Aim: Develop and evaluate logistic regression models for multi-class classification tasks using machine learning.

Software Used: Google Collaborative Python

Theory:

Logistic Regression

Logistic Regression is a supervised machine learning algorithm used for classification problems. Unlike linear regression which predicts continuous values it predicts the probability that an input belongs to a specific class. It is used for binary classification where the output can be one of two possible categories such as Yes/No, True/False or 0/1. It uses sigmoid function to convert inputs into a probability value between 0 and 1. In this article, we will see the basics of logistic regression and its core concepts.

Logistic Regression for Multi-Class Classification is an extension of binary logistic regression used when the target variable contains more than two classes. Instead of predicting a probability for just two classes, the model estimates probabilities for all possible classes and assigns the one with the highest probability.

Concept:

- Unlike binary classification, where the output is either 0 or 1, multi-class classification deals with multiple categories (e.g., predicting if a fruit is apple, banana, or orange).
- The algorithm uses a generalized form of logistic regression to handle more than two outcomes.

Approaches:

- One-vs-Rest (OvR): Trains one binary classifier per class, treating that class as "positive" and all others as "negative".
- Multinomial Logistic Regression: Directly models all classes simultaneously using the softmax function.

Mathematical Model:

- For K classes, the probability that an observation belongs to class jjj is calculated as:

$$P(y = j|x) = \frac{e^{\beta_j \cdot x}}{\sum_{k=1}^{K} e^{\beta_k \cdot x}}$$

- Here, the softmax function ensures all probabilities sum to 1.

Model Training:

- The model parameters are estimated using Maximum Likelihood Estimation (MLE).
- Optimization algorithms like Gradient Descent are used to minimize the loss function (often Cross-Entropy Loss).

Advantages:

- Works well for linearly separable classes.
- Produces probabilistic outputs, which can be useful in decision-making.

Limitations:

- Assumes linear decision boundaries between classes.
- May underperform if classes are highly imbalanced or non-linear in nature.

Evaluation Metrics:

- Accuracy – percentage of correctly classified samples.
- Precision, Recall, F1-score – calculated for each class.
- Confusion Matrix – for detailed class-wise performance.

Implementation in Python:

- Libraries like Scikit-learn provide built-in support for multi-class logistic regression.
- Common parameters include multi_class='multinomial'.

Output:

1)Digit Dataset

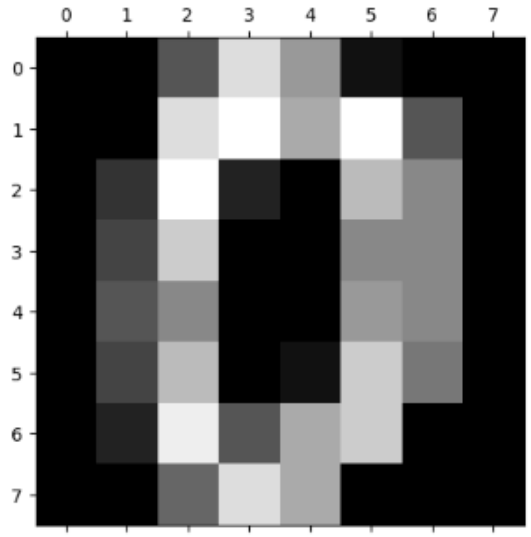
```
[1]  from sklearn.datasets import load_digits
     %matplotlib inline
     import matplotlib.pyplot as plt
     digits = load_digits()
```

```
import numpy as np

# The array of digit labels you are interested in
input_labels = np.array([0, 1, 2, 3, 4])
corresponding_indices = [np.where(digits.target == label)[0][0] for label in input_labels]

# Get the corresponding data from digits.data using these indices
corresponding_data = digits.data[corresponding_indices]

print(f"The pixel data for the digits {input_labels} are:")
print(corresponding_data)

# You can also see the shape of the retrieved data
print(f"\nShape of the retrieved data: {corresponding_data.shape}")
```

```
The pixel data for the digits [0 1 2 3 4] are:
[[ 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.]
 [ 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.]
 [ 0.  0.  0.  4. 15. 12.  0.  0.  0.  0.  3. 16. 15. 14.  0.  0.  0.  0.
   8. 13.  8. 16.  0.  0.  0.  0.  1.  6. 15. 11.  0.  0.  0.  1.  8. 13.
  15.  1.  0.  0.  0.  9. 16. 16.  5.  0.  0.  0.  0.  3. 13. 16. 16. 11.
   5.  0.  0.  0.  0.  3. 11. 16.  9.  0.]
 [ 0.  0.  7. 15. 13.  1.  0.  0.  0.  8. 13.  6. 15.  4.  0.  0.  0.  2.
   1. 13. 13.  0.  0.  0.  0.  0.  2. 15. 11.  1.  0.  0.  0.  0.  0.  1.
  12. 12.  1.  0.  0.  0.  0.  0.  1. 10.  8.  0.  0.  0.  8.  4.  5. 14.
   9.  0.  0.  0.  7. 13. 13.  9.  0.  0.]
 [ 0.  0.  0.  1. 11.  0.  0.  0.  0.  0.  0.  7.  8.  0.  0.  0.  0.  0.
   1. 13.  6.  2.  2.  0.  0.  0.  7. 15.  0.  9.  8.  0.  0.  5. 16. 10.
   0. 16.  6.  0.  0.  4. 15. 16. 13. 16.  1.  0.  0.  0.  0.  3. 15. 10.
   0.  0.  0.  0.  0.  2. 16.  4.  0.  0.]]

Shape of the retrieved data: (5, 64)
```

```
plt.gray()
#for i in range(5):
plt.matshow(digits.images[0])
```

```
<matplotlib.image.AxesImage at 0x7911e914fb90>
<Figure size 640x480 with 0 Axes>
```



[3]
```
dir(digits)
```

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

[4]
```
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.])
```

[6]
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=2000)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(digits.data,digits.target, test_size=0.2)
model.fit(X_train, y_train)
```

```
LogisticRegression(max_iter=2000)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

```
[7] model.score(X_test, y_test)
```

```
0.9666666666666667
```

```
[8] model.predict(digits.data[0:5])
```

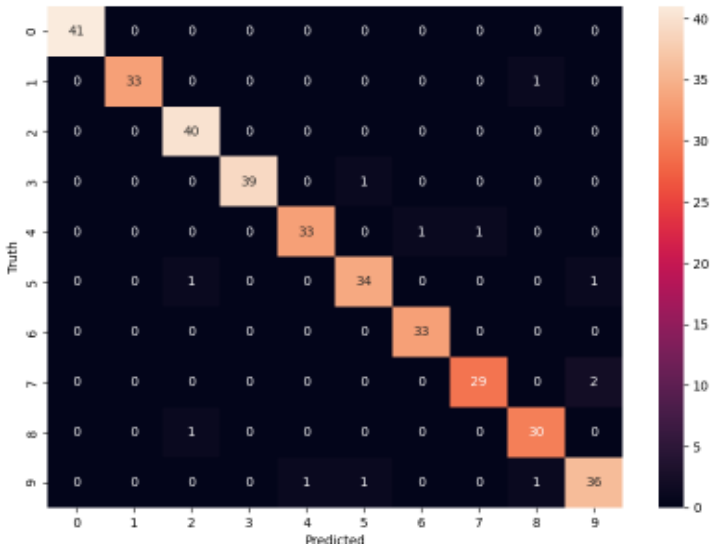
```
array([0, 1, 2, 3, 4])
```

```
[10] y_predicted = model.predict(X_test)
     from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, y_predicted)
     cm
```

```
array([[41,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 33,  0,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0, 40,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 39,  0,  1,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 33,  0,  1,  1,  0,  0],
       [ 0,  0,  1,  0,  0, 34,  0,  0,  0,  1],
       [ 0,  0,  0,  0,  0,  0, 33,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 29,  0,  2],
       [ 0,  0,  1,  0,  0,  0,  0,  0, 30,  0],
       [ 0,  0,  0,  0,  1,  1,  0,  0,  1, 36]])
```

```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(95.72222222222221, 0.5, 'Truth')
```




```
[20] # Display the head of the DataFrame showing Actual and Predicted values
     display(results_df.head())
```

	Actual	Predicted
0	6	6
1	6	6
2	7	7
3	3	3
4	6	6

2)Iris

```
[7] from sklearn.datasets import load_iris
    iris = load_iris()
    display(iris_df.head())
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
[6] # Check for null values in the DataFrame
    print("Null values per column:")
    display(iris_df.isnull().sum())
```

```
Null values per column:
```

	0
sepal length (cm)	0
sepal width (cm)	0
petal length (cm)	0
petal width (cm)	0
target	0

dtype: int64

```
[4] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=42)
```

```
[5] from sklearn.linear_model import LogisticRegression
    model = LogisticRegression()
    model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

```
[6] from sklearn.metrics import accuracy_score

    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

```
[12] model.score(X_test, y_test)
```

```
1.0
```

```
[24] pred = model.predict(X_test)
     print("Predictions (0: Setosa, 1: Versicolour, 2: Virginica):")
     display(pred)
```

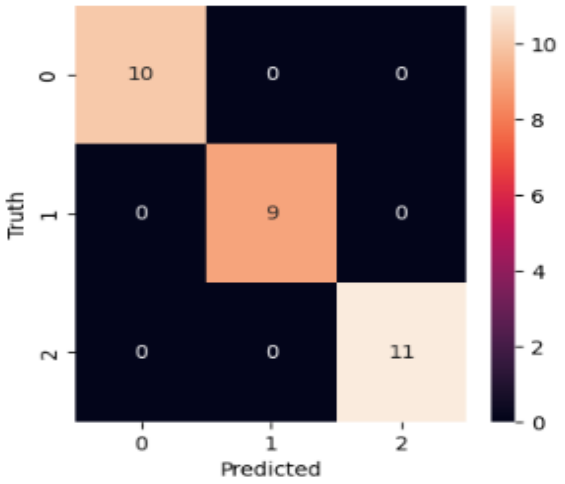
```
Predictions (0: Setosa, 1: Versicolour, 2: Virginica):
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[10,  0,  0],
       [ 0,  9,  0],
       [ 0,  0, 11]])
```

```
[18] import seaborn as sn
     import matplotlib.pyplot as plt
     plt.figure(figsize = (4,4))
     sn.heatmap(cm, annot=True)
     plt.xlabel('Predicted')
     plt.ylabel('Truth')
```

```
Text(20.72222222222222, 0.5, 'Truth')
```



```
[23] import pandas as pd
     feature_names = iris.feature_names
     test_results_df = pd.DataFrame(X_test, columns=feature_names)
     test_results_df['Predicted Flower'] = predicted_names
     display(test_results_df)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Predicted Flower
0	6.1	2.8	4.7	1.2	versicolor
1	5.7	3.8	1.7	0.3	setosa
2	7.7	2.6	6.9	2.3	virginica
3	6.0	2.9	4.5	1.5	versicolor
4	6.8	2.8	4.8	1.4	versicolor
5	5.4	3.4	1.5	0.4	setosa
6	5.6	2.9	3.6	1.3	versicolor
7	6.9	3.1	5.1	2.3	virginica
8	6.2	2.2	4.5	1.5	versicolor
9	5.8	2.7	3.9	1.2	versicolor
10	6.5	3.2	5.1	2.0	virginica
11	4.8	3.0	1.4	0.1	setosa
12	5.5	3.5	1.3	0.2	setosa
13	4.9	3.1	1.5	0.1	setosa
14	5.1	3.8	1.5	0.3	setosa
15	6.3	3.3	4.7	1.6	versicolor
16	6.5	3.0	5.8	2.2	virginica
17	5.6	2.5	3.9	1.1	versicolor
18	5.7	2.8	4.5	1.3	versicolor
19	6.4	2.8	5.6	2.2	virginica
20	4.7	3.2	1.6	0.2	setosa
21	6.1	3.0	4.9	1.8	virginica
22	5.0	3.4	1.6	0.4	setosa
23	6.4	2.8	5.6	2.1	virginica
24	7.9	3.8	6.4	2.0	virginica
25	6.7	3.0	5.2	2.3	virginica
26	6.7	2.5	5.8	1.8	virginica
27	6.8	3.2	5.9	2.3	virginica
28	4.8	3.0	1.4	0.3	setosa
29	4.8	3.1	1.6	0.2	setosa

```
[18] # Get input from the user
     sepal_length = float(input("Enter sepal length (cm): "))
     sepal_width = float(input("Enter sepal width (cm): "))
     petal_length = float(input("Enter petal length (cm): "))
     petal_width = float(input("Enter petal width (cm): "))

     # Create a numpy array with the input values
     # The model expects input in the same format as the training data (a 2D array)
     import numpy as np
     new_data = np.array([[sepal_length, sepal_width, petal_length, petal_width]])

     # Predict the flower species
     predicted_species_index = model.predict(new_data)[0]

     # Get the predicted flower name
     predicted_flower_name = iris.target_names[predicted_species_index]

     print(f"\nBased on the input values, the predicted flower species is: {predicted_flower_name}")
```

```
Enter sepal length (cm): 6.1
Enter sepal width (cm): 2.8
Enter petal length (cm): 4.7
Enter petal width (cm): 1.2

Based on the input values, the predicted flower species is: versicolor
```

Conclusion: Hence, through this experiment, we learned how to develop and evaluate Logistic Regression models for multi-class classification tasks, train them using approaches like One-vs-Rest and Multinomial Logistic Regression, and assess performance. This process is essential for building reliable multi-class models that can accurately classify data into multiple categories.