1.Self-training

Self-training is a semi-supervised learning approach where a single classifier is trained on a small set of labeled data and then used to assign labels to unlabeled data. These pseudo-labeled instances are then added to the training set, and the classifier is retained iteratively.

In general,

At first, the classifier will be trained on the labeled data.

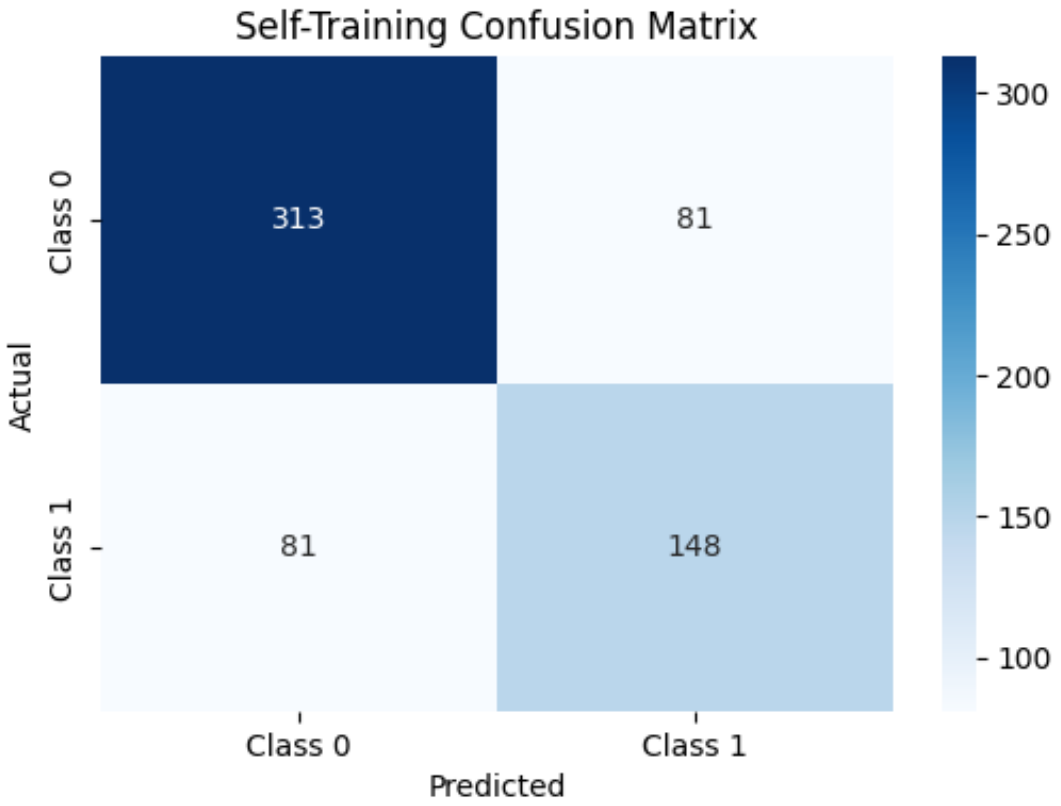
Then it predicts labels for unlabeled instances.

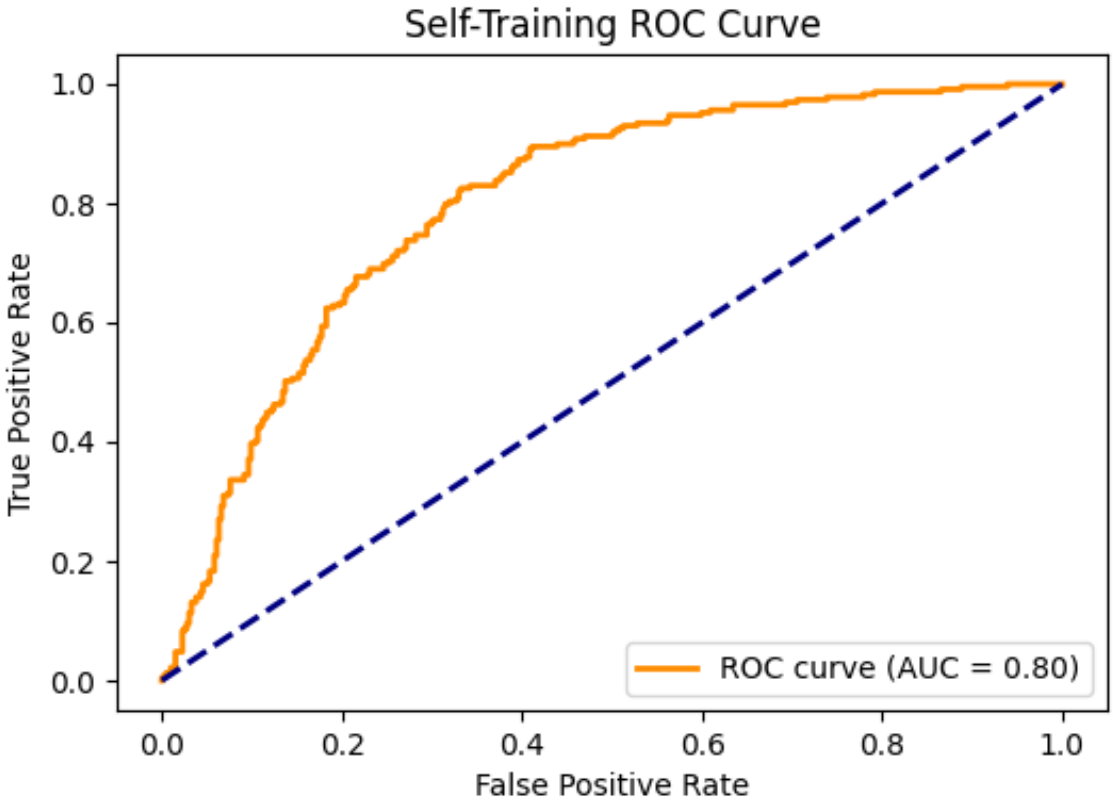
After that, it selects high-confidence predictions(probability >= 0.8), those predicted pseudo-labeled instances will be added into the train set.

After adding, it retrains the base classifier with this new expanded transit.

The above process will be repeated for max_iter times

```
# Initialize base classifier
base_clf = GradientBoostingClassifier(random_state=42)

# Initialize Self-Training Classifier
self_training_clf = SelfTrainingClassifier(base_estimator=base_clf, threshold=0.8,
max_iter=10, verbose=True)

# Fit the model
print("\nTraining Self-Training Classifier...")
self_training_clf.fit(X_train, y_train)

# Save the model
save_model(self_training_clf, directory='models', filename='self_training_model.joblib')

# Predictions
y_pred = self_training_clf.predict(X_test)
y_pred_proba = self_training_clf.predict_proba(X_test)[:, 1]
```


Self-Training Confusion Matrix
313
81
81
148
Class 0
Class 1
Actual
Class 0
Class 1
Predicted
300
250
200
150
100


Self-Training ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.80)

2. Co-training

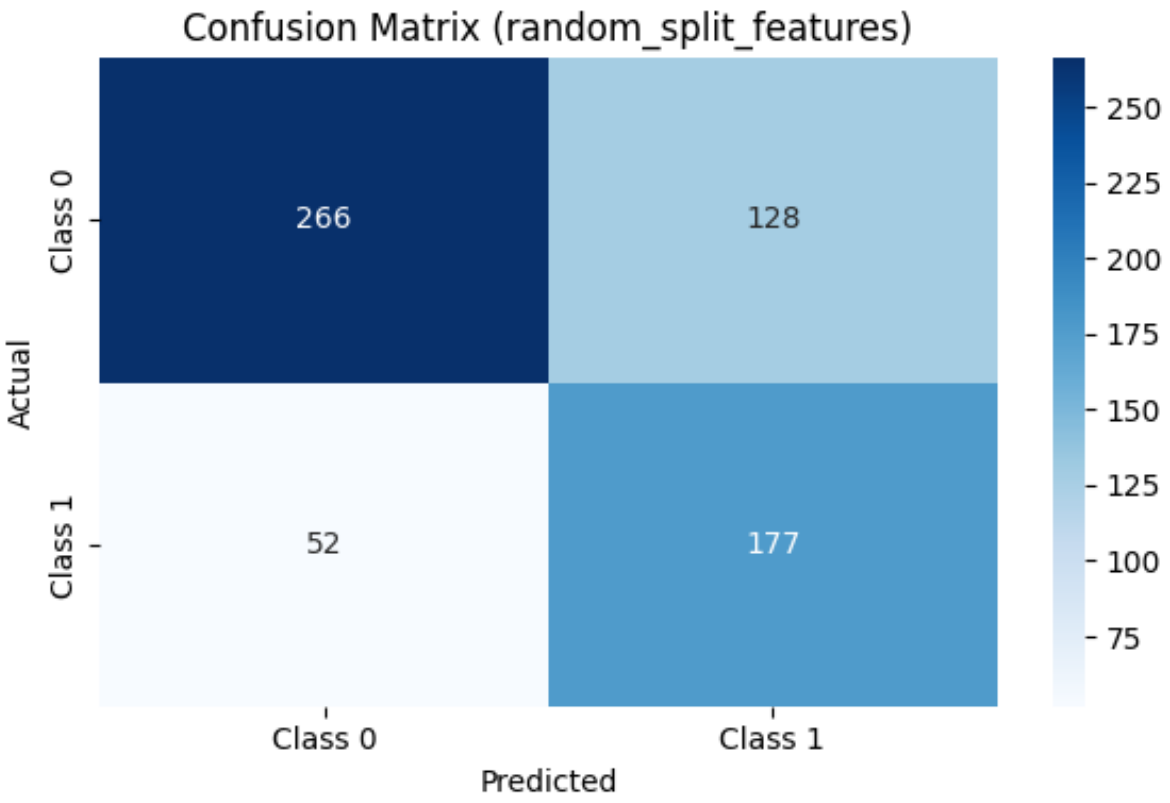
1. Split Features into 2 views(I shuffled the features and then split, for in the original order, all descriptive features are in the front, if we follow this, we will have a very bad result)
2. Initialize 2 classifiers, 1 for each view
3. Train both classifiers on labeled data
4. Iteratively Pseudo-Label Unlabeled data

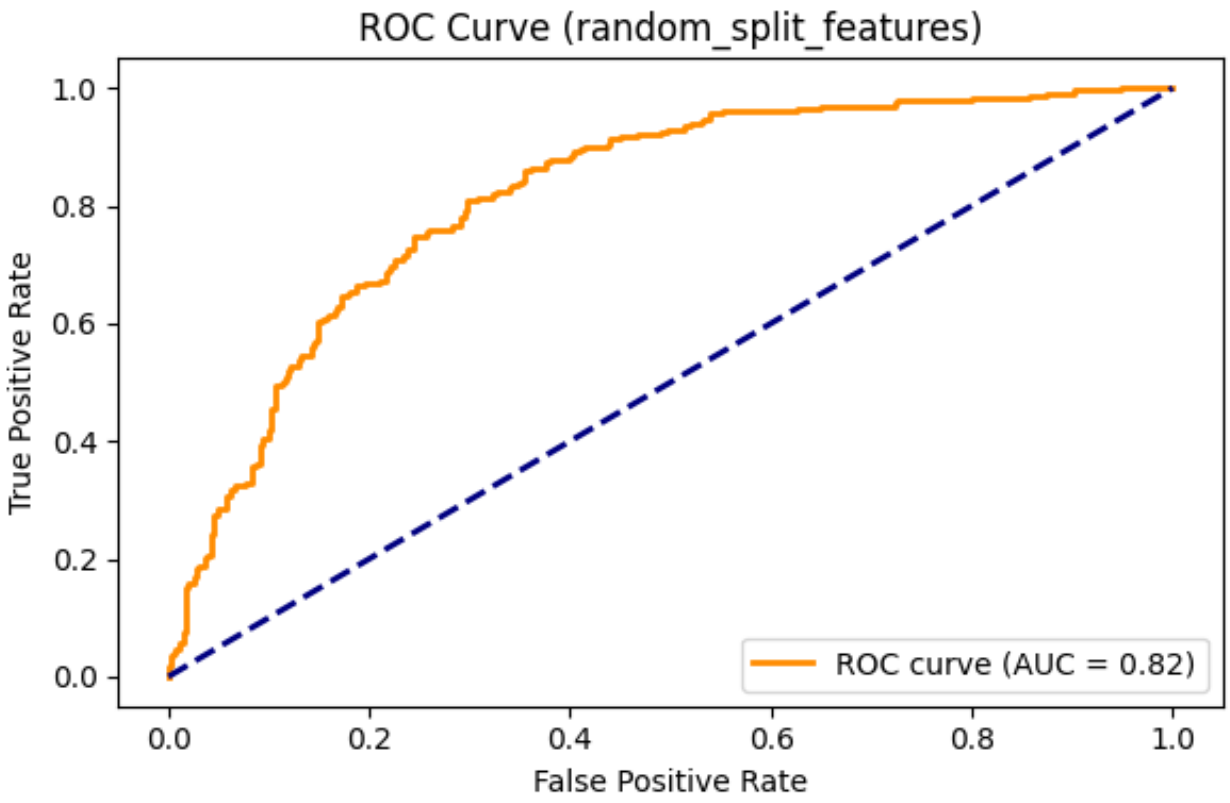
   a. Classifier1 labels unlabeled data for View2 and add those newly labeled data to unlabeled data of View2

   b. Classifier2 labels unlabeled data for View1 and add those newly labeled data to unlabeled data of View2

For in the original order, all descriptive features are in the front, if we follow this, we will have a very bad result. After finding this, I shuffled all features, then we get a better result compared with the unshuffled version. Then I tried PCA-based method, importance-based, correlation-based split ways and with different size of feature view, to get the best results of co-training effect.

Best Method: random_split_features
Best Result:





3. Semi-boosting

The core of the algorithm is a combination of AdaBoost and self-training.

AdaBoost Classifier:

- Boosts the performance of weak learners (here, a Decision Tree with depth 1) by iteratively focusing on harder-to-classify samples.
- Combines multiple weak classifiers to form a strong classifier.

Self-Training Classifier:

- Leverages a base estimator (in this case, AdaBoost) to pseudo-label unlabeled data based on model confidence.
- Iteratively trains on the pseudo-labeled dataset to improve performance.

Initialize base estimator as a weak learner

base_estimator = DecisionTreeClassifier(max_depth=1, random_state=42)

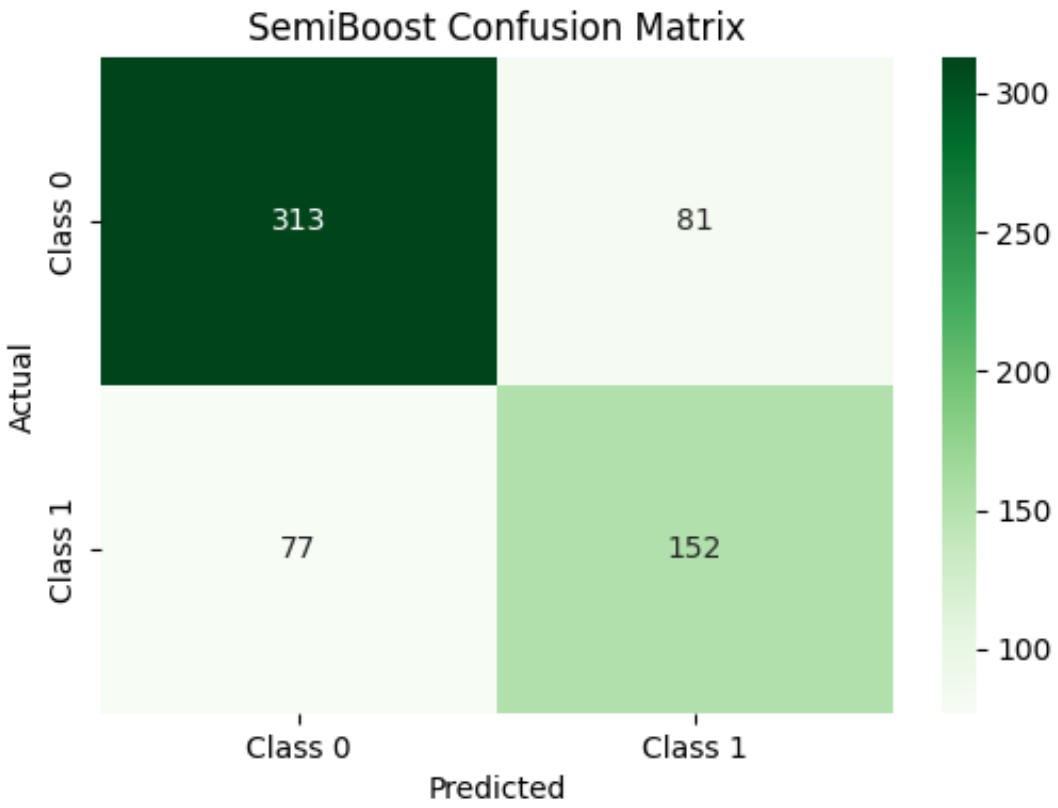
Initialize AdaBoost Classifier with the weak learner

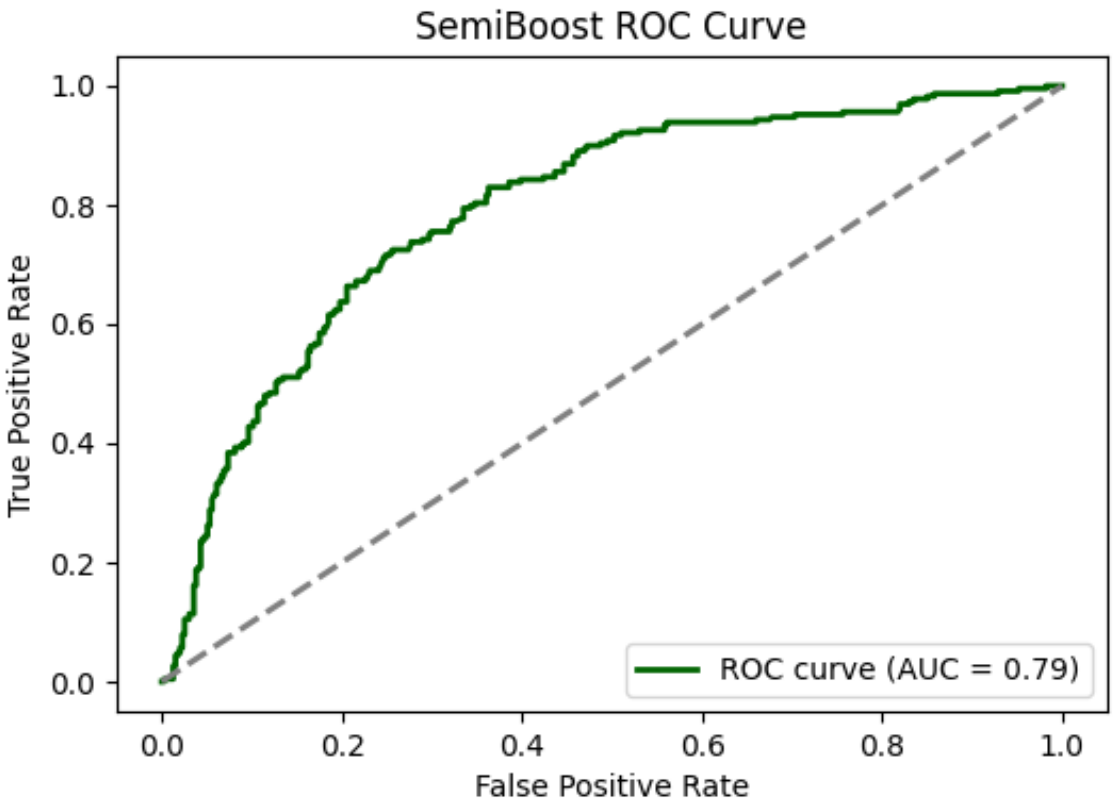
ada_clf = AdaBoostClassifier(estimator=base_estimator, n_estimators=50, random_state=42)

Initialize Self-Training Classifier with AdaBoost as base

self_training_clf = SelfTrainingClassifier(base_estimator=ada_clf, threshold=0.8, max_iter=10, verbose=True)

SemiBoost Test Accuracy: 0.7464
SemiBoost AUC-ROC Score: 0.7929





Label spreading

Label Spreading is a semi-supervised learning algorithm that propagates label information from labeled instances to unlabeled instances using a similarity graph. This technique relies on the principle that points close to each other in the feature space are likely to have similar labels.

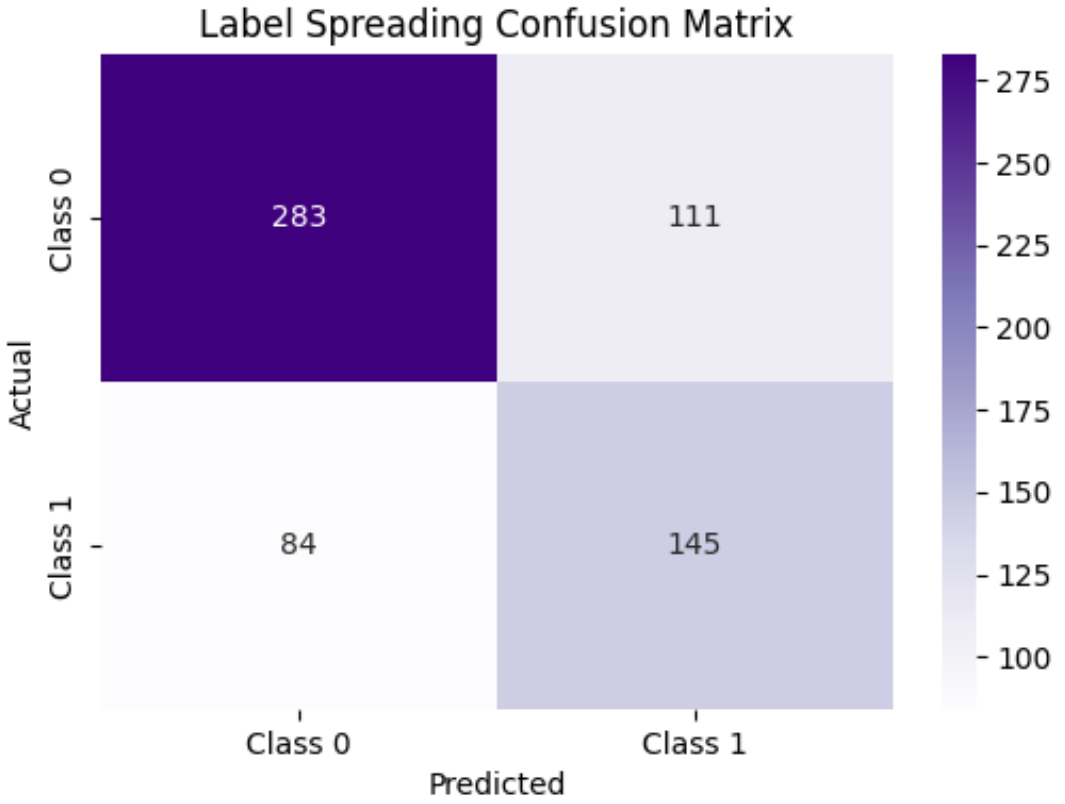
In this implementation, we use **Label Spreading** with an RBF (Radial Basis Function) kernel to construct the similarity graph. It iteratively spreads the label information until convergence or the maximum number of iterations is reached.

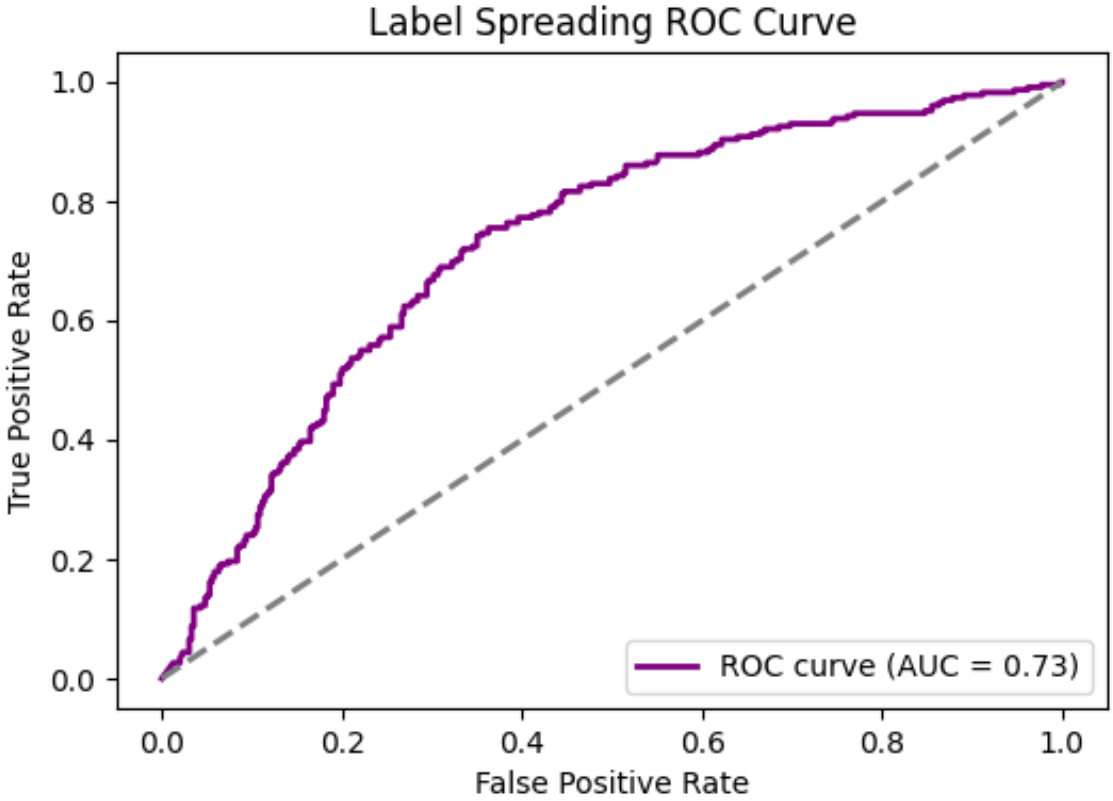
label_spread = LabelSpreading(kernel='rbf', gamma=20, max_iter=30)

Kernel: `'rbf'` is used to compute the similarity between points based on distance.

Gamma: Controls the scale of the RBF kernel (higher values focus on closer points).

Max Iterations: Limits the number of label propagation steps.


Label Spreading Confusion Matrix
283
111
84
145
Class 0
Class 1
Actual
Class 0
Class 1
Predicted
275
250
225
200
175
150
125
100


Label Spreading ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.73)