Name:Kaushik Kotian Roll No.:30 Div:D15B Batch:B

Experiment no.:6

AIM: Implementation of Classification modeling

PROBLEM STATEMENT:

1. Choose a classifier for the classification problem.
2. Evaluate the performance of the classifier.

- **K-Nearest Neighbors (KNN)**
  K-Nearest Neighbors (KNN) is a simple, instance-based learning algorithm where the function is only approximated locally and all computation is deferred until function evaluation. It is a non-parametric method used for classification and regression. For classification, KNN assigns a class to a sample based on the majority class among its k nearest neighbors. The number of neighbors, k, is a critical parameter in KNN and is typically chosen via cross-validation. KNN is sensitive to the local structure of the data and is very intuitive, but it can be computationally expensive, especially as the size of the dataset grows, because it requires calculating the distance of a sample to all other samples.

- **Naive Bayes**
  Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. They are highly scalable and can handle an arbitrary number of categorical or numerical features. Naive Bayes classifiers work well in many real-world situations, such as document classification and spam filtering, despite the simplifying assumption of feature independence. They are particularly useful

when the dimensionality of the input is high. Despite their simplicity, Naive Bayes classifiers have shown to be effective in various applications.

- **Support Vector Machines (SVMs)**
  Support Vector Machines (SVMs) are a set of supervised learning methods used for classification, regression, and outliers detection. The basic SVM takes a set of input data and predicts, for each given input, which of two possible classes the input is a part of, making it a non-probabilistic binary linear classifier. SVM models represent the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. SVMs can also efficiently perform a non-linear classification using the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

- **Decision Trees**
  Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A decision tree is represented as a binary tree, where each node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome. The paths from root to leaf represent classification rules. One of the advantages of decision trees is their interpretability – they are easy to understand and visualize. However, they can create overly complex trees that do not generalize well from the training data, known as overfitting, which can be mitigated by setting limits on tree size or pruning.

IMPLEMENTATION:

1. Loading the dataset and importing libraries

```
[28] import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import LabelEncoder
     from sklearn.neighbors import KNeighborsRegressor
     from sklearn.metrics import mean_squared_error
     from math import sqrt
     import matplotlib.pyplot as plt
```

```
[5] df = pd.read_csv('/content/loan_data_set.csv')
    df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

2. Determining the independent features as X and dependent features as y

```
[7] X = df[['ApplicantIncome', 'Credit_History', 'LoanAmount']]
    y = df['Loan_Status']
```

3. Giving unique values to each category

```
df.loc[:, ['ApplicantIncome', 'Credit_History', 'LoanAmount']] =
df[['ApplicantIncome', 'Credit_History',
'LoanAmount']].fillna(df[['ApplicantIncome', 'Credit_History',
'LoanAmount']].median())
```

4. Splitting the dataset into train and test datasets and then standardizing the values

```
[9] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

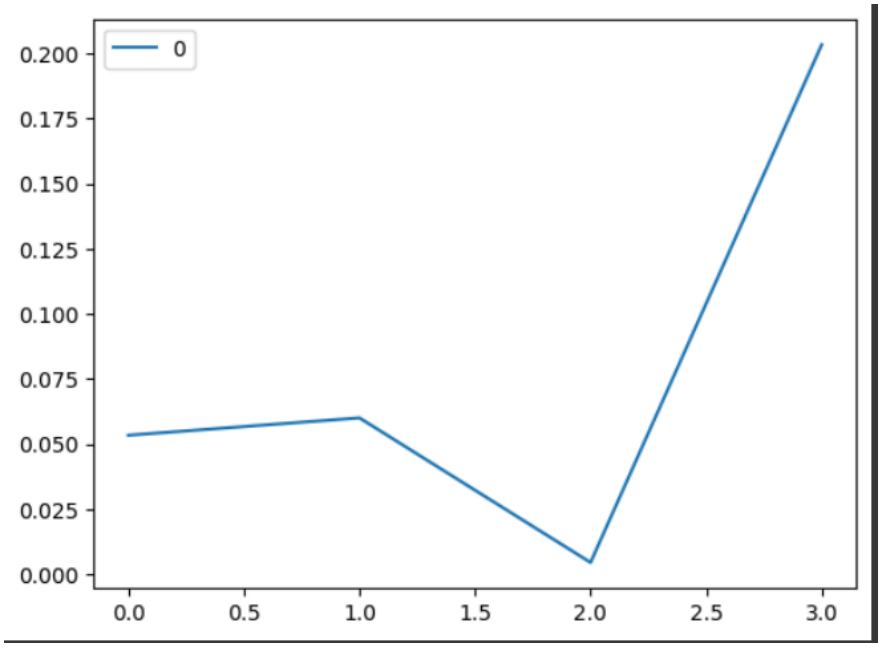
K-Nearest Neighbor(KNN) Algorithm

1. Finding the best value of n_neighbour

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse_val = [] #to store rmse values for different k
for K in range(4):
  K = K+1
  model = neighbors.KNeighborsRegressor(n_neighbors = K)

  model.fit(X_train, y_train) #fit the model
  pred=model.predict(X_test) #make prediction on test set
  error = sqrt(mean_squared_error(y_test,pred)) #calculate rmse
  rmse_val.append(error) #store rmse values
# print('RMSE value for k= ', K, 'is:', error)

#plotting the rmse values against k values
curve = pd.DataFrame(rmse_val) #elbow curve
curve.plot()
```



2. Model implementation

```
[66] from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score, classification_report

[67] knn = KNeighborsClassifier()
     knn.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

3. Model evaluation

```
[68] y_pred_knn = knn.predict(X_test)
```

```
print(f"KNN Accuracy: {accuracy_score(y_test, y_pred_knn)}")
```

```
KNN Accuracy: 0.7479674796747967
```

```
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.71      0.47      0.56        43
           1       0.76      0.90      0.82        80

    accuracy                           0.75       123
   macro avg       0.74      0.68      0.69       123
weighted avg       0.74      0.75      0.73       123
```

Support Vector Machine (SVM)

1. Model Implementation

```
from sklearn.svm import SVC
svm = SVC()
svm.fit(X_train, y_train)
```

```
▾ SVC
SVC()
```

2. Model Evaluation

```
[78] print(classification_report(y_test, y_pred_svm))
```

	precision	recall	f1-score	support
0	1.00	0.42	0.59	43
1	0.76	1.00	0.86	80
accuracy			0.80	123
macro avg	0.88	0.71	0.73	123
weighted avg	0.85	0.80	0.77	123

3. Prediction

```
y_pred_svm = svm.predict(X_test)
print(f"SVM Accuracy: {accuracy_score(y_test, y_pred_svm)}")
```

```
SVM Accuracy: 0.7967479674796748
```

Decision Tree Classifier

1.Model implementation

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

2.Model Evaluation

```
[81] print(classification_report(y_test, y_pred_dt))
```

	precision	recall	f1-score	support
0	0.57	0.49	0.53	43
1	0.74	0.80	0.77	80
accuracy			0.69	123
macro avg	0.66	0.64	0.65	123
weighted avg	0.68	0.69	0.69	123

3.Prediction

```
y_pred_dt = dt.predict(X_test)
print(f"Decision Tree Accuracy: {accuracy_score(y_test, y_pred_dt)}")
```

```
Decision Tree Accuracy: 0.6910569105691057
```

Naive Bayes

1. Gaussian Naive Bayes
    a. Model Implementation

```
[71] from sklearn.naive_bayes import GaussianNB
```

```
[72] gnb = GaussianNB()
```

```
gnb.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

b. Model Evaluation

```
[75] print(classification_report(y_test, y_pred_gnb))
```

	precision	recall	f1-score	support
0	0.90	0.42	0.57	43
1	0.76	0.97	0.85	80
accuracy			0.78	123
macro avg	0.83	0.70	0.71	123
weighted avg	0.81	0.78	0.75	123

c. Prediction

```
[74] y_pred_gnb = gnb.predict(X_test)
     print(f"Naive Bayes Accuracy: {accuracy_score(y_test, y_pred_gnb)}")

     Naive Bayes Accuracy: 0.7804878048780488
```

CONCLUSION:

In this experiment, we have predicted 'Home Ownership' with the help of different features like 'Current Loan Amount, Annual Income and Monthly Debt'. This is done with the help of Supervised Machine Learning through Classification Algorithms like KNN, Naive Bayes, SVM and Decision Tree. All the algorithms have different algorithm scores.