Ml week 2.2

python
Copy code
```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report
import matplotlib.pyplot as plt

# Load the dataset
loan = pd.read_csv('/path/to/your/dataset.csv')  # Replace with the correct
path

# Display basic information and first few rows of the dataset
print("Initial Data Info:")
print(loan.info())
print(loan.head())
```

1. **Import Libraries**: Import necessary libraries for data manipulation (pandas), preprocessing (sklearn.impute, sklearn.preprocessing), model building (sklearn.model_selection, sklearn.tree, sklearn.ensemble), evaluation (sklearn.metrics), and visualization (matplotlib.pyplot).
2. **Load Dataset**: Load the Loan Prediction dataset from a specified path and display its basic information and first few rows to understand the structure of the data.

python
Copy code
```
# Fill missing values for categorical variables with the most frequent value
categorical_cols = ['Gender', 'Dependents', 'Self_Employed',
'Credit_History']
imputer = SimpleImputer(strategy='most_frequent')
loan[categorical_cols] = imputer.fit_transform(loan[categorical_cols])

# Fill missing values for numerical variables with the median value
numerical_cols = ['LoanAmount', 'Loan_Amount_Term']
imputer = SimpleImputer(strategy='median')
loan[numerical_cols] = imputer.fit_transform(loan[numerical_cols])
```

3. **Handle Missing Values**:
   - For categorical columns (`Gender`, `Dependents`, `Self_Employed`, `Credit_History`), fill missing values with the most frequent value (mode) using `SimpleImputer`.
   - For numerical columns (`LoanAmount`, `Loan_Amount_Term`), fill missing values with the median value using `SimpleImputer`.

python
Copy code

```
# Encode categorical variables
label_encoders = {}
for column in loan.select_dtypes(include=['object']).columns:
    if column != 'Loan_ID':
        label_encoders[column] = LabelEncoder()
        loan[column] = label_encoders[column].fit_transform(loan[column])
```

4. **Encode Categorical Variables**:
   - Encode all categorical variables (excluding `Loan_ID`) to numerical values using `LabelEncoder` to prepare the data for machine learning algorithms.

```
python
Copy code
# Display the processed data
print("\nProcessed Data Info:")
print(loan.info())
print(loan.head())
```

5. **Display Processed Data**: Show the basic information and first few rows of the processed data to ensure that the missing values are handled and categorical variables are encoded correctly.

```
python
Copy code
# Separate the features and target variable (using Credit_History as a proxy
target for demonstration)
X = loan.drop(columns=['Loan_ID', 'Credit_History'])
y = loan['Credit_History']
```

6. **Feature and Target Separation**:
   - Separate the features (`X`) and the target variable (`y`). Here, `Loan_ID` is dropped as it is just an identifier, and `Credit_History` is used as the target variable for this demonstration.

```
python
Copy code
# 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)
```

7. **Train-Test Split**:
   - Split the data into training and testing sets using `train_test_split` with a test size of 20% and a fixed random state for reproducibility.

```
python
Copy code
# Train the decision tree classifier
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)
```

8. **Train Decision Tree**:

- Initialize and train a `DecisionTreeClassifier` on the training data (`X_train`, `y_train`).

```
python
Copy code
# Visualize the decision tree
plt.figure(figsize=(20,10))
plot_tree(decision_tree, feature_names=X.columns, class_names=['0', '1'],
filled=True, rounded=True)
plt.show()
```

9. **Visualize Decision Tree**:
   - Visualize the trained decision tree using `plot_tree` to understand its structure and decision rules.

```
python
Copy code
# Train the random forest classifier
random_forest = RandomForestClassifier(random_state=42)
random_forest.fit(X_train, y_train)
```

10. **Train Random Forest**:
    - Initialize and train a `RandomForestClassifier` on the training data (`X_train`, `y_train`).

```
python
Copy code
# Train a randomized decision tree (single tree in random forest with
bootstrap=False)
random_tree = RandomForestClassifier(n_estimators=1, bootstrap=False,
random_state=42)
random_tree.fit(X_train, y_train)

# Extract the single tree from the random forest
randomized_decision_tree = random_tree.estimators_[0]
```

11. **Train Randomized Decision Tree**:
    - Initialize and train a `RandomForestClassifier` with only one estimator (`n_estimators=1`) and bootstrap disabled (`bootstrap=False`). This creates a single decision tree with randomized features.
    - Extract the single tree from the trained `RandomForestClassifier`.

```
python
Copy code
# Visualize the randomized decision tree
plt.figure(figsize=(20,10))
plot_tree(randomized_decision_tree, feature_names=X.columns,
class_names=['0', '1'], filled=True, rounded=True)
plt.show()
```

12. **Visualize Randomized Decision Tree**:

- Visualize the extracted randomized decision tree using `plot_tree` to understand its structure and decision rules.

```python
Copy code
# Make predictions with all models
y_pred_tree = decision_tree.predict(X_test)
y_pred_forest = random_forest.predict(X_test)
y_pred_random_tree = random_tree.predict(X_test)
```

13. **Make Predictions**:
    - Use the trained models (decision tree, random forest, and randomized decision tree) to make predictions on the test data (`X_test`).

```python
Copy code
# Evaluate performance of all models
metrics = {
    'Accuracy': accuracy_score,
    'Precision': precision_score,
    'Recall': recall_score,
    'F1 Score': f1_score
}

performance_tree = {name: metric(y_test, y_pred_tree) for name, metric in
metrics.items()}
performance_forest = {name: metric(y_test, y_pred_forest) for name, metric in
metrics.items()}
performance_random_tree = {name: metric(y_test, y_pred_random_tree) for name,
metric in metrics.items()}
```

14. **Evaluate Performance**:
    - Define a dictionary of evaluation metrics (accuracy, precision, recall, F1 score).
    - Calculate these metrics for each model's predictions and store them in dictionaries (`performance_tree`, `performance_forest`, `performance_random_tree`).

```python
Copy code
# Generate classification reports
report_tree = classification_report(y_test, y_pred_tree, target_names=['0',
'1'])
report_forest = classification_report(y_test, y_pred_forest,
target_names=['0', '1'])
report_random_tree = classification_report(y_test, y_pred_random_tree,
target_names=['0', '1'])
```

15. **Generate Classification Reports**:
    - Generate detailed classification reports for each model using `classification_report`, which includes precision, recall, F1 score, and support for each class.

```
python
```

```
Copy code
# Display performance comparison
print("\nDecision Tree Performance:")
for metric, value in performance_tree.items():
    print(f"{metric}: {value:.4f}")

print("\nRandom Forest Performance:")
for metric, value in performance_forest.items():
    print(f"{metric}: {value:.4f}")

print("\nRandomized Decision Tree Performance:")
for metric, value in performance_random_tree.items():
    print(f"{metric}: {value:.4f}")

print("\nClassification Report for Decision Tree:\n", report_tree)
print("Classification Report for Random Forest:\n", report_forest)
print("Classification Report for Randomized Decision Tree:\n",
report_random_tree)
```

16. **Display Performance Comparison**:
    - Print the evaluation metrics and classification reports for each model to compare their performance and understand their strengths and weaknesses.