Handwritten Digit Classification using Logistic Regression

Project Overview

This project involves building a classification model to recognize handwritten digits using the load_digits dataset from Scikit-Learn. The dataset comprises 8x8 pixel grayscale images of digits (0-9) and is commonly used as a benchmark in machine learning. The aim of this project is to implement a logistic regression model, evaluate its performance, and visualize the results, providing insights into the classification process.

Dataset Source: The dataset is loaded using the load_digits function from Scikit-Learn's datasets module.

Features: The dataset contains 64 features (pixel values) representing each image of the digit.

Target: The target variable consists of the corresponding digit labels (0-9) for each image.

1. Import Necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_digits
from sklearn import metrics
```

2. Load and Explore the Dataset

This loads the digits dataset from scikit-learn, which contains images of handwritten digits (0-9). Each image is represented as an 8x8 pixel array.

```
digits = load_digits()
digits
```

```
       [ 0.,  0.,  8., ..., 10.,  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.]],
```

digits.keys() - Used to Check the Key value in dataset

```
digits.keys()
```

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

Change to DataFrame

```
data = pd.DataFrame(digits.data,columns = digits.feature_names)
```

```
data['target'] = digits.target
```

```
data.head()
```

	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_6_7	pixel_7_
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.

5 rows × 65 columns

3. Data Preprocessing

Clean the dataset (handle missing values, remove outliers, etc.). For instance, if any columns have missing values, you can either fill them or drop those rows.

```
data.isna().sum()
```

	0
pixel_0_0	0
pixel_0_1	0
pixel_0_2	0
pixel_0_3	0
pixel_0_4	0
...	...
pixel_7_4	0
pixel_7_5	0
pixel_7_6	0
pixel_7_7	0
target	0

65 rows × 1 columns

dtype: int64

```
data.dtypes
```

	0
pixel_0_0	float64
pixel_0_1	float64
pixel_0_2	float64
pixel_0_3	float64
pixel_0_4	float64
...	...
pixel_7_4	float64
pixel_7_5	float64
pixel_7_6	float64
pixel_7_7	float64
target	int64

65 rows × 1 columns

dtype: object

```
data.info()
```

```
 5   pixel_0_5  1797 non-null   float64
 6   pixel_0_6  1797 non-null   float64
 7   pixel_0_7  1797 non-null   float64
 8   pixel_1_0  1797 non-null   float64
 9   pixel_1_1  1797 non-null   float64
 10  pixel_1_2  1797 non-null   float64
 11  pixel_1_3  1797 non-null   float64
 12  pixel_1_4  1797 non-null   float64
 13  pixel_1_5  1797 non-null   float64
 14  pixel_1_6  1797 non-null   float64
 15  pixel_1_7  1797 non-null   float64
 16  pixel_2_0  1797 non-null   float64
 17  pixel_2_1  1797 non-null   float64
 18  pixel_2_2  1797 non-null   float64
 19  pixel_2_3  1797 non-null   float64
```

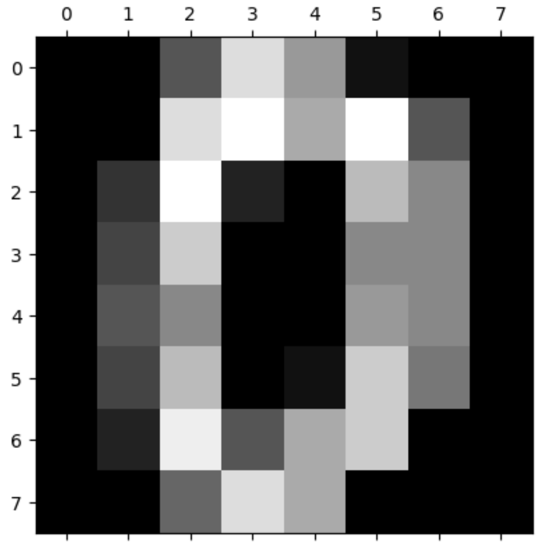
```
 31  pixel_3_7  1797 non-null   float64
 32  pixel_4_0  1797 non-null   float64
 33  pixel_4_1  1797 non-null   float64
 34  pixel_4_2  1797 non-null   float64
 35  pixel_4_3  1797 non-null   float64
 36  pixel_4_4  1797 non-null   float64
 37  pixel_4_5  1797 non-null   float64
 38  pixel_4_6  1797 non-null   float64
 39  pixel_4_7  1797 non-null   float64
 40  pixel_5_0  1797 non-null   float64
 41  pixel_5_1  1797 non-null   float64
 42  pixel_5_2  1797 non-null   float64
 43  pixel_5_3  1797 non-null   float64
 44  pixel_5_4  1797 non-null   float64
 45  pixel_5_5  1797 non-null   float64
 46  pixel_5_6  1797 non-null   float64
 47  pixel_5_7  1797 non-null   float64
 48  pixel_6_0  1797 non-null   float64
 49  pixel_6_1  1797 non-null   float64
 50  pixel_6_2  1797 non-null   float64
 51  pixel_6_3  1797 non-null   float64
 52  pixel_6_4  1797 non-null   float64
 53  pixel_6_5  1797 non-null   float64
 54  pixel_6_6  1797 non-null   float64
 55  pixel_6_7  1797 non-null   float64
 56  pixel_7_0  1797 non-null   float64
 57  pixel_7_1  1797 non-null   float64
 58  pixel_7_2  1797 non-null   float64
 59  pixel_7_3  1797 non-null   float64
 60  pixel_7_4  1797 non-null   float64
 61  pixel_7_5  1797 non-null   float64
 62  pixel_7_6  1797 non-null   float64
 63  pixel_7_7  1797 non-null   float64
```

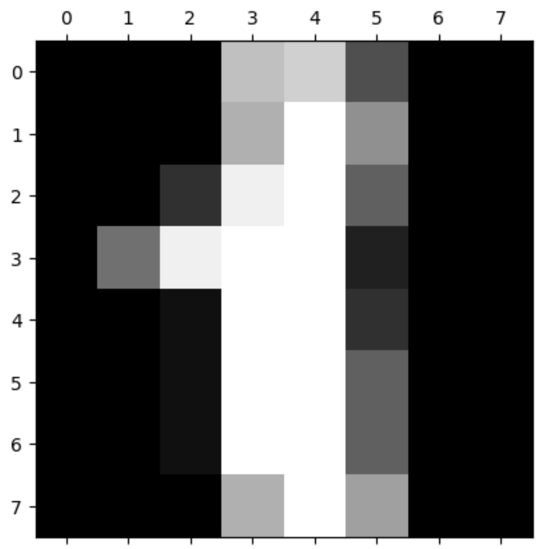
4. Visualizing the Data

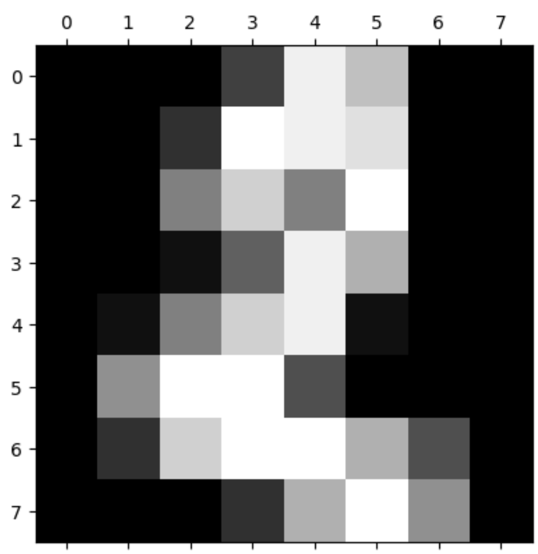
In this step, we visualize the data to get an initial understanding of how the features are distributed

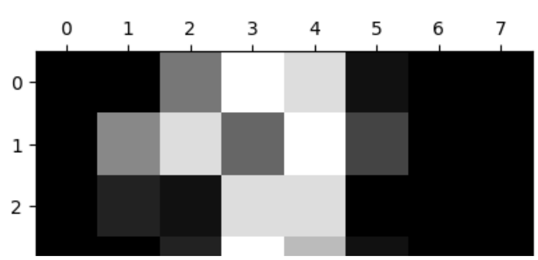
```
plt.figure(figsize=(20,4))
plt.gray()
for i in range(5):
  plt.matshow(digits.images[i])
```

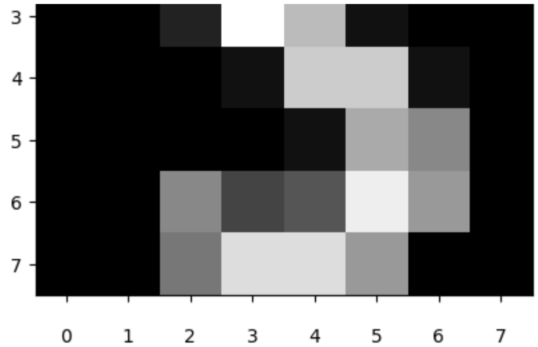
<Figure size 2000x400 with 0 Axes>

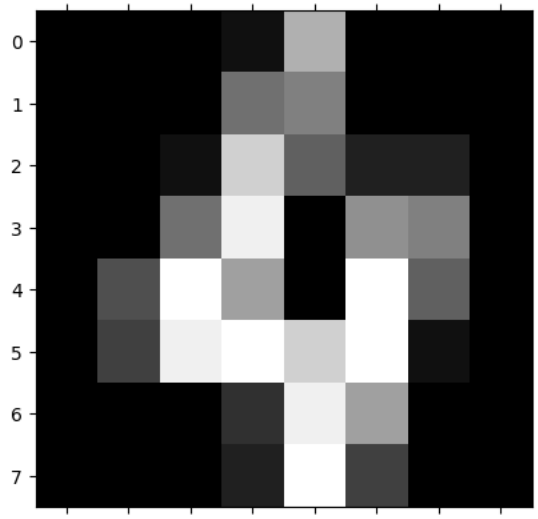












5. Feature Selection

Choose relevant features (independent variables) to predict house prices (target variable).

```
x = data.drop(columns = 'target', axis =1)
y = data['target']
```

6. Train-Test Split

Split the data into training and testing sets to evaluate the model's performance.

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .2, random_state =1)
```

Suggested code may be subject to a licence | standbyme/gender-name-by-ML

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(1437, 64)
(1437,)
(360, 64)
(360,)
```

7. Train the Linear Regression Model

Fit the model on the training data.

```
model = LogisticRegression()
model.fit(x_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (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(
```

```
▾ LogisticRegression
LogisticRegression()
```

8. Make Predictions

Predict Digits data on the test data.

```
y_pred = model.predict(x_test)
```

9. Evaluate the Model

Evaluate the performance of the model using metrics like Mean Squared Error (MSE) and R^2 score.

```
model.score(x_test,y_test)
```

```
0.9694444444444444
```

```
cm = metrics.confusion_matrix(y_test,y_pred)
```

```
plt.figure(figsize=(10,10))
sns.heatmap(cm,annot=True)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show
```