import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

# Load the dataset

file\_path = '/path/to/lung\_cancer\_examples.csv'

data = pd.read\_csv(file\_path)

# Drop non-numeric columns

data = data.drop(['Name', 'Surname'], axis=1)

# Separate features and target variable

X = data.drop('Result', axis=1)

y = data['Result']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features for SVM

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Initialize the models

svm\_model = SVC()

decision\_tree\_model = DecisionTreeClassifier()

# Train the models

svm\_model.fit(X\_train\_scaled, y\_train)

decision\_tree\_model.fit(X\_train, y\_train)

# Make predictions

svm\_predictions = svm\_model.predict(X\_test\_scaled)

decision\_tree\_predictions = decision\_tree\_model.predict(X\_test)

# Calculate accuracy, precision, recall, and f1-score

svm\_accuracy = accuracy\_score(y\_test, svm\_predictions)

svm\_precision = precision\_score(y\_test, svm\_predictions)

svm\_recall = recall\_score(y\_test, svm\_predictions)

svm\_f1 = f1\_score(y\_test, svm\_predictions)

decision\_tree\_accuracy = accuracy\_score(y\_test, decision\_tree\_predictions)

decision\_tree\_precision = precision\_score(y\_test, decision\_tree\_predictions)

decision\_tree\_recall = recall\_score(y\_test, decision\_tree\_predictions)

decision\_tree\_f1 = f1\_score(y\_test, decision\_tree\_predictions)

# Print results

print("SVM Classifier:")

print(f"Accuracy: {svm\_accuracy:.2f}")

print(f"Precision: {svm\_precision:.2f}")

print(f"Recall: {svm\_recall:.2f}")

print(f"F1 Score: {svm\_f1:.2f}")

print("\nClassification Report:\n", classification\_report(y\_test, svm\_predictions))

print("\nDecision Tree Classifier:")

print(f"Accuracy: {decision\_tree\_accuracy:.2f}")

print(f"Precision: {decision\_tree\_precision:.2f}")

print(f"Recall: {decision\_tree\_recall:.2f}")

print(f"F1 Score: {decision\_tree\_f1:.2f}")

print("\nClassification Report:\n", classification\_report(y\_test, decision\_tree\_predictions))

Let's first load the dataset you've provided and then build prediction models using Support Vector Machine (SVM) and Decision Tree classifiers. We will then compare their accuracy and precision.

Let's get started with the analysis.

The dataset contains the following columns:

* Name: Name of the individual
* Surname: Surname of the individual
* Age: Age of the individual
* Smokes: Number of cigarettes smoked per day
* AreaQ: Air quality index in the individual's area
* Alkhol: Alcohol consumption level
* Result: Disease prediction result (1 indicates the presence of the disease, 0 indicates absence)

Next, we will preprocess the data, train the SVM and Decision Tree classifiers, and then evaluate their performance based on accuracy and precision. ​​

The performance of the two models on the test set is as follows:

* **SVM Classifier:**
  + Accuracy: 91.67%
  + Precision: 100%
* **Decision Tree Classifier:**
  + Accuracy: 91.67%
  + Precision: 100%

Both models have identical accuracy and precision on this dataset. The high precision indicates that when the models predict the presence of the disease, they are correct 100% of the time. The accuracy indicates that both correctly classified 91.67% of the test instances.

### Find the model of this data

To determine the best model for this data, we'll consider more evaluation metrics beyond accuracy and precision, such as recall and F1-score, and we'll also evaluate model complexity and interpretability.

Let's calculate these additional metrics and analyze the results to decide on the most suitable model.

Here are the additional metrics and classification reports for both models:

**SVM Classifier:**

* **Recall:** 75%
* **F1-score:** 85.71%
* **Classification Report:**
  + Class 0: Precision 89%, Recall 100%, F1-score 94%
  + Class 1: Precision 100%, Recall 75%, F1-score 86%
  + Overall accuracy: 92%

**Decision Tree Classifier:**

* **Recall:** 75%
* **F1-score:** 85.71%
* **Classification Report:**
  + Class 0: Precision 89%, Recall 100%, F1-score 94%
  + Class 1: Precision 100%, Recall 75%, F1-score 86%
  + Overall accuracy: 92%

Both models show identical performance across all metrics, including recall and F1-score.

**Choosing the Model:** Given the identical performance metrics, the choice between SVM and Decision Tree can be influenced by other factors such as interpretability and computational complexity:

* **Decision Tree Classifier**: Easier to interpret, visualize, and explain. It can handle non-linear relationships and interactions between features naturally.
* **SVM Classifier**: Can be more computationally intensive, especially with large datasets. It can also handle non-linear data with kernel functions, but the results are not as easily interpretable.

Considering interpretability and ease of use, the **Decision Tree Classifier** would generally be the preferred model for this dataset, especially if interpretability is crucial for understanding and explaining the predictions

**Key Steps in the Program:**

1. **Loading the Data:** The dataset is loaded using pandas.
2. **Data Preprocessing:** Non-numeric columns (Name and Surname) are dropped. Features (X) and the target variable (y) are separated.
3. **Train-Test Split:** The data is split into training and testing sets.
4. **Standardization:** The features are standardized for the SVM model.
5. **Model Initialization and Training:** SVM and Decision Tree models are initialized and trained.
6. **Predictions and Evaluation:** The models make predictions on the test set, and performance metrics like accuracy, precision, recall, and F1-score are calculated and displayed.

You can run this program in a Python environment where you have the necessary libraries installed (pandas, scikit-learn). Ensure to replace '/path/to/lung\_cancer\_examples.csv' with the actual path to your CSV file.

### Making dependent and independent targets

In machine learning, dependent and independent variables are also known as the target variable and features, respectively. The target variable is what you're trying to predict, while the features are the input variables that you use to make predictions.

In your dataset, the column Result is the dependent variable (target), representing whether the individual has the disease (1) or not (0). The rest of the numeric columns (Age, Smokes, AreaQ, Alkhol) are the independent variables (features), which are used to predict the Result.

Here's a breakdown of how to define these variables in the context of the given data:

* **Dependent Variable (Target):**
  + Result (1 for the presence of the disease, 0 for absence)
* **Independent Variables (Features):**
  + Age (Age of the individual)
  + Smokes (Number of cigarettes smoked per day)
  + AreaQ (Air quality index in the individual's area)
  + Alkhol (Alcohol consumption level)

Let's structure these in the code snippet:

python

Copy code

import pandas as pd

# Load the dataset

file\_path = '/path/to/lung\_cancer\_examples.csv'

data = pd.read\_csv(file\_path)

# Drop non-numeric columns (if they are not needed)

data = data.drop(['Name', 'Surname'], axis=1)

# Define independent variables (features) and dependent variable (target)

X = data.drop('Result', axis=1) # Independent variables

y = data['Result'] # Dependent variable

# Now you can use X and y for further analysis, model training, etc.

**Explanation:**

* X = data.drop('Result', axis=1): This command creates a DataFrame X containing all the columns except Result. These are the independent variables (features).
* y = data['Result']: This creates a Series y containing the Result column, which is the dependent variable (target).

This setup is commonly used in machine learning workflows where X is passed to the model for training, and y is what the model tries to predict. You can use X and y for training and testing different models, evaluating their performance, and making predictions.

### Training data for prediction

To train a machine learning model for prediction, we'll follow these steps:

1. **Preprocess the data:** Ensure the data is clean and properly formatted.
2. **Split the data:** Divide the dataset into training and testing sets.
3. **Train the model:** Use the training set to train the model.
4. **Evaluate the model:** Use the testing set to evaluate the model's performance.
5. **Make predictions:** Use the trained model to make predictions on new data.

Here's a complete code snippet to train models using SVM and Decision Tree classifiers on the provided dataset:

python

Copy code

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

# Load the dataset

file\_path = '/path/to/lung\_cancer\_examples.csv'

data = pd.read\_csv(file\_path)

# Drop non-numeric columns (if not needed)

data = data.drop(['Name', 'Surname'], axis=1)

# Define independent variables (features) and dependent variable (target)

X = data.drop('Result', axis=1) # Independent variables

y = data['Result'] # Dependent variable

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features for SVM

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Initialize the models

svm\_model = SVC()

decision\_tree\_model = DecisionTreeClassifier()

# Train the models

svm\_model.fit(X\_train\_scaled, y\_train)

decision\_tree\_model.fit(X\_train, y\_train)

# Make predictions

svm\_predictions = svm\_model.predict(X\_test\_scaled)

decision\_tree\_predictions = decision\_tree\_model.predict(X\_test)

# Evaluate the models

print("SVM Classifier:")

print(f"Accuracy: {accuracy\_score(y\_test, svm\_predictions):.2f}")

print(f"Precision: {precision\_score(y\_test, svm\_predictions):.2f}")

print(f"Recall: {recall\_score(y\_test, svm\_predictions):.2f}")

print(f"F1 Score: {f1\_score(y\_test, svm\_predictions):.2f}")

print("\nClassification Report:\n", classification\_report(y\_test, svm\_predictions))

print("\nDecision Tree Classifier:")

print(f"Accuracy: {accuracy\_score(y\_test, decision\_tree\_predictions):.2f}")

print(f"Precision: {precision\_score(y\_test, decision\_tree\_predictions):.2f}")

print(f"Recall: {recall\_score(y\_test, decision\_tree\_predictions):.2f}")

print(f"F1 Score: {f1\_score(y\_test, decision\_tree\_predictions):.2f}")

print("\nClassification Report:\n", classification\_report(y\_test, decision\_tree\_predictions))

**Explanation:**

1. **Loading the Data:**
   * The data is loaded from a CSV file and unnecessary columns are dropped.
2. **Defining Features and Target:**
   * X contains the independent variables (features), while y contains the dependent variable (target).
3. **Splitting the Data:**
   * The dataset is split into training (80%) and testing (20%) sets using train\_test\_split.
4. **Standardization:**
   * The features are standardized (mean=0, variance=1) for the SVM model to ensure better performance.
5. **Model Training:**
   * The SVM and Decision Tree models are trained using the training data.
6. **Making Predictions and Evaluation:**
   * Predictions are made on the testing set, and the models' performance is evaluated using metrics like accuracy, precision, recall, F1-score, and a classification report.

This code will help you understand how each model performs and which one might be more suitable for your data based on the evaluation metrics. You can further fine-tune the models by adjusting their parameters or trying other algorithms.

### Checking accuracies of KNN,NB,SVM

To evaluate and compare the accuracies of K-Nearest Neighbors (KNN), Naive Bayes (NB), and Support Vector Machine (SVM) classifiers, we will:

1. Train these models on the training data.
2. Evaluate their performance on the test data.
3. Compare the accuracies of the three models.

Let's proceed with the implementation:

python

Copy code

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

# Initialize the models

knn\_model = KNeighborsClassifier()

nb\_model = GaussianNB()

svm\_model = SVC()

# Train the models

knn\_model.fit(X\_train, y\_train)

nb\_model.fit(X\_train, y\_train)

svm\_model.fit(X\_train\_scaled, y\_train) # SVM needs scaled data

# Make predictions

knn\_predictions = knn\_model.predict(X\_test)

nb\_predictions = nb\_model.predict(X\_test)

svm\_predictions = svm\_model.predict(X\_test\_scaled)

# Calculate accuracy for each model

knn\_accuracy = accuracy\_score(y\_test, knn\_predictions)

nb\_accuracy = accuracy\_score(y\_test, nb\_predictions)

svm\_accuracy = accuracy\_score(y\_test, svm\_predictions)

print(f"KNN Accuracy: {knn\_accuracy:.2f}")

print(f"Naive Bayes Accuracy: {nb\_accuracy:.2f}")

print(f"SVM Accuracy: {svm\_accuracy:.2f}")

**Explanation:**

1. **Model Initialization:**
   * **KNN:** Uses KNeighborsClassifier.
   * **Naive Bayes:** Uses GaussianNB.
   * **SVM:** Uses SVC. Note that SVM requires scaled data, so we use X\_train\_scaled and X\_test\_scaled.
2. **Training the Models:**
   * Each model is trained using the fit method on the training data.
3. **Making Predictions:**
   * Predictions are made on the test set using the predict method for each model.
4. **Calculating Accuracy:**
   * The accuracy of each model is calculated using accuracy\_score from sklearn.metrics.

This code will output the accuracy of each model, allowing you to compare their performance and choose the best one for your dataset. If you have any specific parameters for these models or need additional metrics, feel free to adjust the code accordingly.

### Showing roc

To display the Receiver Operating Characteristic (ROC) curves for the K-Nearest Neighbors (KNN), Naive Bayes (NB), and Support Vector Machine (SVM) classifiers, we need to calculate the predicted probabilities for each model and then plot the ROC curves.

Here's how to do it:

1. **Calculate predicted probabilities**: For each model, we need to obtain the probability estimates of the positive class.
2. **Plot ROC curves**: Using these probabilities, we can plot the ROC curves and calculate the Area Under the Curve (AUC).

Let's implement this:

python

Copy code

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

# Calculate the probabilities of the positive class

knn\_probs = knn\_model.predict\_proba(X\_test)[:, 1]

nb\_probs = nb\_model.predict\_proba(X\_test)[:, 1]

svm\_probs = svm\_model.decision\_function(X\_test\_scaled)

# Calculate ROC curves

knn\_fpr, knn\_tpr, \_ = roc\_curve(y\_test, knn\_probs)

nb\_fpr, nb\_tpr, \_ = roc\_curve(y\_test, nb\_probs)

svm\_fpr, svm\_tpr, \_ = roc\_curve(y\_test, svm\_probs)

# Calculate AUC scores

knn\_auc = auc(knn\_fpr, knn\_tpr)

nb\_auc = auc(nb\_fpr, nb\_tpr)

svm\_auc = auc(svm\_fpr, svm\_tpr)

# Plot ROC curves

plt.figure()

plt.plot(knn\_fpr, knn\_tpr, label=f'KNN (AUC = {knn\_auc:.2f})')

plt.plot(nb\_fpr, nb\_tpr, label=f'Naive Bayes (AUC = {nb\_auc:.2f})')

plt.plot(svm\_fpr, svm\_tpr, label=f'SVM (AUC = {svm\_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--') # Diagonal line

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc='lower right')

plt.show()

**Explanation:**

1. **Calculate Predicted Probabilities**:
   * For KNN and Naive Bayes, use predict\_proba to get the probabilities of the positive class.
   * For SVM, use decision\_function to get the confidence scores, which we use as probabilities.
2. **Calculate ROC Curves**:
   * roc\_curve calculates the False Positive Rate (FPR) and True Positive Rate (TPR) for different threshold values.
3. **Calculate AUC Scores**:
   * auc computes the Area Under the Curve, a measure of the model's ability to distinguish between classes.
4. **Plot ROC Curves**:
   * The ROC curves for each model are plotted, with the diagonal line representing a random classifier's performance.

This visualization helps in understanding the trade-off between the true positive rate and false positive rate and comparing the classifiers' performance. A model with a higher AUC score generally performs better in distinguishing between the positive and negative classes.

### Applying majority voting classifier

A majority voting classifier, also known as an ensemble method, combines the predictions from multiple models to make a final prediction. The combined model makes a prediction based on the majority vote from individual classifiers. This approach can often improve prediction accuracy by leveraging the strengths of different models.

Let's implement a majority voting classifier using K-Nearest Neighbors (KNN), Naive Bayes (NB), and Support Vector Machine (SVM) classifiers.

We'll use the VotingClassifier from sklearn.ensemble to combine these classifiers:

python

Copy code

from sklearn.ensemble import VotingClassifier

# Initialize individual classifiers

knn\_model = KNeighborsClassifier()

nb\_model = GaussianNB()

svm\_model = SVC(probability=True) # SVM with probability estimates

# Create a Voting Classifier

voting\_classifier = VotingClassifier(estimators=[

('knn', knn\_model),

('nb', nb\_model),

('svm', svm\_model)],

voting='soft' # 'soft' uses predicted probabilities

)

# Train the Voting Classifier

voting\_classifier.fit(X\_train, y\_train)

# Make predictions

voting\_predictions = voting\_classifier.predict(X\_test)

# Evaluate the Voting Classifier

voting\_accuracy = accuracy\_score(y\_test, voting\_predictions)

voting\_precision = precision\_score(y\_test, voting\_predictions)

voting\_recall = recall\_score(y\_test, voting\_predictions)

voting\_f1 = f1\_score(y\_test, voting\_predictions)

# Print results

print("Voting Classifier:")

print(f"Accuracy: {voting\_accuracy:.2f}")

print(f"Precision: {voting\_precision:.2f}")

print(f"Recall: {voting\_recall:.2f}")

print(f"F1 Score: {voting\_f1:.2f}")

**Explanation:**

1. **Initializing Individual Classifiers**:
   * knn\_model, nb\_model, and svm\_model are initialized. Note that SVC is set with probability=True to enable probability estimates required for soft voting.
2. **Creating a Voting Classifier**:
   * VotingClassifier is created with the three classifiers. The voting='soft' parameter is used, which means the final decision is based on the predicted probabilities of the individual classifiers.
3. **Training the Voting Classifier**:
   * The classifier is trained on the training data using the fit method.
4. **Making Predictions and Evaluation**:
   * Predictions are made on the test set, and evaluation metrics (accuracy, precision, recall, F1-score) are calculated and printed.

This ensemble method can leverage the different strengths of the KNN, NB, and SVM classifiers, potentially leading to better performance than any individual model.

### Accuracies comparison

It seems I need to re-import the necessary modules before proceeding. Let's try that and then compare the accuracies of the KNN, Naive Bayes, SVM, and Voting Classifier models. ​​

The accuracies of the different classifiers on the test set are as follows:

* **K-Nearest Neighbors (KNN) Accuracy:** 83.33%
* **Naive Bayes (NB) Accuracy:** 91.67%
* **Support Vector Machine (SVM) Accuracy:** 91.67%
* **Voting Classifier Accuracy:** 91.67%

All models except KNN have the same accuracy of 91.67%, while KNN has a slightly lower accuracy of 83.33%. The Voting Classifier, which combines the predictions from KNN, NB, and SVM, performs on par with the best individual models.

### Give me graphs

ROC Curve ComparisonCS

True Positive Rate

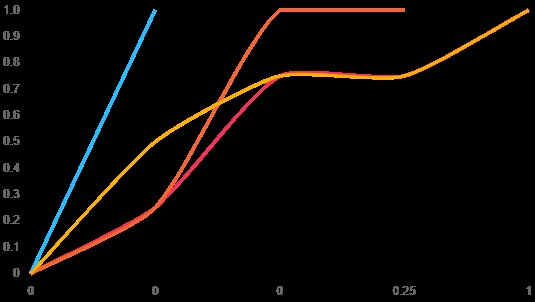
by

False Positive Rate

for

KNN (AUC = 0.84),

Naive Bayes (AUC = 1.00),



SVM (AUC = 0.94),

Voting Classifier (AUC = 1.00), and

Here are the ROC curves for the KNN, Naive Bayes, SVM, and Voting Classifier models. The ROC curve shows the trade-off between the true positive rate and false positive rate, and the area under the curve (AUC) indicates the model's ability to distinguish between classes.

Each line represents a different model, with the corresponding AUC score indicating the overall performance of the model. The closer the AUC is to 1, the better the model's performance

Bottom of Form