Plug and Play ROC

Absolutely! The code provided above is designed to be **plug-and-play** for various models. Let's break it down to ensure that you can easily adapt the code to fit different models without much hassle.

Adapting the Code for Different Models

The key parts that need adjustment for different models are:

1. **Model Initialization**: Change the model you're using (Logistic Regression, Decision Tree, Random Forest, SVM, etc.).
2. **Predicted Probabilities**: Ensure that the model supports `predict_proba()` or use `decision_function()` for models like SVM if `predict_proba()` is not available.

Generalized Approach: Plug-and-Play Code

Below, I've organized the code into a function that you can reuse for multiple models. This function will work for **Logistic Regression**, **Decision Trees**, **Random Forest**, and **SVM** without requiring major changes.

Step 1: Define a Function to Plot Multiclass ROC Curve

```
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np

def plot_multiclass_roc(model, X_train, X_test, y_train, y_test):
    """
    Function to plot the ROC curve for a given model.
    Supports multiclass classification using One-vs-Rest strategy.
    """
    # Fit the model on training data
    model.fit(X_train, y_train)

    # Binarize the output labels for multiclass ROC analysis
    classes = np.unique(y_train)
    y_test_binarized = label_binarize(y_test, classes=classes)
    n_classes = y_test_binarized.shape[1]

    # Predict probabilities
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)
    elif hasattr(model, "decision_function"):
        y_proba = model.decision_function(X_test)
    else:
        raise ValueError("Model does not support probability predictions")

    # Initialize dictionaries to store ROC data
    fpr = {}
    tpr = {}
    roc_auc = {}

    # Compute ROC curve and AUC for each class
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test_binarized.ravel(),
y_proba.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    # Plot the ROC curves
    plt.figure(figsize=(10, 8))
    colors = ['aqua', 'darkorange', 'cornflowerblue']
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label=f'Class {classes[i]} (AUC = {roc_auc[i]:0.2f})')

    # Plot micro-average ROC curve
    plt.plot(fpr["micro"], tpr["micro"], color='deeppink', linestyle=':',
linewidth=2,
             label=f'Micro-average ROC curve (AUC = {roc_auc["micro"]:0.2f})')

    # Plot diagonal line for reference
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Multiclass ROC Curve - {type(model).__name__}', fontsize=16)
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.show()
```

Step 2: Using the Function with Different Models

You can now call the function with any of the supported models:

Example 1: Logistic Regression

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000, solver='lbfgs')
plot_multiclass_roc(model, X_train, X_test, y_train, y_test)
```

Example 2: Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
plot_multiclass_roc(model, X_train, X_test, y_train, y_test)
```

Example 3: Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100)
plot_multiclass_roc(model, X_train, X_test, y_train, y_test)
```

Example 4: Support Vector Machine (SVM)

```
from sklearn.svm import SVC

model = SVC(kernel='rbf', probability=True)
plot_multiclass_roc(model, X_train, X_test, y_train, y_test)
```

Notes

1. **Probability Predictions**:
   - For most classifiers (`LogisticRegression`, `DecisionTreeClassifier`, `RandomForestClassifier`), use `predict_proba()`.

   - For SVM, set `probability=True` when initializing the model to enable `predict_proba()`.
2. **Handling Models without Probability Predictions**:
   - The function checks if the model supports `predict_proba()` or `decision_function()`.
   - If neither is available, the function raises an error.
3. **Customizing the Function**:
   - You can easily adjust the plot's appearance (colors, line styles, labels).
   - You can add additional metrics like precision-recall curves if needed.

Absolutely! Let's extend the **plug-and-play functionality** to include additional metrics like **F1-score**, **Precision**, **Recall**, and the **Confusion Matrix**. This will allow you to evaluate multiple models with consistent reporting.

Updated Approach

I'll create a single function that can:

1. **Train a given model**.
2. **Generate ROC curves for multiclass classification**.
3. **Compute evaluation metrics**: F1-score, Precision, Recall, and Confusion Matrix.
4. **Display the results in a structured manner**.

Plug-and-Play Function for Model Evaluation

Below is a comprehensive function that handles everything:

- **ROC Curve & AUC**
- **F1-Score, Precision, Recall**
- **Confusion Matrix**

Comprehensive Function

```
from sklearn.metrics import roc_curve, auc, roc_auc_score, f1_score,
precision_score, recall_score, confusion_matrix, classification_report
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
```

```python
import seaborn as sns
import numpy as np

def evaluate_model(model, X_train, X_test, y_train, y_test):
    """
    Function to train the model, generate metrics (F1, Precision, Recall,
Confusion Matrix),
    and plot the ROC curve for multiclass classification.
    """
    # Step 1: Train the model
    model.fit(X_train, y_train)

    # Step 2: Binarize the output labels for multiclass ROC analysis
    classes = np.unique(y_train)
    y_test_binarized = label_binarize(y_test, classes=classes)
    n_classes = y_test_binarized.shape[1]

    # Step 3: Predict probabilities for ROC curve
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)
    elif hasattr(model, "decision_function"):
        y_proba = model.decision_function(X_test)
    else:
        raise ValueError("Model does not support probability predictions")

    # Step 4: Predict class labels for other metrics
    y_pred = model.predict(X_test)

    # Step 5: Compute evaluation metrics (F1, Precision, Recall)
    f1 = f1_score(y_test, y_pred, average='weighted')
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')

    print(f"\nModel: {type(model).__name__}")
    print(f"F1-Score (Weighted): {f1:.2f}")
    print(f"Precision (Weighted): {precision:.2f}")
    print(f"Recall (Weighted): {recall:.2f}")

    # Step 6: Display classification report
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    # Step 7: Generate confusion matrix
    print("\nConfusion Matrix:")
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes,
yticklabels=classes)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title(f'Confusion Matrix - {type(model).__name__}')
    plt.show()

    # Step 8: Compute ROC curve and AUC for each class
    fpr = {}
    tpr = {}
    roc_auc = {}

    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test_binarized.ravel(),
y_proba.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    # Step 9: Plot the ROC curves
    plt.figure(figsize=(10, 8))
    colors = ['aqua', 'darkorange', 'cornflowerblue']
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label=f'Class {classes[i]} (AUC = {roc_auc[i]:0.2f})')

    # Plot micro-average ROC curve
    plt.plot(fpr["micro"], tpr["micro"], color='deeppink', linestyle=':',
linewidth=2,
             label=f'Micro-average ROC curve (AUC = {roc_auc["micro"]:0.2f})')

    # Plot diagonal line
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Multiclass ROC Curve - {type(model).__name__}')
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.show()
```

Step 2: Using the Function with Different Models

You can call the function with any classifier you want to evaluate. Here are examples of how to use it.

Example 1: Logistic Regression

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=1000)
evaluate_model(model, X_train, X_test, y_train, y_test)
```

Example 2: Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
evaluate_model(model, X_train, X_test, y_train, y_test)
```

Example 3: Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100)
evaluate_model(model, X_train, X_test, y_train, y_test)
```

Example 4: Support Vector Machine (SVM)

```
from sklearn.svm import SVC

model = SVC(kernel='rbf', probability=True)
evaluate_model(model, X_train, X_test, y_train, y_test)
```

Notes on How the Function Works

1. **Training the Model**:
    - Trains the specified model using `fit()` on the training data.
2. **ROC Curve Calculation**:

    - Uses `predict_proba()` or `decision_function()` depending on the model to generate probabilities.
    - Plots ROC curves for each class and a micro-average curve.
3. **Evaluation Metrics**:
    - Computes **F1-score**, **Precision**, and **Recall** using `average='weighted'` to handle class imbalances.
    - Displays a **confusion matrix** for visual evaluation.
4. **Adaptable**:
    - Works for any model that supports `predict_proba()` or `decision_function()`.
    - Easy to extend for other classifiers by simply passing the model to the function.

Key Takeaways

- This single function handles **model training**, **evaluation**, and **visualization**.
- You can easily swap out the model to see how different classifiers perform on your dataset.
- It provides a comprehensive view of your model's performance, including **ROC curves**, **confusion matrix**, and standard classification metrics.

This approach is perfect for exam preparation and for quickly evaluating the performance of various models on your dataset. Let me know if you need additional customizations!