Self-Training Classifier using Scikit-Learn

Project Description

This project demonstrates the implementation of a Self-Training Classifier, a semi-supervised learning technique that leverages both labeled and unlabeled data to improve classification performance. By iteratively assigning pseudo-labels to unlabeled data, the model enhances its training dataset, aiming to achieve better generalization, especially when labeled data is scarce.

Why Self-Training?

- **Data Efficiency**: Utilizes unlabeled data, which is often more abundant and easier to obtain than labeled data.
- **Improved Performance**: Enhances model accuracy by expanding the training set with high-confidence predictions.
- **Flexibility**: Can be combined with various base classifiers to adapt to different datasets and problems.

Prerequisites

Required Libraries

- Python 3.7 or later
- `numpy` : For numerical computations.
- `pandas` : For data manipulation and analysis.
- `matplotlib` : For data visualization.
- `seaborn` : For advanced visualizations.
- `scikit-learn` : For machine learning models and evaluation metrics.

Installation

Run the following command to install the necessary libraries:

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

Files Included

- `your_dataset.csv` : A placeholder dataset (replace with your actual dataset file).
- Python script implementing the Self-Training Classifier.

Code Description

Steps in the Code

1. Dataset Creation

```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=20, n_informative=2, n_classes=2,
```

A synthetic dataset is generated with 1,000 samples, 20 features, and 2 informative features for binary classification.

2. Simulating Unlabeled Data

```
y[::5] = -1  # Assigning -1 (unlabeled) to every 5th sample
```

To simulate unlabeled data, every 5th label in the target array `y` is set to `-1`.

3. Splitting Data into Training and Test Sets

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

The dataset is split into training and testing sets, with 80% for training and 20% for testing.

4. Initializing the Base Classifier

```
from sklearn.ensemble import RandomForestClassifier
base_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

A `RandomForestClassifier` is chosen as the base estimator for the self-training model.

5. Creating and Training the Self-Training Model

```
from sklearn.semi_supervised import SelfTrainingClassifier
self_training_model = SelfTrainingClassifier(base_classifier)
self_training_model.fit(X_train, y_train)
```

The `SelfTrainingClassifier` wraps the base classifier and is trained on the training data, which includes both labeled and unlabeled samples.

6. Making Predictions on the Test Set

```
y_pred = self_training_model.predict(X_test)
```

Predictions are made on the test dataset.

7. Evaluating Model Performance

Accuracy Score:

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Classification Report:

```
from sklearn.metrics import classification_report
print("Classification Report:\n", classification_report(y_test, y_pred))
```

The model's performance is assessed using accuracy and a detailed classification report.

8. Visualizing the Confusion Matrix

```
 from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"], y
plt.title("Confusion Matrix")
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.show()
```

A heatmap of the confusion matrix is plotted to visualize the model's performance across different classes.

Outputs

Metrics

- **Accuracy Score**: Indicates the overall correctness of the model's predictions.
- **Classification Report**: Provides precision, recall, F1-score, and support for each class.

Visualization

- **Confusion Matrix**: A heatmap displaying the true vs. predicted classifications, aiding in the assessment of model performance.

Example Output

```
 Accuracy: 85.00%
Classification Report:
              precision    recall  f1-score   support
           0       0.85      0.86      0.85        90
           1       0.85      0.84      0.85        80
    accuracy                           0.85       170
   macro avg       0.85      0.85      0.85       170
weighted avg       0.85      0.85      0.85       170
```

Note: These values are hypothetical and will vary based on the actual dataset used.

Use Cases

This project is applicable for:

- **Situations with Limited Labeled Data**: When acquiring labeled data is expensive or time-consuming, and unlabeled data is readily available.
- **Improving Model Performance**: Enhancing the accuracy of classifiers by incorporating unlabeled data through self-training.
- **Exploratory Data Analysis**: Understanding the potential benefits of semi-supervised learning techniques in various domains.

Future Enhancements

- **Experimenting with Different Base Classifiers**: Assessing the impact of various classifiers on the self-training process.
- **Hyperparameter Tuning**: Optimizing parameters such as the self-training threshold and the base classifier’s settings to improve performance.
- **Evaluating with Real-World Datasets**: Applying the self-training classifier to real-world datasets to validate its effectiveness in practical scenarios.