Experiment 06

Aim:

Perform Classification modelling
a. Choose classifier for classification problem.
b. Evaluate the performance of classifier.

- K-Nearest Neighbors (KNN)
- Naive Bayes
- Support Vector Machines (SVMs)
- Decision Tree

Theory:

Decision Tree:
The Decision Tree classifier builds a model by recursively splitting the data based on feature values, creating a tree where each node represents a decision rule and each leaf a class label. This approach is highly interpretable, as the decision rules can be easily visualized and understood.

K-Nearest Neighbors (KNN):
KNN classifies a new instance by finding the k closest training examples based on a distance metric (typically Euclidean distance) and assigning the majority class among these neighbors. It is a non-parametric and intuitive method that performs well when features are properly scaled.

Naive Bayes:
Naive Bayes uses Bayes' theorem with the strong assumption that all features are conditionally independent given the class label. This probabilistic classifier is computationally efficient and performs robustly in high-dimensional settings, despite its simplicity.

Support Vector Machines (SVM):
SVM finds the optimal hyperplane that separates classes by maximizing the margin between them, and it can handle non-linear boundaries through the use of kernel functions. It is especially effective in high-dimensional spaces and tends to offer robust performance with appropriate parameter tuning.

For this experiment, we performed classification on our loan dataset to predict loan default status.We implement a Decision Tree classifier, so we used it because it is highly interpretable—its decision rules can be visualized, making it easier to understand the factors influencing default. Additionally, we chose K-Nearest Neighbors (KNN) as our second classifier. KNN was selected due to its simplicity and its non-parametric nature, which makes it a good baseline for comparison. Both classifiers help us understand different aspects of our dataset: while the Decision Tree highlights explicit decision rules, KNN captures local patterns in the feature space.

Data Description

Our loan dataset consists of over 250,000 records and 18 columns, containing a mixture of numerical and categorical features. Key numerical attributes include Age, Income, LoanAmount, CreditScore, MonthsEmployed, NumCreditLines, InterestRate, LoanTerm, and DTIRatio, while important categorical variables include Education, EmploymentType, MaritalStatus, HasMortgage, HasDependents, LoanPurpose, and HasCoSigner. The target variable for classification is “Default,” a binary indicator where 0 represents non-default and 1 indicates default.

In preprocessing, we verified that there were no missing values in most columns; however, we removed rows with missing values in 'HasCoSigner' and 'Default'. Categorical variables were encoded (e.g., using LabelEncoder), and numerical features were standardized to ensure consistent scaling. These steps prepared our dataset for effective classification modeling.

Implementation

1. **Data Preparation:**
    - Load the preprocessed loan dataset.

```
missing_values = df.isnull().sum()
print("Missing values in each column:\n", missing_values)
```

```
Missing values in each column:
 LoanID                     0
Age                         0
Income                      0
LoanAmount                  0
CreditScore                 0
MonthsEmployed              0
NumCreditLines              0
InterestRate                0
LoanTerm                    0
DTIRatio                    0
Education                   0
EmploymentType              0
MaritalStatus               0
HasMortgage                 0
HasDependents               0
LoanPurpose                 0
HasCoSigner                 1
Default                     1
Education_encoded           0
EmploymentType_encoded      0
NewLoanScore                0
ExtraNF                     0
dtype: int64
```

```
[8]  df = df.dropna(subset=['HasCoSigner', 'Default'])
```

- Verify that missing values in ‘HasCoSigner’ and ‘Default’ have been removed and that categorical features (e.g., Education, EmploymentType) have been encoded.
- Standardize numerical features using StandardScaler.

```
le = LabelEncoder()
df['Education_encoded'] = le.fit_transform(df['Education'])
df['EmploymentType_encoded'] = le.fit_transform(df['EmploymentType'])

features = ['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed',
            'NumCreditLines', 'InterestRate', 'LoanTerm', 'DTIRatio',
            'Education_encoded', 'EmploymentType_encoded']
target = 'Default'


X = df[features]
y = df[target]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=39)
```

2. **Data Splitting:**

- Split the dataset into training (70%) and testing (30%) sets, with the target variable being ‘Default’.

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=39)
```

3. **Classifier Setup and Training:**

- **Decision Tree:**
  - Train a Decision Tree classifier with a limited maximum depth (e.g., max_depth=3) to ensure interpretability.
  - Use export_text() to extract and display decision rules.
  - Plot the pruned tree for visualization.

```
from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier(max_depth=3,random_state=42)
dt_clf.fit(X_train, y_train)
y_pred_dt = dt_clf.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt))
```

```
Decision Tree Accuracy: 0.8843537414965986
Decision Tree Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.94      5720
         1.0       0.00      0.00      0.00       748

    accuracy                           0.88      6468
   macro avg       0.44      0.50      0.47      6468
weighted avg       0.78      0.88      0.83      6468
```

- **K-Nearest Neighbors (KNN):**
  - Train a KNN classifier (e.g., using n_neighbors=5) on the standardized training data.
  - Optionally, reduce data dimensionality using PCA for visualizing the decision boundary.

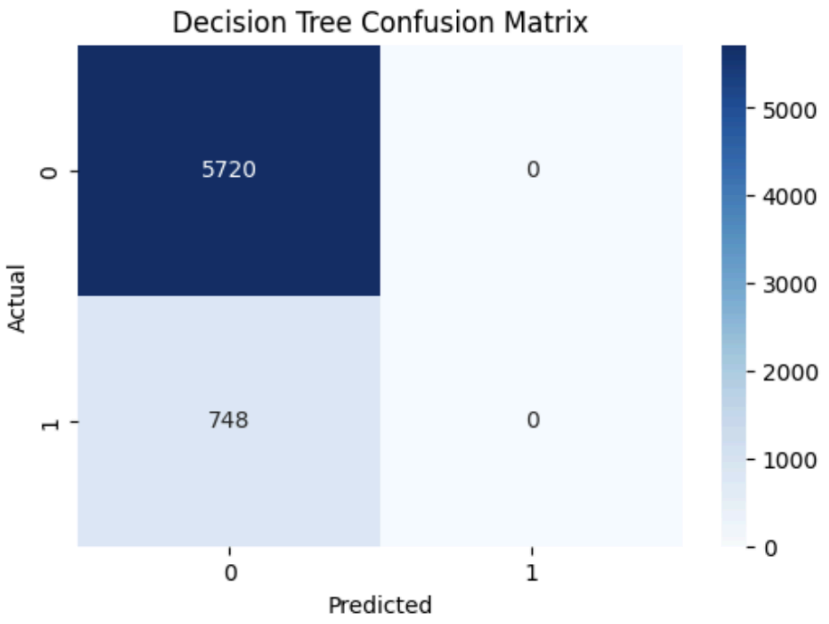
```
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train, y_train)
y_pred_knn = knn_clf.predict(X_test)

print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("KNN Classification Report:")
print(classification_report(y_test, y_pred_knn))
```

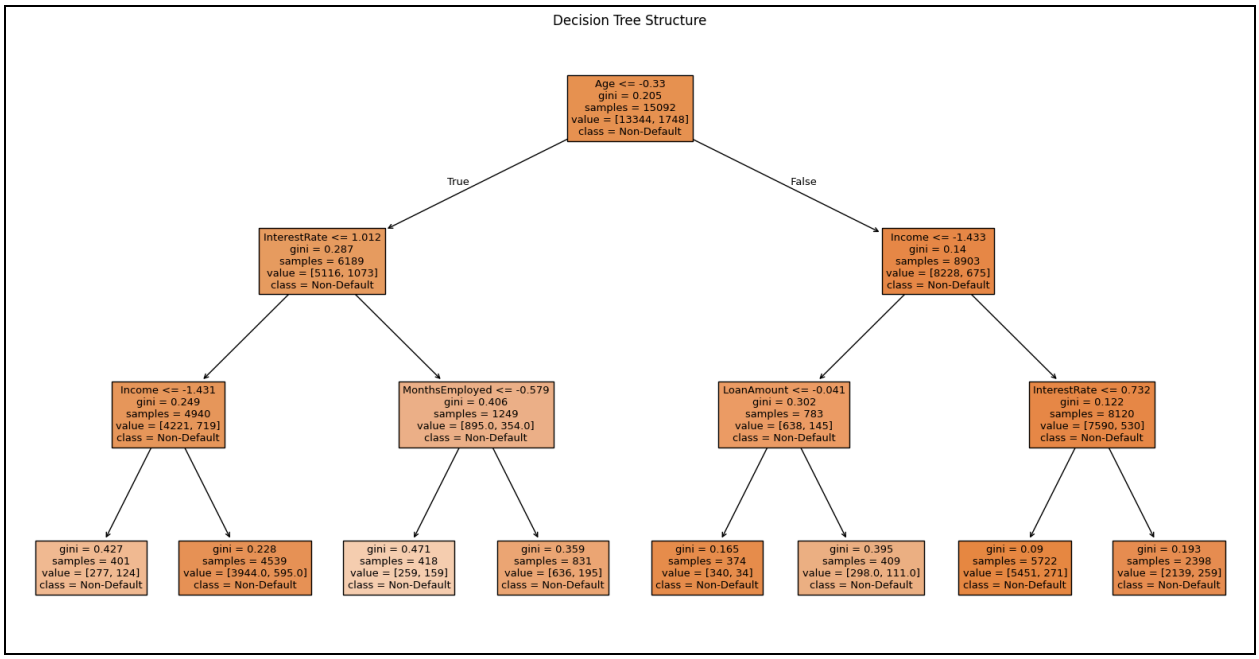
```
KNN Accuracy: 0.8749226963512677
KNN Classification Report:
              precision    recall  f1-score   support

         0.0       0.89      0.98      0.93      5720
         1.0       0.28      0.05      0.09       748

    accuracy                           0.87      6468
   macro avg       0.59      0.52      0.51      6468
weighted avg       0.82      0.87      0.84      6468
```

4. **Model Evaluation:**

   - Evaluate each classifier using metrics like accuracy, precision, recall, and F1-score.
   - Display confusion matrices for both models and capture any relevant plots for further analysis.

```
cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(6,4))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues')
plt.title('Decision Tree Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

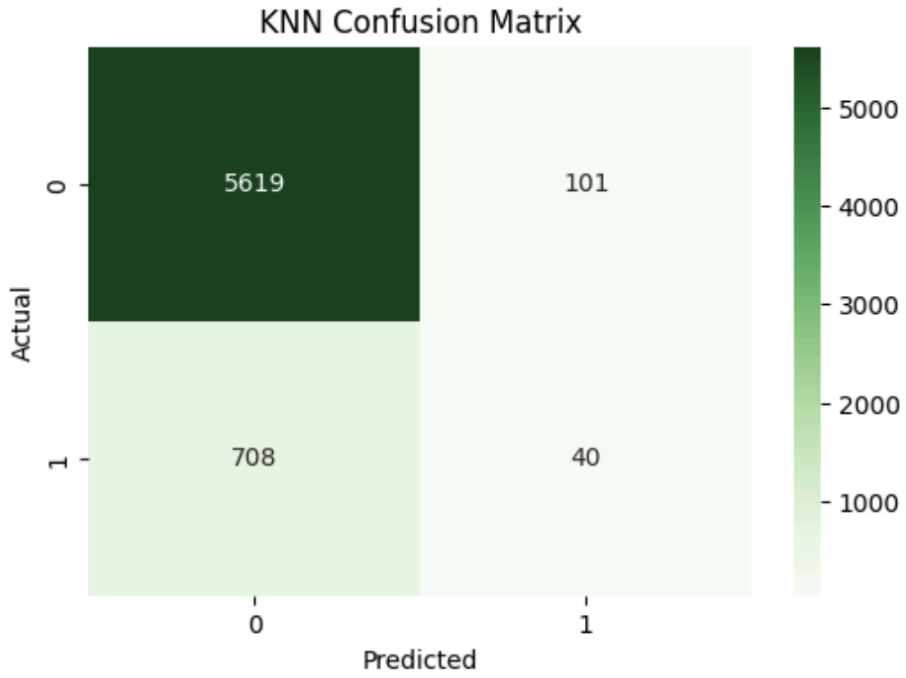





```
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
import matplotlib.pyplot as plt
r = export_text(dt_clf, feature_names=features)
print("Decision Tree Rules:\n", r)
```

```
Decision Tree Rules:
 |--- Age <= -0.33
|   |--- InterestRate <= 1.01
|   |   |--- Income <= -1.43
|   |   |   |--- class: 0.0
|   |   |--- Income >  -1.43
|   |   |   |--- class: 0.0
|   |--- InterestRate >  1.01
|   |   |--- MonthsEmployed <= -0.58
|   |   |   |--- class: 0.0
|   |   |--- MonthsEmployed >  -0.58
|   |   |   |--- class: 0.0
|--- Age >  -0.33
|   |--- Income <= -1.43
|   |   |--- LoanAmount <= -0.04
|   |   |   |--- class: 0.0
|   |   |--- LoanAmount >  -0.04
|   |   |   |--- class: 0.0
|   |--- Income >  -1.43
|   |   |--- InterestRate <= 0.73
|   |   |   |--- class: 0.0
|   |   |--- InterestRate >  0.73
|   |   |   |--- class: 0.0
```

```
cm_knn = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(6,4))
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Greens')
plt.title('KNN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Conclusion:

In our classification experiments, we used the Decision Tree and K-Nearest Neighbors (KNN) classifiers to predict loan default status. The Decision Tree model, with a maximum depth set to 3 for clarity, achieved an accuracy of about 88.4%. However, while it performed well on the majority (non-default) class, its performance on the default class was very poor, as seen in its classification report and confusion matrix.

The KNN classifier achieved an accuracy of approximately 87.5%. Similar to the Decision Tree, KNN exhibited strong performance for predicting non-default loans but struggled with the minority (default) class, resulting in low precision and recall for defaults.