Co-Training Model Implementation

Project Description

This project demonstrates the implementation of a **Co-Training Model**, a semi-supervised learning technique that utilizes two distinct classifiers trained on different views of the data. By iteratively labeling unlabeled data, each classifier aids the other in improving performance, especially when labeled data is limited.

Why Co-Training?

- **Semi-Supervised Learning**: Leverages both labeled and unlabeled data to enhance model accuracy.
- **Diverse Perspectives**: Utilizes multiple views or feature sets, allowing classifiers to complement each other's strengths.
- **Improved Generalization**: Reduces overfitting by incorporating diverse information from different data views.

Prerequisites

Required Libraries

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

Installation

Run the following command to install the necessary libraries:

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

Files Included

- **co_training_model.py**: Python script implementing the Co-Training model.
- **your_dataset.csv**: Placeholder for your dataset file.

Code Description

Steps in the Code

1. Dataset Creation

A synthetic dataset is generated using `make_classification` from `scikit-learn`.

```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
```

- **Description**: Generates a dataset with 1,000 samples, 20 features, and 2 classes for binary classification.

2. Simulating Unlabeled Data

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

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

3. Splitting Data into Training and Testing Sets

The dataset is split into training and testing sets using `train_test_split`.

```
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=4
```

- **Description**: 80% of the data is used for training, and 20% is reserved for testing.

4. Defining the Co-Training Classifier

A custom `CoTraining` class is defined, which initializes two base learners and iteratively updates them with pseudo-labeled data.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.base import BaseEstimator, ClassifierMixin

class CoTraining(BaseEstimator, ClassifierMixin):
    def __init__(self, base_learner=DecisionTreeClassifier(), n_iterations=10):
        self.base_learner = base_learner
        self.n_iterations = n_iterations

    def fit(self, X, y):
        # Split data into two views (features)
        X1, X2 = X[:, :X.shape[1]//2], X[:, X.shape[1]//2:]

        # Initialize two base learners
        self.learner1 = self.base_learner.fit(X1, y)
        self.learner2 = self.base_learner.fit(X2, y)

        for _ in range(self.n_iterations):
            # Generate pseudo-labels from each classifier for the unlabeled data
            pseudo_labels1 = self.learner1.predict(X1)
            pseudo_labels2 = self.learner2.predict(X2)

            # Add pseudo-labeled samples to each classifier's training set
            X1_new, y1_new = X1[pseudo_labels1 != -1], pseudo_labels1[pseudo_labels1 !=
            X2_new, y2_new = X2[pseudo_labels2 != -1], pseudo_labels2[pseudo_labels2 !=

            # Re-train with updated labels
            self.learner1.fit(X1_new, y1_new)
            self.learner2.fit(X2_new, y2_new)

        return self

    def predict(self, X):
        X1, X2 = X[:, :X.shape[1]//2], X[:, X.shape[1]//2:]
        pred1 = self.learner1.predict(X1)
        pred2 = self.learner2.predict(X2)
        # Combine predictions from both learners
        return np.round((pred1 + pred2) / 2)
```

- **Description**: The `CoTraining` class uses two classifiers trained on different feature views and iteratively improves their performance using pseudo-labels.

---

5. Training the Co-Training Model

The `CoTraining` model is instantiated and trained on the training data.

```
co_training_model = CoTraining(n_iterations=5)
co_training_model.fit(X_train, y_train)
```

---

6. Making Predictions on the Test Set

Predictions are made on the test dataset using the trained model.

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

---

7. Evaluating Model Performance

The model's performance is assessed using accuracy.

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

---

8. Visualizing the Results

A scatter plot is generated to visualize the model's predictions on the test set.

```
import matplotlib.pyplot as plt
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='plasma')
plt.title("Co-Training - Prediction")
plt.show()
```

Outputs

Metrics

- **Accuracy Score**: Indicates the overall correctness of the model's predictions.

Visualization

- **Scatter Plot**: Displays the predicted classifications on a 2D feature space.

Example Output

```
Accuracy: 85.00%
```

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

Use Cases

This project is applicable for:

- **Semi-Supervised Learning Scenarios**: When labeled data is limited, and unlabeled data is abundant.
- **Multi-View Data**: Datasets that can be naturally split into distinct feature sets or views.
- **Improving Classifier Performance**: Enhancing accuracy by leveraging unlabeled data through co-training.

Future Enhancements

- **Diverse Base Learners**: Experimenting with different classifiers to assess their impact on co-training effectiveness.
- **Parameter Optimization**: Tuning hyperparameters such as the number of iterations and base learner settings.
- **Real-World Applications**: Applying the co-training model to real-world datasets to validate its practical utility.