Selection of an appropriate dataset

The Titanic dataset is a comprehensive and historical compilation of passenger data from the RMS Titanic's tragic inaugural voyage in April 1912. In data science and machine learning, this dataset is frequently used to demonstrate a number of ideas, including exploratory data analysis, predictive modeling, and data preprocessing. The 891 entries, each of which represents a passenger, include demographic information like age, gender, and class in addition to journey-related factors like ticket price, cabin number, and whether the passenger survived. Predicting survival rates from the available attributes is frequently the main goal of this dataset analysis, which enables the investigation of the variables that influenced survival outcomes. All things considered, the Titanic dataset's historical significance, complexity, and diversity make it an excellent option for both practical and educational applications in the domains of data analysis and machine learning. This allows students to acquire knowledge about real-world data challenges and modeling strategies.

Data Inspected and Loaded

The Titanic dataset is loaded either locally as a CSV file or immediately from an internet source. With data on Titanic passengers including age, gender, class, and survivor status, the dataset is a well-known benchmark in data science.

• Initial Inspection: Utilizing data, the top few rows are displayed.Understanding the data's structure and contents, the kinds of features that are available, and whether any columns are missing or unnecessary could make training the model more difficult are all made easier with the help of head().

2. Data Preprocessing

• Eliminating Irrelevant Columns: o PassengerId, Name, Ticket, and Cabin are examples of columns that don't contain information that is directly relevant to survival.

o A ticket is primarily unique to each passenger and provides no useful pattern for survival prediction; a passenger ID is merely a unique identification for each passenger; and a name is textual and unlikely to aid in survival prediction.

o Although the cabin may contain significant information, it is eliminated to make the dataset simpler because to its high rate of missing values.

• Managing Missing Values:

o Age: Predicting survival depends on age because survival rates varied by age group. In order to maintain central tendencies and guarantee that no data points are lost, we substitute the median age for any missing values in Age.
o Anchored: To fill in the gaps without substantially changing the dataset, we substitute the most prevalent (mode) value in the column for the few missing values in this column.

Categorical variables are encoded:

One of the binary category features is sex, which we translate into numerical values: 0 for males and 1 for females. The fact that machine learning algorithms often work with numerical data enables us to use it directly in model training.

The three categories for this feature are embarked (S, C, and Q for Southampton, Cherbourg, and Queenstown, respectively). These values are mapped to either 0 or 1. Effective utilization of these categorical data by the model is ensured by this encoding.

3. Train-Test Split

• Creating Training and Testing Sets:

o We used an 80-20 split (test_size=0.2) to divide the dataset into training and testing subsets. The testing set assesses how effectively the learned models generalize to fresh, untested data, while the training set is used to fit (or train) the models.

o Random State: By setting a random_state (for example, random_state=42), the split is guaranteed to be reproducible, yielding consistent outcomes each time the code executes.

4. Decision Tree Model

Model Training:

- We initialize a **Decision Tree Classifier** (DecisionTreeClassifier(random_state=42)) and fit it to the training data using dt_model.fit(X_train, y_train). This process involves the model learning from the data by splitting the data at different feature values to maximize survival prediction accuracy.
- Decision trees are known for their ability to capture non-linear relationships and are relatively interpretable, as the model structure resembles a flowchart of decisions based on feature values.

Making Predictions:

- We use the trained model to predict survival on the testing set (dt_predictions = dt_model.predict(X_test)). This provides a set of predicted labels (survived or not) that we can compare to the actual labels in y_test.

Model Evaluation:

- **Accuracy Score**: The accuracy score (computed using accuracy_score(y_test, dt_predictions)) provides the percentage of correct predictions over the total predictions, giving a straightforward measure of model performance.
- **Classification Report**: The classification report (classification_report(y_test, dt_predictions)) breaks down performance by precision, recall, and F1-score for each class (survived or not), which helps assess the balance and detail of the model's accuracy.
- **Confusion Matrix**: This matrix (confusion_matrix(y_test, dt_predictions)) shows the count of true positives, true negatives, false positives, and false negatives, giving insight into which types of errors the model makes.

5. Random Forest Model

Model Training:

- We initialize a **Random Forest Classifier** with 100 trees (RandomForestClassifier(random_state=42, n_estimators=100)) and fit it to the training data. Random forests are an ensemble method that builds multiple decision trees and aggregates their predictions, reducing the overfitting tendency of individual trees.
- During training, each tree in the forest is trained on a random subset of features and a random subset of training samples, which enhances robustness and generalization ability.

Making Predictions:

- Using the trained Random Forest model, we predict survival on the testing set (rf_predictions = rf_model.predict(X_test)), producing a set of predicted labels that we can evaluate against y_test.

Model Evaluation:

- We evaluate the Random Forest model in the same way as the Decision Tree model, using accuracy score, classification report, and confusion matrix. Generally, Random Forests perform better than single Decision Trees due to their ensemble nature, which reduces variance.

This approach gives a comprehensive understanding of the Titanic dataset and provides a clear comparison between a single Decision Tree and a more robust ensemble model (Random Forest).

1. Loading and Taking a First Look at the Data

- First, we bring in the Titanic dataset, which is a famous collection of data on the passengers of the RMS Titanic. This dataset includes details about each passenger, such as their age, gender, travel class, and whether or not they survived.
- To get a sense of what we're working with, we check the first few rows. This is like glancing at a spreadsheet to see if there are any obvious problems, such as missing values or features (columns) that won't help us understand survival.

2. Cleaning and Preparing the Data

Removing Unhelpful Columns:

- Some columns, like PassengerId, Name, Ticket, and Cabin, don't provide useful clues about survival. PassengerId is just a unique identifier, Name is mostly irrelevant to survival prediction, Ticket numbers are unique and therefore hard to analyze, and Cabin has too many missing entries to be useful. So, we drop these columns to keep the dataset manageable.

Filling in Missing Information:

- The Age column has some missing values, which we fill with the median age, representing the middle age range. We do this because age is an important predictor (younger passengers might have had a different survival rate than older ones).
- Embarked (the port where a passenger boarded) has a few missing values as well. We replace these with the most common port, which avoids losing data without skewing results.

Converting Text to Numbers:

- Since models work best with numbers, we convert text columns into numerical values. For example, we turn Sex into 0 for male and 1 for female, and we do something similar with Embarked (where each port is represented by a number). This makes the data machine-friendly without changing its meaning.

3. Splitting Data for Training and Testing

- To build and evaluate our models, we divide the data into two groups: 80% for training and 20% for testing.
- The training data helps the model learn patterns about survival, while the test data lets us see how well the model can predict survival on new, unseen data. We set a random_state for consistency, meaning we'll get the same split each time, making our results easier to reproduce.

4. Building a Decision Tree Model

Training the Model:

- We start with a Decision Tree, which is a model that learns by making a series of yes-or-no splits on the data (for example, "Is the passenger's age below 16?"). The goal is to create a "flowchart" that can help the model make accurate predictions about survival.

Making Predictions and Evaluating the Results:

- Once the tree is trained, we use it to make predictions on the test data. Then we check:
    - **Accuracy**: What percentage of our predictions were correct?
    - **Classification Report**: We look at additional details, like precision (how often the model correctly predicted survival) and recall (how often it found all actual survivors).
    - **Confusion Matrix**: This shows us where the model made mistakes (e.g., predicting survival when the person didn't survive). This breakdown helps us see if the model is over- or under-predicting any outcomes.

5. Building a Random Forest Model

- **Training the Model**:
    - Unlike the single Decision Tree, a Random Forest creates many trees (we set it to 100 here) and combines their predictions. Each tree learns slightly different patterns because it's trained on a different subset of data and features. By averaging across all these trees, the Random Forest can be more accurate and less prone to errors than a single Decision Tree.
- **Making Predictions and Evaluating the Results**:
    - Just like with the Decision Tree, we make predictions and evaluate them. Because a Random Forest is a collection of trees, it often balances out mistakes that a single tree might make. Generally, Random Forests give us a more robust model that's less likely to "overfit" (memorize training data too closely).

```
[1]: pip install pandas numpy scikit-learn
```

```
Requirement already satisfied: pandas in c:\users\madushi\anaconda3\lib\site-packages (2.2.2)
Requirement already satisfied: numpy in c:\users\madushi\anaconda3\lib\site-packages (1.26.4)
Requirement already satisfied: scikit-learn in c:\users\madushi\anaconda3\lib\site-packages (1.4.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\madushi\anaconda3\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\madushi\anaconda3\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\madushi\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: scipy>=1.6.0 in c:\users\madushi\anaconda3\lib\site-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\madushi\anaconda3\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\madushi\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\madushi\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

```
[3]: # Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the Titanic dataset from an online source or local CSV file
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
data = pd.read_csv(url)

# Inspect the data
print(data.head())

# Data Preprocessing
# Drop unnecessary columns
data = data.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'])

# Fill missing values
data['Age'].fillna(data['Age'].median(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

# Convert categorical features to numeric
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})
data['Embarked'] = data['Embarked'].map({'S': 0, 'C': 1, 'Q': 2})
```

```python
# Define features and target variable
X = data.drop(columns=['Survived'])
y = data['Survived']

# Split the data 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)

# Apply Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Make predictions and evaluate the Decision Tree model
dt_predictions = dt_model.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy)
print("Classification Report:\n", classification_report(y_test, dt_predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, dt_predictions))

# Apply Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
rf_model.fit(X_train, y_train)

# Make predictions and evaluate the Random Forest model
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy)
print("Classification Report:\n", classification_report(y_test, rf_predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, rf_predictions))
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
Decision Tree Accuracy: 0.7988826815642458
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.83      0.83       105
           1       0.76      0.76      0.76        74

    accuracy                           0.80       179
   macro avg       0.79      0.79      0.79       179
weighted avg       0.80      0.80      0.80       179

Confusion Matrix:
 [[87 18]
 [18 56]]
```

Critical analysis and discussion

While our original models, which used Decision Trees and Random Forests, gave a strong basis for forecasting passenger survivability, our analysis of the Titanic dataset revealed that there are a number of ways to increase accuracy. First, sophisticated feature engineering could greatly improve model performance; for example, developing additional features like "Family Size" or grouping passengers according to age could provide more in-depth understanding of survival patterns. Furthermore, using more sophisticated algorithms like Gradient Boosting Machines (GBM) or XGBoost could catch complicated patterns that simple models might overlook, while hyperparameter tuning using techniques like Grid Search can improve model settings for improved predictions.

In the future, adding other datasets and investigating feature significance may improve our analysis and offer useful applications, such a survival probability prediction system that operates in real time. Last but not least, it's critical to think about the moral ramifications of our prediction models, guaranteeing equity and openness in their use while putting in place a structure for ongoing learning to keep our model current as new information becomes available. We may increase the efficacy and relevance of our analysis by following these advancements and potential paths forward.