**Soft Voting Classifier**

A **Soft Voting Classifier** is an ensemble learning method where multiple models (classifiers) are combined to make predictions. In soft voting, each classifier predicts the probability of a class, and the final prediction is made by averaging these probabilities. The class with the highest average probability is chosen as the final prediction.

#### **Key Concepts:**

1. **Voting Classifier**:
   * **Ensemble Learning**: Combines predictions from multiple models to improve overall accuracy and robustness.
   * **Soft Voting**: Each classifier predicts probabilities for each class. These probabilities are averaged across classifiers, and the class with the highest probability is selected.
   * **Hard Voting**: Each classifier directly votes for a class. The class with the majority of votes is the final prediction. (Soft voting is usually preferred in practice because it incorporates more information from each classifier.)
2. **Advantages of Soft Voting**:
   * **Improved Accuracy**: By averaging probabilities, soft voting tends to smooth out the predictions, reducing the impact of outliers or highly confident but incorrect predictions from individual models.
   * **Model Diversity**: Combining different models (e.g., logistic regression, decision tree, random forest) can improve performance, especially if they are good at handling different types of patterns in the data.
3. **When to Use**:
   * When you want to combine the strengths of different models.
   * When the base classifiers are probabilistic (i.e., can provide class probabilities).
   * When you want a more stable and robust model that reduces overfitting.

### **Code Explanation:**

|  |
| --- |
| *# Import necessary libraries* import numpy as np import pandas as pd from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import VotingClassifier, BaggingClassifier from sklearn.linear\_model import LogisticRegression from sklearn.tree import DecisionTreeClassifier from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import roc\_curve, roc\_auc\_score, RocCurveDisplay from sklearn.calibration import calibration\_curve, CalibrationDisplay from sklearn.preprocessing import LabelEncoder, StandardScaler import matplotlib.pyplot as plt |

This part imports the required libraries for data manipulation (pandas, numpy), machine learning (scikit-learn), and plotting (matplotlib).

### **1. Data Preparation:**

|  |
| --- |
| *# Load the dataset* data = pd.read\_csv(r"D:\Research\Personal Projects\Breast Cancer Data\Data\Breast\_Cancer.csv")  *# Encode categorical variables and target* *# Encode 'Status' column as target (0 for 'Dead', 1 for 'Alive')* data['Status'] = data['Status'].apply(lambda x: 1 **if** x == 'Alive' **else** 0)  *# Updated categorical columns* categorical\_columns = ['Race', 'Marital\_Status', 'T\_Stage', 'N\_Stage', '6th\_Stage', 'differentiate',   'Grade', 'A\_Stage', 'Estrogen\_Status', 'Progesterone\_Status']  *# Initialize LabelEncoder* le = LabelEncoder()  *# Encode categorical columns* **for** col **in** categorical\_columns:  data[col] = le.fit\_transform(data[col])  *# Define features (X) and target (y)* X = data.drop('Status', axis=1) *# Features* y = data['Status'] *# Target*  *# Split data into training and testing sets (70% train, 30% test)* X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  *# Optional: Scale features (standardize them)* scaler = StandardScaler() X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test) |

* **Data Loading**: Loads the breast cancer dataset from a CSV file.
* **Encoding**: Converts categorical variables (e.g., 'Race', 'Marital\_Status') into numerical values using LabelEncoder.
* **Target Encoding**: The 'Status' column, which indicates if the patient is 'Alive' or 'Dead', is encoded as binary (1 for Alive, 0 for Dead).
* **Splitting the Data**: Divides the data into training (70%) and testing (30%) sets.
* **Feature Scaling**: Scales the features using StandardScaler to standardize them (mean = 0, variance = 1).

### **2. LazyClassifier for Quick Model Comparison:**

|  |
| --- |
| *# Step 1: Use Lazy Predict to quickly compare models* clf = LazyClassifier(verbose=0, ignore\_warnings=True, custom\_metric=None) models, predictions = clf.fit(X\_train, X\_test, y\_train, y\_test)  *# Display Lazy Predict results* print("Lazy Predict Model Results:") print(models) |

* **LazyClassifier**: This is a convenience tool for quickly comparing the performance of different classifiers. It runs a variety of machine learning models on the data and returns the accuracy, training time, and other metrics for each model.
* **Purpose**: Helps to get an overview of which models might perform well before focusing on specific ones.

### **3. Defining Specific Models:**

|  |
| --- |
| *# Step 2: Define specific models for soft voting* log\_clf = LogisticRegression(solver='liblinear', random\_state=42) rf\_clf = RandomForestClassifier(n\_estimators=100, random\_state=42) dt\_clf = DecisionTreeClassifier(random\_state=42) |

* **Logistic Regression** (log\_clf): A linear model used for binary classification.
* **Random Forest** (rf\_clf): An ensemble of decision trees that reduces overfitting by averaging predictions.
* **Decision Tree** (dt\_clf): A tree-based model that is easy to interpret but prone to overfitting.

### **4. Soft Voting Classifier:**

|  |
| --- |
| *# Step 3: Create a Soft Voting Classifier* voting\_clf = VotingClassifier(estimators=[('lr', log\_clf), ('rf', rf\_clf), ('dt', dt\_clf)], voting='soft')  *# Train the Voting Classifier* voting\_clf.fit(X\_train, y\_train) |

* **Voting Classifier**: Combines the three classifiers (log\_clf, rf\_clf, dt\_clf).
  + The parameter voting='soft' ensures that the classifiers' predicted probabilities are averaged.
* **Training**: The ensemble is trained on the training data.

### **5. ROC Curve:**

|  |
| --- |
| *# Step 4: Plot ROC Curve for individual models and the voting classifier* **for** clf **in** (log\_clf, rf\_clf, dt\_clf, voting\_clf):  clf.fit(X\_train, y\_train)  y\_pred\_proba = clf.predict\_proba(X\_test)[:, 1]    *# Plot ROC Curve*  fpr, tpr, \_ = roc\_curve(y\_test, y\_pred\_proba)  roc\_auc = roc\_auc\_score(y\_test, y\_pred\_proba)  plt.plot(fpr, tpr, label=f'{clf.\_\_class\_\_.\_\_name\_\_} (AUC = {roc\_auc:.2f})')  plt.plot([0, 1], [0, 1], 'k--') plt.xlim([0.0, 1.0]) plt.ylim([0.0, 1.05]) plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate') plt.title('ROC Curve') plt.legend(loc="lower right") plt.show() |

* **ROC Curve**: A graph that shows the performance of a classifier across all decision thresholds by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR).
* **AUC**: The Area Under the Curve (AUC) score represents the classifier's ability to distinguish between the classes.

### **6. Calibration Curve:**

|  |
| --- |
| *# Step 5: Calibration Plot for each classifier* **for** clf **in** (log\_clf, rf\_clf, dt\_clf, voting\_clf):  prob\_pos = clf.predict\_proba(X\_test)[:, 1]    *# Use CalibrationDisplay's from\_predictions method*  CalibrationDisplay.from\_predictions(y\_test, prob\_pos, n\_bins=10)   plt.title("Calibration Curve") plt.show() |

* **Calibration Curve**: Shows how well predicted probabilities match actual outcomes. A perfectly calibrated model would have all predictions lie on the diagonal line.

### **7. Bagging Classifier:**

|  |
| --- |
| *# Step 6: Bagging Classifier using the Voting Classifier* bagging\_clf = BaggingClassifier(base\_estimator=voting\_clf, n\_estimators=10, random\_state=42) bagging\_clf.fit(X\_train, y\_train) y\_pred\_bagging = bagging\_clf.predict(X\_test) |

* **Bagging Classifier**: Another ensemble method where multiple instances of the voting classifier are trained on random subsets of the data, and their predictions are averaged. This helps to further reduce variance and overfitting.

### **8. Evaluating Bagging Classifier with ROC Curve:**

|  |
| --- |
| *# Step 7: Evaluate Bagging Classifier with ROC Curve* **for** clf **in** (log\_clf, rf\_clf, dt\_clf, voting\_clf):  clf.fit(X\_train, y\_train)  RocCurveDisplay.from\_estimator(clf, X\_test, y\_test, name=clf.\_\_class\_\_.\_\_name\_\_)  plt.plot([0, 1], [0, 1], 'k--') plt.xlim([0.0, 1.0]) plt.ylim([0.0, 1.05]) plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate') plt.title('ROC Curve for Bag |