NAME: Sahil Motiramani DIV: D15C ROLL NO:35

AIDS Lab Exp 06

Aim: Classification modelling– Use a classification algorithm and evaluate the performance.

a) Choose a classifier for a classification problem.

b) Evaluate the performance of the classifier.

Perform Classification using (2 of) the below 4 classifiers on the same dataset which you have used for experiment no 5:

K-Nearest Neighbors (KNN)

Naive Bayes

Support Vector Machines (SVMs)

Decision Tree

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, non-parametric, instance-based learning algorithm used for classification and regression. It classifies a new data point based on the majority class among its k nearest neighbors in the feature space. The algorithm relies on distance metrics (e.g., Euclidean distance) to find the closest data points.

Naive Bayes

Naïve Bayes is a probabilistic classification algorithm based on Bayes' Theorem, assuming that all features are independent of each other (the "naïve" assumption). Despite this simplification, it performs well in many real-world applications such as spam detection and sentiment analysis. It works by calculating the probability of each class given the input features and selecting the class with the highest probability.

Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are powerful supervised learning models used for classification and regression. SVMs aim to find the optimal hyperplane that best separates different classes in the dataset by maximizing the margin between data points. It supports both linear and non-linear classification using kernel functions (e.g., polynomial, RBF). SVMs are effective in high-dimensional spaces and work well for complex datasets, but they can be computationally intensive, especially for large datasets.

Decision Tree

A Decision Tree is a supervised learning algorithm that splits data into branches based on feature values, forming a tree-like structure. Each internal node represents a decision based on a feature, and each leaf node represents a class label. The model uses criteria like Gini impurity or entropy to decide the best splits. Decision Trees are easy to interpret and handle both numerical and categorical data, but they can overfit the training data unless pruned or regularized.

NAME: Sahil Motiramani DIV: D15C ROLL NO:35

Dataset: Bank Churn Modelling

The dataset used in this experiment is related to bank churn prediction, where the goal is to analyze factors affecting whether a customer will churn (leave the bank) or not. The dataset contains variables such as:

- Customer ID (Unique Identifier)
- Credit Score
- Geography
- Gender
- Age
- Tenure
- Balance
- Number of Products
- Has Credit Card
- Is Active Member
- Estimated Salary
- Exited (Target variable: 1 if customer churned, 0 otherwise)

Step 1:
This step involves importing necessary Python libraries such as Pandas for data manipulation, NumPy for numerical operations, and visualization libraries like Matplotlib and Seaborn. The dataset, Churn_Modelling.csv, is loaded into a Pandas DataFrame. Feature selection is performed by choosing relevant columns (CreditScore, Age, Tenure, Balance, etc.), and the target variable (Exited) is identified. Missing values are handled using the SimpleImputer with a median strategy to maintain data consistency. Standardization is applied using StandardScaler to bring all numerical features to a uniform scale, ensuring that features with larger magnitudes do not dominate the model.

```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
file_path = "/content/Churn_Modelling.csv"
df = pd.read_csv(file_path)

# Select features and target
X = df[['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']]
y = df['Exited']  # Target variable (1 = Exited, 0 = Not Exited)

# Handle missing values by filling with the median
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Check for remaining NaN values (should be none)
print(f"Missing values in X_train: {np.isnan(X_train).sum()}")
print(f"Missing values in X_test: {np.isnan(X_test).sum()}")
```

```
Missing values in X_train: 0
Missing values in X_test: 0
```

After preprocessing, the dataset is split into training (80%) and testing (20%) sets using train_test_split. Checking for missing values in X_train and X_test ensures that no NaN values remain, confirming that missing data handling was effective. Standardization ensures that features like CreditScore and EstimatedSalary are on the same scale, which improves model performance by preventing bias due to differing feature magnitudes.

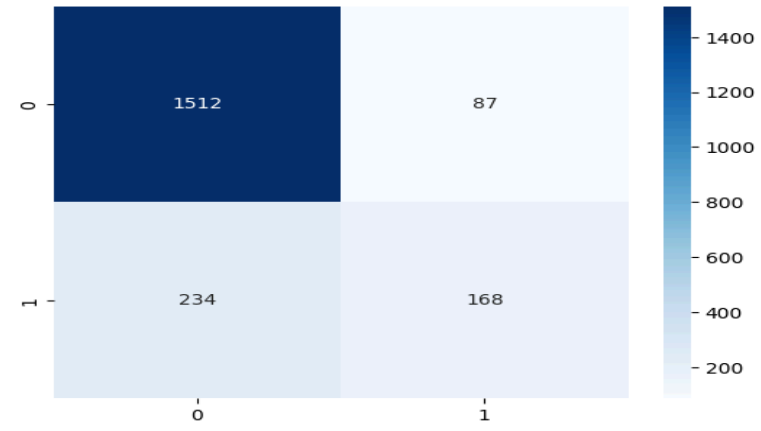
Step 2: Implementing K-Nearest Neighbors (KNN) Classifier

The K-Nearest Neighbors (KNN) algorithm is a non-parametric, instance-based learning method used for classification. It classifies a data point based on the majority class of its k nearest neighbors in the feature space. Here, we initialize a KNN classifier with k=5, meaning it considers the five closest points when making predictions. The model is trained using the fit() method on X_train and y_train, and predictions are made on the test set. Accuracy is measured

using accuracy_score, while classification_report provides precision, recall, and F1-score insights. The confusion matrix, visualized using Seaborn, helps analyze misclassifications.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Initialize KNN classifier with k=5
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predictions
y_pred_knn = knn.predict(X_test)

# Calculate accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)

# Evaluation
print(f"KNN Accuracy: {accuracy_knn:.4f}")
print("KNN Classification Report:")
print(classification_report(y_test, y_pred_knn))
print("Confusion Matrix:")
sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True, fmt="d", cmap="Blues")
plt.show()
```

```
KNN Accuracy: 0.8396
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.95      0.90      1599
           1       0.66      0.42      0.51       402

    accuracy                           0.84      2001
   macro avg       0.76      0.68      0.71      2001
weighted avg       0.82      0.84      0.83      2001

Confusion Matrix:
```



The KNN model achieved an accuracy of 0.8396, indicating its effectiveness in classifying customer churn. The classification report showed precision, recall, and F1-score values for both classes (Exited = 1 and Not Exited = 0). The confusion matrix revealed that 1512 true positives and 168 true negatives were correctly classified.This analysis helps understand where the model performs well and where it struggles in distinguishing churned and non-churned customers.

Step 3: Implementing Naïve Bayes Classifier

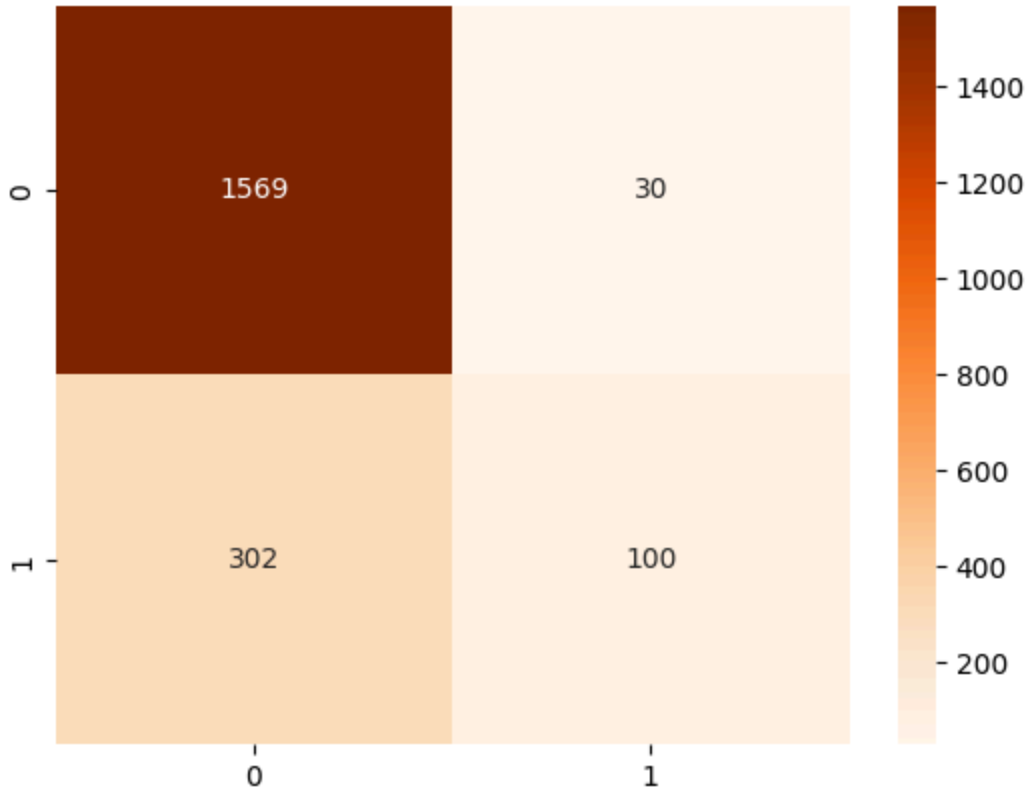
NAME: Sahil Motiramani DIV: D15C ROLL NO:35

The Naïve Bayes classifier is a probabilistic machine learning algorithm based on Bayes' Theorem, assuming independence between features. Here, we use the GaussianNB model, which is suitable for continuous numerical features and assumes a normal distribution. The classifier is trained on X_train and y_train, and predictions are made on X_test. The model's performance is evaluated using accuracy, precision, recall, F1-score, and a confusion matrix, which provides insights into classification errors.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Initialize Naive Bayes classifier
nb = GaussianNB()
nb.fit(X_train, y_train)

# Predictions
y_pred_nb = nb.predict(X_test)

# Calculate accuracy
accuracy_nb = accuracy_score(y_test, y_pred_nb)

# Evaluation
print(f"Naive Bayes Accuracy: {accuracy_nb:.4f}")
print("Naive Bayes Classification Report:")
print(classification_report(y_test, y_pred_nb))
print("Confusion Matrix:")
sns.heatmap(confusion_matrix(y_test, y_pred_nb), annot=True, fmt="d", cmap="Oranges")
plt.show()
```

NAME: Sahil Motiramani DIV: D15C ROLL NO:35

```
Naive Bayes Accuracy: 0.8341
Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.98      0.90      1599
           1       0.77      0.25      0.38       402

    accuracy                           0.83      2001
   macro avg       0.80      0.61      0.64      2001
weighted avg       0.82      0.83      0.80      2001

Confusion Matrix:
```



The Naïve Bayes model achieved an accuracy of 0.8341, reflecting its effectiveness in predicting customer churn. The classification report provided precision, recall, and F1-score values, showing that class performed better. The confusion matrix showed 1569 true positives and 100 true negatives. These results help assess how well the model differentiates between churned and non-churned customers.

Step 4: Implementing Decision Tree Classifier

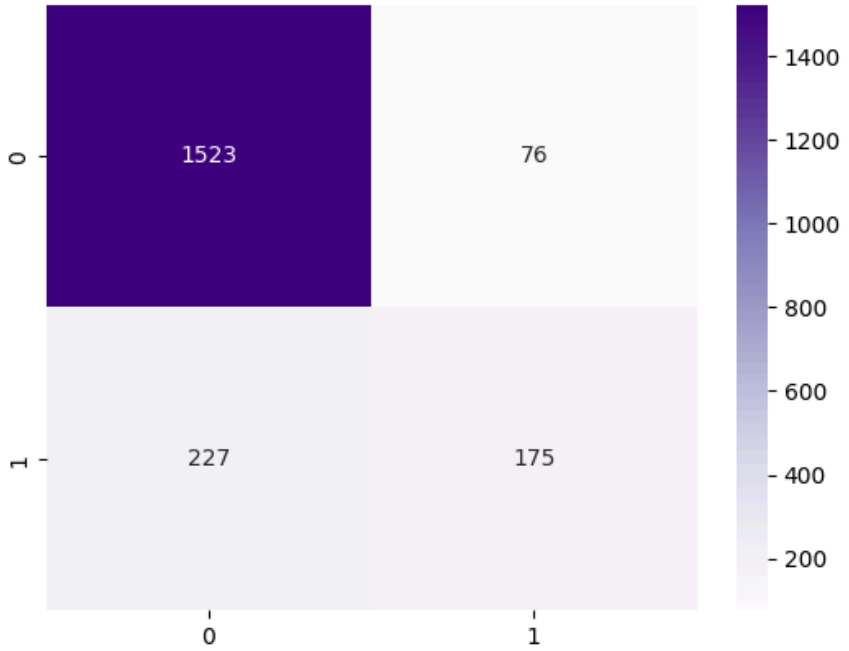
The Decision Tree algorithm is a supervised learning method used for classification and regression tasks. It recursively splits the dataset based on feature values, aiming to create pure subsets using a selected criterion (e.g., gini impurity or entropy). Here, we initialize a Decision Tree with max_depth=3, limiting the depth to prevent overfitting. The model is trained using fit(), and predictions are made on X_test. Accuracy, classification metrics, and a confusion matrix are used for evaluation. Additionally, plot_tree() provides a visual representation of how the model makes decisions based on feature splits.

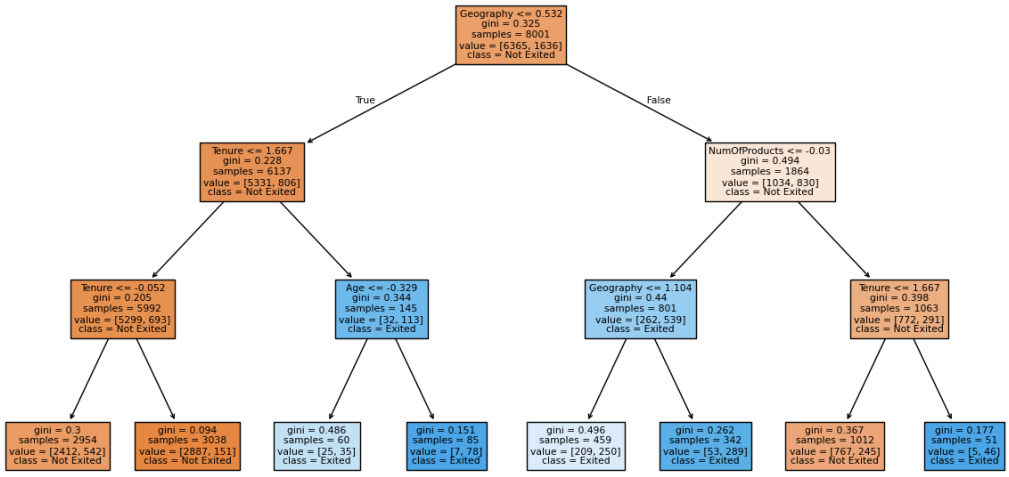
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns

# Initialize Decision Tree classifier with max_depth=3
dt = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
dt.fit(X_train, y_train)

# Predictions
y_pred_dt = dt.predict(X_test)

# Calculate accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)

# Evaluation
print(f"Decision Tree Accuracy: {accuracy_dt:.4f}")  # Display accuracy with 4 decimal places
print("Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt))
print("Confusion Matrix:")
sns.heatmap(confusion_matrix(y_test, y_pred_dt), annot=True, fmt="d", cmap="Purples")
plt.show()

# Visualizing the decision tree
feature_names = ['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
                 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']  # Adjust based on your dataset

plt.figure(figsize=(15, 8))
plot_tree(dt, feature_names=feature_names, class_names=['Not Exited', 'Exited'], filled=True)
plt.show()
```

NAME: Sahil Motiramani DIV: D15C ROLL NO:35

```
Decision Tree Accuracy: 0.8486
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.95      0.91      1599
           1       0.70      0.44      0.54       402

    accuracy                           0.85      2001
   macro avg       0.78      0.69      0.72      2001
weighted avg       0.84      0.85      0.83      2001
```

Confusion Matrix:





The Decision Tree model achieved an accuracy of0.8468, indicating its ability to classify customer churn. The classification report showed precision, recall, and F1-score, with class performing better. The confusion matrix revealed 1523 true positives and 175 true negatives.The tree visualization highlighted key decision-making features, with Geography being the most influential in predicting churn.

Step 5: Comparing Model Accuracies

Model evaluation involves comparing different classification algorithms based on accuracy, which measures the percentage of correctly predicted instances. The three classifiers—KNN, Naïve Bayes, and Decision Tree—are assessed using accuracy scores, helping to determine which model performs best for customer churn prediction. Accuracy alone, however, does not fully capture model effectiveness; other metrics like precision, recall, and F1-score should also be considered.

```
# Print accuracy scores for each classifier
print(f"KNN Accuracy: {accuracy_knn:.4f}")
print(f"Naïve Bayes Accuracy: {accuracy_nb:.4f}")
print(f"Decision Tree Accuracy: {accuracy_dt:.4f}")
```

```
KNN Accuracy: 0.8396
Naïve Bayes Accuracy: 0.8341
Decision Tree Accuracy: 0.8486
```

Observation:

- KNN Accuracy: 0.8396
- Naïve Bayes Accuracy: 0.8341
- Decision Tree Accuracy: 0.8486

Conclusion:

In this experiment, we implemented three classification algorithms—KNN, Naïve Bayes, and Decision Tree—on a customer churn dataset. The dataset was preprocessed by handling missing values and standardizing features before splitting it into training and testing sets. Each model was trained, tested, and evaluated based on accuracy, classification reports, and confusion matrices. Among the models, the Decision Tree performed the best with 84.86% accuracy, followed by KNN (83.96%) and Naïve Bayes (83.41%). The Decision Tree's structured approach to splitting data likely contributed to its superior performance. Further tuning of hyperparameters and feature engineering could enhance model accuracy even further.