  1],
       [ 0,  0,  0,  0,  0,  0, 30,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 30,  0,  0],
       [ 0,  0,  0,  4,  0,  0,  0,  0, 26,  1],
       [ 0,  0,  0,  0,  3,  0,  0,  0,  0, 29]])
```

**Print the Classification Report**

```
from sklearn.metrics import classification_report

print(classification_report(predictions, targets[1501:]))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        26
           1       1.00      0.94      0.97        32
           2       1.00      1.00      1.00        27
           3       0.80      1.00      0.89        24
           4       0.91      0.97      0.94        31
           5       1.00      0.91      0.95        33
           6       1.00      1.00      1.00        30
           7       1.00      1.00      1.00        30
           8       0.93      0.84      0.88        31
           9       0.94      0.91      0.92        32

    accuracy                           0.95       296
```