This task involves implementing and comparing decision tree and random forest classifiers for predicting loan approvals using the Loan Prediction Dataset from Kaggle. The process includes data cleaning, feature encoding, model training, and evaluation using accuracy, F1-score, precision, and recall. Visualization of the decision tree and comparison of model performance metrics provide insights into their effectiveness.

WEEK 2

SUPERVISED LEARNING

ML-WEEK 2.2

Noor Ul Ain

Task 2.2: Decision Trees and Random Forests

OBJECTIVE

Learn to implement tree-based methods for classification and compare their performance.

DATASET

Loan Prediction Dataset from Kaggle. This dataset involves predicting loan approval based on attributes like gender, marital status, income, and loan amount.

Link to dataset

https://www.kaggle.com/code/bhavikbb/loan-prediction dataset/output

ACTIVITIES

1. Model Implementation
- Build a decision tree model and visualize the tree.
- Implement a random forest to improve model performance and reduce overfitting.

2. Performance Comparison:
- Compare the performance of the decision tree and random forest using accuracy, F1-score, and other relevant metrics.

Libraries Used

Pandas: A powerful data manipulation and analysis library for Python, providing data structures like DataFrames to handle and process structured data efficiently.

Matplotlib: A plotting library for Python and its numerical mathematics extension NumPy. It provides a comprehensive set of tools for creating static, animated, and interactive visualizations.

Seaborn: A Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Scikit-learn: A machine learning library for Python that includes simple and efficient tools for data mining and data analysis, supporting various supervised and unsupervised learning algorithms.

Explanation

Chunk 1: Import Libraries

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score,
confusion_matrix, classification_report

%matplotlib inline
```

This chunk imports the necessary libraries for data manipulation, visualization, and machine learning. It includes libraries like Pandas, Matplotlib, Seaborn, and Scikit-learn.

Chunk 2: Load and Explore Dataset

```
train_data = pd.read_csv("../input/train.csv")

print(train_data.head())

print(train_data.describe())
```

This chunk loads the dataset into a Pandas DataFrame and prints the first few rows and a summary of the dataset, providing an initial understanding of the data.

Chunk 3: Data Visualization

```
# Applicant Income Histogram
train_data['ApplicantIncome'].hist(bins=70, grid=False)
plt.title('Applicant Income Distribution')
plt.xlabel('Applicant Income')
plt.ylabel('Frequency')
plt.show()

# Applicant Income Boxplot
train_data.boxplot(column='ApplicantIncome')
plt.title('Applicant Income Boxplot')
plt.show()

# Applicant Income Boxplot by Education
train_data.boxplot(column='ApplicantIncome', by='Education', grid=False)
plt.title('Applicant Income Boxplot by Education')
plt.show()

# Loan Amount Histogram
train_data['LoanAmount'].hist(bins=100, grid=False)
plt.title('Loan Amount Distribution')
plt.xlabel('Loan Amount')
plt.ylabel('Frequency')
plt.show()

# Credit History Bar Plot
temp = train_data['Credit_History'].value_counts(ascending=True)
temp.plot(kind='bar')
```

```
plt.title('Credit History Distribution')

plt.xlabel('Credit History')

plt.ylabel('Frequency')

plt.show()
```

This chunk visualizes various features of the dataset using histograms, boxplots, and bar plots. It helps in understanding the distributions and relationships within the data.

Chunk 4: Data Cleaning

```
# Check for missing values

print(train_data.apply(lambda x: sum(x.isnull()), axis=0))


# Fill missing values

train_data['LoanAmount'].fillna(train_data['LoanAmount'].mean(), inplace=True)

train_data['Self_Employed'].fillna('No', inplace=True)

train_data['Gender'].fillna(train_data['Gender'].mode()[0], inplace=True)

train_data['Married'].fillna(train_data['Married'].mode()[0], inplace=True)

train_data['Dependents'].fillna(train_data['Dependents'].mode()[0], inplace=True)

train_data['Loan_Amount_Term'].fillna(train_data['Loan_Amount_Term'].mode()[0],
inplace=True)

train_data['Credit_History'].fillna(train_data['Credit_History'].mode()[0], inplace=True)


# Verify no missing values remain

print(train_data.apply(lambda x: sum(x.isnull()), axis=0))
```

This chunk addresses missing values in the dataset by filling them with appropriate values such as the mean or mode. It ensures the dataset is complete for model training.

Chunk 5: Data Encoding

```
var_mod = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area',
'Loan_Status']

le = LabelEncoder()

for i in var_mod:

    train_data[i] = le.fit_transform(train_data[i])

print(train_data.head())
```

This chunk encodes categorical variables into numerical values using LabelEncoder, making them suitable for machine learning algorithms.

Chunk 6: Define Features and Target Variable

```
X = train_data[['Credit_History', 'Gender', 'Married', 'Education']]

y = train_data['Loan_Status']
```

This chunk defines the features (X) and the target variable (y) for model training. The features include Credit_History, Gender, Married, and Education.

Chunk 7: Model Implementation: Decision Tree

```
dt_model = DecisionTreeClassifier()

dt_model.fit(X, y)

dt_predictions = dt_model.predict(X)


# Evaluate Decision Tree

dt_accuracy = accuracy_score(y, dt_predictions)

dt_f1 = f1_score(y, dt_predictions)

dt_precision = precision_score(y, dt_predictions)

dt_recall = recall_score(y, dt_predictions)


print("Decision Tree Performance:")

print(f"Accuracy: {dt_accuracy}")
print(f"F1 Score: {dt_f1}")

print(f"Precision: {dt_precision}")

print(f"Recall: {dt_recall}")

print(confusion_matrix(y, dt_predictions))

print(classification_report(y, dt_predictions))


# Visualize Decision Tree

plt.figure(figsize=(20,10))

plot_tree(dt_model, filled=True, feature_names=['Credit_History', 'Gender', 'Married',
'Education'], class_names=['No', 'Yes'])

plt.show()
```

This chunk implements the decision tree model, trains it on the dataset, makes predictions, and evaluates its performance using various metrics. It also visualizes the decision tree.

Chunk 8: Model Implementation: Random Forest

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

rf_model.fit(X, y)

rf_predictions = rf_model.predict(X)


# Evaluate Random Forest

rf_accuracy = accuracy_score(y, rf_predictions)

rf_f1 = f1_score(y, rf_predictions)

rf_precision = precision_score(y, rf_predictions)

rf_recall = recall_score(y, rf_predictions)


print("Random Forest Performance:")

print(f"Accuracy: {rf_accuracy}")

print(f"F1 Score: {rf_f1}")
print(f"Precision: {rf_precision}")

print(f"Recall: {rf_recall}")

print(confusion_matrix(y, rf_predictions))

print(classification_report(y, rf_predictions))
```

This chunk implements the random forest model, trains it on the dataset, makes predictions, and evaluates its performance using the same metrics as the decision tree.

Chunk 9: Performance Comparison

```
performance_comparison = pd.DataFrame({

    'Model': ['Decision Tree', 'Random Forest'],

    'Accuracy': [dt_accuracy, rf_accuracy],

    'F1 Score': [dt_f1, rf_f1],

    'Precision': [dt_precision, rf_precision],

    'Recall': [dt_recall, rf_recall]

})

print(performance_comparison)
```

This final chunk creates a DataFrame to compare the performance metrics of the decision tree and random forest models, providing a clear comparison of their effectiveness.