

## Shabdh Dataset

### Performing Multiclass Logistic Regression

Introduction: When outcome has more than to categories, Multi class regression is used for classification. For e.g. mail classification as primary, social, promotions, forums

Multiclass logistic regression is an extension of binary logistic regression that allows for the classification of instances into more than two classes. This method is particularly useful when dealing with problems where the output can belong to one of three or more discrete classes.

Key Concepts: Logistic Regression: Originally designed for binary classification, logistic regression estimates the probability that a given input belongs to a certain class. It uses the logistic (sigmoid) function to map predicted values to probabilities between 0 and 1.

Multiclass Classification: For multiclass problems, logistic regression is extended using techniques such as One-vs-Rest (OvR) or Softmax Regression.

## Import All Necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn import model_selection
from sklearn import linear_model
from sklearn import metrics
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

## Loading the Dataset

```
df = pd.read_csv('/content/test( grayscale).csv')
df
```

	Index	label	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	...	f_1015	f_1016	f_1017
0	12	अ	255	255	255	255	255	255	255	255	...	255.0	255.0	255.0
1	27	अ	255	255	255	255	255	255	255	255	...	255.0	255.0	255.0
2	30	अ	255	255	255	255	255	255	255	255	...	255.0	255.0	255.0
3	33	अ	255	255	255	255	255	255	255	255	...	255.0	255.0	255.0
4	36	अ	255	255	255	255	255	255	255	255	...	255.0	255.0	255.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
772	700	ख	255	255	255	255	255	255	255	255	...	255.0	255.0	255.0
773	704	ख	255	255	255	255	255	255	255	255	...	255.0	255.0	255.0
774	706	ख	255	255	255	255	255	255	255	255	...	255.0	255.0	255.0
775	709	ख	255	255	255	255	255	255	255	255	...	255.0	255.0	255.0
776	711	ख	255	255	255	255	255	255	255	255	...	NaN	NaN	NaN

777 rows × 1026 columns


```
df.shape
```

```
(777, 1026)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 777 entries, 0 to 776
Columns: 1026 entries, Index to f_1024
dtypes: float64(40), int64(985), object(1)
```


```
df.describe()
```



	Index	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	...	
count	777.000000	777.0	777.0	777.0	777.0	777.0	777.0	777.0	777.0	777.0	...	776.
mean	388.496782	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	254.
std	227.953299	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.
min	1.000000	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	244.
25%	189.000000	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	255.
50%	379.000000	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	255.
75%	590.000000	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	255.
max	792.000000	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	255.0	...	255.

8 rows × 1025 columns

```
df.isnull().sum()
```



Index	0
label	0
f_1	0
f_2	0
f_3	0
..	
f_1020	1
f_1021	1
f_1022	1
f_1023	1
f_1024	1

Length: 1026, dtype: int64

Using subplot to plot first 12

```
# Using subplot to plot first 12
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(7.5, 7.5))

# Flatten axes array for easy iteration
axes_flat = axes.flat

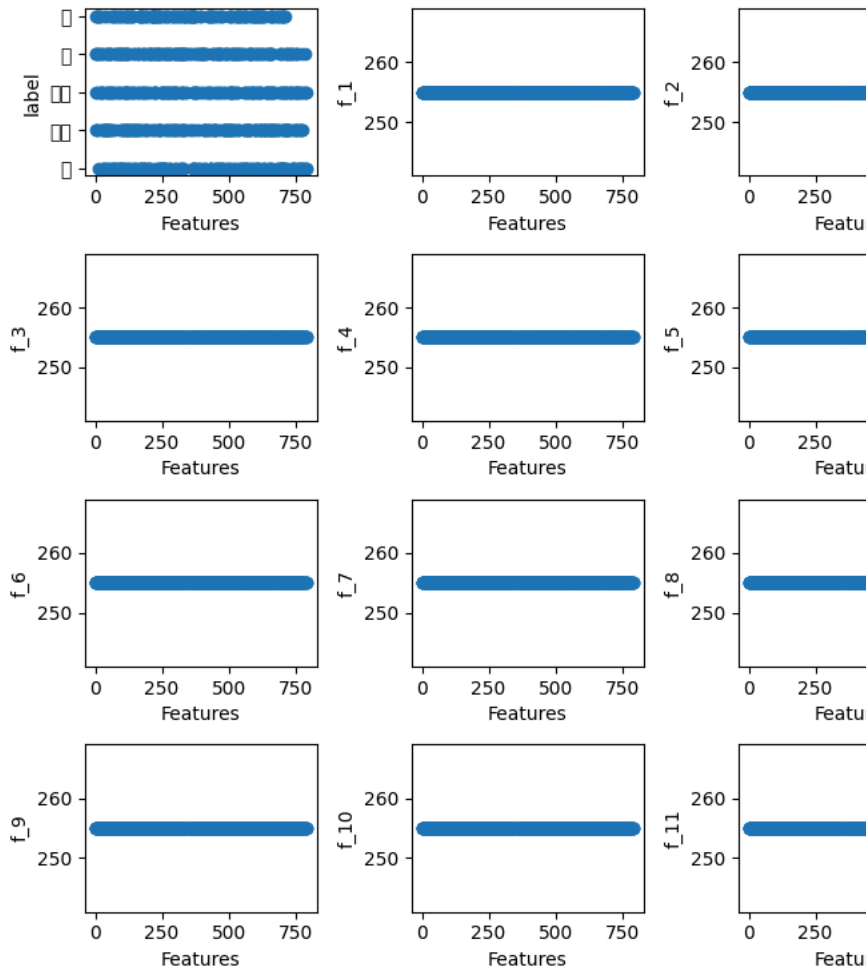
# Iterate over axes and DataFrame columns
for i, ax in enumerate(axes_flat):
    if i < len(df.columns) - 1:
        ax.scatter(df.iloc[:, 0], df.iloc[:, i + 1])
        ax.set_xlabel('Features')
        ax.set_ylabel(df.columns[i + 1])
    else:
        ax.set_visible(False)

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

```

<ipython-input-10-f439ba77a7a5>:17: UserWarning: Glyph 2309 (\N{DEVANAGARI LETTER A})
plt.tight_layout()
<ipython-input-10-f439ba77a7a5>:17: UserWarning: Matplotlib currently does not support
plt.tight_layout()
<ipython-input-10-f439ba77a7a5>:17: UserWarning: Glyph 2306 (\N{DEVANAGARI SIGN ANUSV
plt.tight_layout()
<ipython-input-10-f439ba77a7a5>:17: UserWarning: Glyph 2307 (\N{DEVANAGARI SIGN VISAR
plt.tight_layout()
<ipython-input-10-f439ba77a7a5>:17: UserWarning: Glyph 2310 (\N{DEVANAGARI LETTER AA}
plt.tight_layout()
<ipython-input-10-f439ba77a7a5>:17: UserWarning: Glyph 2311 (\N{DEVANAGARI LETTER I})
plt.tight_layout()

```



Explanation: The code creates a 4x3 grid of scatter plots, each showing the relationship between the first column of the DataFrame (df) and one of the other columns, up to a maximum of 12 subplots. If there are fewer than 12 columns in df, the extra subplots are hidden. The layout is adjusted to ensure a neat display.

## ✓ Select the feature columns and the target variable

```

# Select the feature columns and the target variable
X = df.iloc[:, 2:]
y = df.iloc[:, 1]

```

## ✓ Split the data into training and testing sets

```

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=40)

```

```

print('X_train dimension= ', X_train.shape)
print('X_test dimension= ', X_test.shape)
print('y_train dimension= ', y_train.shape)
print('y_test dimension= ', y_test.shape)

```

```

X_train dimension= (466, 1024)
X_test dimension= (311, 1024)
y_train dimension= (466,)
y_test dimension= (311,)

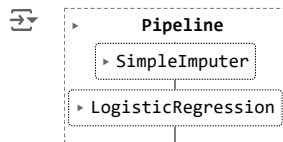
```

```
# Create a pipeline with an imputer and LogisticRegression
pl = Pipeline([
    ('imputer', SimpleImputer(missing_values=np.nan, strategy='mean')),
    ('logistic_regression', LogisticRegression(multi_class='ovr', solver='liblinear'))
])

# Fit the pipeline to the data
pl.fit(X_train, y_train)

# Make predictions
y_pred = pl.predict(X_test)

# Display the pipeline
pl
```



This code creates a pipeline that first imputes missing values and then applies a logistic regression model. The pipeline is trained on the training data, used to make predictions on the test data, and finally displayed to show its structure.

- Print predicted values

```
# Print predicted values
y_pred
```

```
array(['अः', 'इ', 'अं', 'अ', 'अ', 'अं', 'इ', 'इ', 'अ', 'अः', 'आ', 'अः',  
      'इ', 'अं', 'अः', 'आ', 'अः', 'इ', 'अ', 'अं', 'अं', 'अः', 'अः', 'आ',  
      'अ', 'अः', 'आ', 'अं', 'अ', 'अः', 'अ', 'अ', 'अः', 'अ', 'अः', 'आ',  
      'अः', 'अः', 'अं', 'इ', 'अः', 'अ', 'आ', 'इ', 'अं', 'आ', 'अ', 'अं',  
      'अः', 'अ', 'आ', 'अं', 'अः', 'अ', 'आ', 'इ', 'अः', 'अं', 'अं', 'अः',  
      'अ', 'अ', 'आ', 'अ', 'अ', 'अ', 'अ', 'अ', 'अः', 'अ', 'अ', 'अ', 'आ',  
      'अं', 'अ', 'आ', 'आ', 'इ', 'अः', 'अ', 'अं', 'अ', 'अं', 'अ', 'अः', 'इ',  
      'अं', 'अ', 'आ', 'आ', 'आ', 'इ', 'अं', 'अः', 'अ', 'अ', 'इ', 'इ',  
      'अः', 'अ', 'इ', 'अं', 'अं', 'अ', 'आ', 'अ', 'अ', 'अः', 'अः', 'अं',  
      'आ', 'अं', 'अ', 'अः', 'इ', 'अः', 'अ', 'अ', 'अ', 'अ', 'अ', 'इ', 'आ',  
      'इ', 'अ', 'अ', 'अं', 'इ', 'इ', 'अ', 'अ', 'आ', 'आ', 'अं', 'इ', 'इ',  
      'अ', 'अं', 'अः', 'आ', 'अ', 'अ', 'इ', 'अ', 'अ', 'अं', 'इ', 'अः',  
      'अ', 'अ', 'अः', 'अं', 'इ', 'अः', 'अ', 'अ', 'इ', 'अ', 'अ', 'आ',  
      'अ', 'अ', 'अ', 'अ', 'अः', 'अ', 'आ', 'इ', 'अं', 'अः', 'अ', 'अं',  
      'आ', 'अः', 'इ', 'इ', 'आ', 'अं', 'अः', 'अ', 'अः', 'आ', 'अः', 'इ',  
      'अं', 'अं', 'इ', 'अं', 'आ', 'अ', 'आ', 'अं', 'अः', 'इ', 'अ', 'आ',  
      'आ', 'अं', 'अं', 'अः', 'इ', 'आ', 'अः', 'अ', 'अं', 'अः', 'अ', 'इ',  
      'अः', 'अः', 'अः', 'अ', 'अ', 'इ', 'आ', 'आ', 'इ', 'अं', 'आ', 'आ',  
      'अः', 'अः', 'अ', 'आ', 'अं', 'अ', 'आ', 'इ', 'अ', 'अं', 'अं', 'अः',  
      'अं', 'अं', 'अं', 'अ', 'अ', 'इ', 'अं', 'अ', 'अ', 'अं', 'आ', 'अ',  
      'आ', 'अः', 'अं', 'इ', 'अ', 'आ', 'अः', 'अ', 'अं', 'आ', 'अ', 'अं',  
      'अ', 'अः', 'अः', 'अं', 'आ', 'आ', 'इ', 'अं', 'अं', 'अं', 'आ', 'अं',  
      'इ', 'इ', 'अं', 'अं', 'अः', 'आ', 'इ', 'अ', 'अः', 'आ', 'इ'],  
      dtype=object)
```

- ✓ Evaluate the model

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

report = classification_report(y_test, y_pred)
print('Classification Report:\n', report)

conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', conf_matrix)
```

⇒	Accuracy: 1.0				
	Classification Report:				
	precision	recall	f1-score	support	
	अ	1.00	1.00	1.00	62
	अ	1.00	1.00	1.00	66

अः	1.00	1.00	1.00	76
अः	1.00	1.00	1.00	56
अः	1.00	1.00	1.00	51
accuracy			1.00	311
macro avg	1.00	1.00	1.00	311
weighted avg	1.00	1.00	1.00	311

Confusion Matrix:

```
[[62  0  0  0  0]
 [ 0 66  0  0  0]
 [ 0  0 76  0  0]
 [ 0  0  0 56  0]
 [ 0  0  0  0 51]]
```

The accuracy of 1.0 (or 100%) indicates that the model correctly classified all the instances in the test dataset. This is a perfect score, meaning no errors were made in predictions.

The classification report provides detailed performance metrics for each class:

**Precision:** This measures the accuracy of the positive predictions. Precision for each class is 1.00, meaning there are no false positives.

**Recall:** This measures the ability of the model to identify all positive instances. Recall for each class is also 1.00, indicating no false negatives.

**F1-score:** This is the harmonic mean of precision and recall. An F1-score of 1.00 for each class suggests perfect balance between precision and recall.

**Support:** This is the number of actual occurrences of each class in the test dataset.

**Overall Accuracy:** The accuracy is confirmed as 1.00 across all 311 instances.

**Macro Average:** This is the average of the precision, recall, and F1-score for all classes, treating all classes equally.

**Weighted Average:** This is the average of the precision, recall, and F1-score, weighted by the number of instances in each class.

The confusion matrix provides a detailed view of the model's performance for each class:

Each row represents the actual class, and each column represents the predicted class. The diagonal elements (62, 66, 76, 56, 51) represent the number of correct predictions for each class. The off-diagonal elements (all zeros) represent the number of incorrect predictions. Since all off-diagonal elements are zero, there are no misclassifications.

In summary, the model achieved perfect classification performance on the test set, with 100% accuracy, precision, recall, and F1-scores across all classes. The confusion matrix confirms that all predictions were correct for each class. This is an ideal result, suggesting that the model is highly effective for this particular dataset.

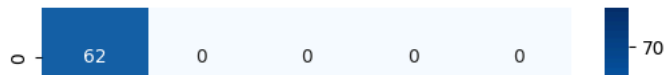
## ✓ Printing the Confusion Matrix

```
# Get the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap of the confusion matrix
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted label")
plt.ylabel("True label")
plt.title("Confusion Matrix")
plt.show()
```



Confusion Matrix



## Printing Precision, Recall score and F1-score.

```
# Print precision recall f1-score separately
from sklearn.metrics import precision_score, recall_score, f1_score

# Calculate precision, recall, and f1-score for each class
precision = precision_score(y_test, pl.predict(X_test), average=None)
recall = recall_score(y_test, pl.predict(X_test), average=None)
f1 = f1_score(y_test, pl.predict(X_test), average=None)

# Print the results
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

Precision: [1. 1. 1. 1. 1.]  
 Recall: [1. 1. 1. 1. 1.]  
 F1-score: [1. 1. 1. 1. 1.]

Precision measures the accuracy of the positive predictions for each class. A precision of 1.0 for all classes means that the model made no false positive errors. Every instance predicted as a given class actually belongs to that class.

Recall measures the ability of the model to correctly identify all positive instances of each class. A recall of 1.0 for all classes means that the model made no false negative errors. Every instance of a given class was correctly identified by the model.

The F1-score is the harmonic mean of precision and recall. An F1-score of 1.0 for all classes indicates a perfect balance between precision and recall, showing that the model is both accurate and comprehensive in its predictions.

**Summary** The results indicate that the model has achieved perfect classification performance for each class in the test set:

Precision, Recall, and F1-score of 1.0 for all classes suggest that the model made no errors in predicting the class labels. This implies that the model perfectly identified and classified all instances in the test data, making it highly effective and reliable for this particular dataset.