Classification with Logistic Regression using Scikit-Learn

Project Overview

This project showcases the implementation of a **Logistic Regression** model using Python's Scikit-Learn library. Logistic Regression is a simple and effective algorithm used for binary and multi-class classification tasks. It models the probability that a given input belongs to a particular category using the logistic (sigmoid) function.

Why Use Logistic Regression?

- **Simplicity**: Easy to implement and computationally efficient.
- **Interpretability**: Coefficients indicate the influence of each feature on the outcome.
- **Probabilistic Output**: Provides class probabilities, making it suitable for ranking tasks.
- **Versatility**: Supports binary and multi-class classification.

Prerequisites

Required Libraries

- `pandas` : For data manipulation and analysis.
- `numpy` : For numerical computations.
- `scikit-learn` : For machine learning algorithms and evaluation metrics.
- `matplotlib` & `seaborn` : For data visualization.

Installation

Install the required libraries using pip:

```
pip install pandas numpy scikit-learn matplotlib seaborn
```

Files Included

- `your_dataset.csv` : The dataset containing the features and target variable.
- `logistic_regression.py` : The Python script implementing Logistic Regression.

Code Description

The implementation is divided into several key steps:

1. Importing Libraries

```
 import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

2. Loading and Exploring the Dataset

```
# Load the dataset
data = pd.read_csv('your_dataset.csv')

# Display the first few rows
print(data.head())

# Check for missing values
print("Missing Values:\n", data.isnull().sum())
```

3. Handling Missing Values

```
# Fill missing numerical values with the mean
data.fillna(data.mean(), inplace=True)
```

4. Preprocessing the Data

```
# Splitting the data into features (X) and target (y)
X = data.iloc[:, :-1]  # Features (all columns except the last)
y = data.iloc[:, -1]   # Target (last column)

# Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

5. Training the Logistic Regression Model

```
# Initialize and train the Logistic Regression model
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train, y_train)
```

6. Making Predictions

```
# Make predictions on the test set
y_pred = model.predict(X_test)
```

7. Evaluating the Model

```
# Confusion matrix, classification report, and accuracy score
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
```

8. Visualizing the Confusion Matrix

```
# Plot confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
```

```
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Expected Outputs

- **Confusion Matrix**: Tabular representation of true vs. predicted labels.
- **Classification Report**: Precision, recall, f1-score, and support for each class.
- **Accuracy Score**: Overall accuracy of the model.
- **Confusion Matrix Heatmap**: Visual representation of the confusion matrix.

Use Cases

- **Healthcare**: Predicting disease diagnosis outcomes.
- **Marketing**: Determining customer purchase likelihood.
- **Finance**: Identifying creditworthiness of loan applicants.
- **Social Media**: Predicting user engagement with posts.

Future Enhancements

- **Hyperparameter Tuning**: Optimize parameters like regularization strength and solver.
- **Cross-Validation**: Ensure robustness and generalizability of the model.
- **Feature Engineering**: Enhance predictive power by creating and selecting relevant features.
- **Class Imbalance Handling**: Address imbalance using techniques like SMOTE or class weighting.

References

- Scikit-Learn Logistic Regression Documentation
- Logistic Regression with Scikit-Learn Tutorial - DataCamp
- Introduction to Logistic Regression in Python
