EXPERIMENT - 6

AIM: Classification modeling

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

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

ABOUT DATASET:

We took the train dataset which contains the loan stats of people.

It also has many other factors such as home ownership, grade,funded amount etc

THEORY:

Classification is when the feature to be predicted contains categories of values. Each of these categories is considered as a class into which the predicted value falls and hence has its name, classification.
As stated earlier, classification is when the feature to be predicted contains categories of values. Each of these categories is considered as a class into which the predicted value falls.
Classification algorithms include:

- Naive Bayes
- Logistic regression
- K-nearest neighbors
- (Kernel) SVM
- Decision tree
- Ensemble learning

K-nearest neighbors

The general idea behind K-nearest neighbors (KNN) is that data points are considered to belong to the class with which it shares the most number of common points in terms of its distance. K number of nearest points around the data point to be predicted are taken into consideration.

These K points at this time already belong to a class. The data point under consideration is said to belong to the class with which the most number of points from these K points belong. There are several methods to calculate the distance between points. The most popular formula to calculate this is the Euclidean distance.

Support Vector Machines

Support Vector Machines (SVM) output an optimal line of separation between the classes, based on the training data entered as input. This line of separation is called a hyperplane in a multi-dimensional environment. SVM takes into consideration outliers that lie pretty close to another class to derive this separating hyperplane. After the model is constructed with this hyperplane, any new point to be predicted checks to see which side of the hyperplane this values lies in. Even in 2-dimensional space, constructing this line of separation between classes can sometimes be tricky if the points are distributed without a clear distinction. Also, doing this when multiple features contribute to describe a data point is a complicated process. For these multi-dimensional spaces, where data is not linearly separable, we map it to a higher dimensional space to create this separation. This mapping to a higher dimension is achieved by applying a *kernel function*. There are several types of kernel functions, and the most common ones are the polynomial and the Guassian radial basis function (RBF). After this plane of separation is derived, the data is mapped back to its original dimension. Prediction at this point is merely finding if this point lies within or outside the plane.

Decision tree

Decision tree-based models use training data to derive rules that are used to predict an output. For example, assume that the problem statement was to identify if a person can play tennis today. Depending on the values from the training data, the model forms a decision tree. The model derived could have constructed a decision tree with the following rules.

1. First check the outlook column. If it's overcast, you definitely never go.
2. But if it's sunny and humid, then you don't go.
3. If it's sunny and normal, you go.
4. If it's rainy and windy, you don't go.
5. And if it's rainy and not windy, you go.

Naive Bayes

Naive Bayes applies the Bayes' theorem to calculate the probability of a data point belonging to a particular class. Given the probability of certain related values, the formula to calculate the probability of an event *B*, given event *A* to occur is calculated as follows.
$P(B|A) = (P(A|B) * P(B) / P(A))$. This theory is considered naive, because it assumes that there is no dependency between any of the input features. Even with this not true or naive assumption, the Naive Bayes algorithm has been proven to perform really well in certain use cases like spam filters.

IMPLEMENTATION:

- **Naive Bayes**

Naive Bayes

```
[197] print(data['home_ownership'].unique())
```

```
['OWN' 'MORTGAGE' 'RENT' 'OTHER' 'NONE' 'ANY']
```

```
[198] x = data[['loan_amnt','tot_cur_bal','loan_status']]
      y = data['home_ownership']
```

```
[199] data['home_ownership'].replace(['OWN', 'MORTGAGE' ,'RENT', 'OTHER' ,'NONE', 'ANY'],[1,2,3,4,5,6],inplace=True)
```

```
[200] #Gaussian Naive Bayes
```

```
[204] from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(X,y, random_state = 0)
```

```
[205] from sklearn.naive_bayes import GaussianNB
      clf1=GaussianNB()
      clf1.fit(X_train,y_train)
```

```
GaussianNB
GaussianNB()
```

```
from sklearn.metrics import classification_report, accuracy_score
y_pred1=clf1.predict(X_test)
score=accuracy_score(y_pred1,y_test)
print(score)
```

```
0.6867557679160374
```

```
[207] df1=pd.DataFrame(y_test,y_pred1)
      df1=pd.DataFrame({"Actual":y_test,'Predicted':y_pred1})
      print(df1)
```

```
        Actual  Predicted
410253       3          3
3662         3          3
207553       2          2
12488        2          2
324791       2          2
...        ...        ...
60529        2          2
459620       2          3
91310        3          3
307260       3          2
19600        3          3

[133107 rows x 2 columns]
```

```
[209] #Bernoulli Naive Bayes
```

```
[210] from sklearn.naive_bayes import BernoulliNB
      clf2=BernoulliNB()
      clf2.fit(X_train,y_train)
```

```
▾ BernoulliNB
BernoulliNB()
```

```
[211] from sklearn.metrics import classification_report, accuracy_score
      y_pred2=clf2.predict(X_test)
      score=accuracy_score(y_pred2,y_test)
      print(score)
```

```
0.5027383984313372
```

```
df2=pd.DataFrame(y_test,y_pred2)
df2=pd.DataFrame({"Actual":y_test,'Predicted':y_pred2})
print(df2)
```

```
        Actual  Predicted
410253       3          2
3662         3          2
207553       2          2
12488        2          2
324791       2          2
...        ...        ...
60529        2          2
459620       2          2
91310        3          2
307260       3          2
19600        3          2

[133107 rows x 2 columns]
```

```
[213] #Multinomial Naive Bayes
```

```
[214] from sklearn.naive_bayes import MultinomialNB
      clf3=MultinomialNB()
      clf3.fit(X_train,y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
[215] from sklearn.metrics import classification_report, accuracy_score
      y_pred3=clf3.predict(X_test)
      score=accuracy_score(y_pred3,y_test)
      print(score)
```

```
0.4186406424906278
```

```
df3=pd.DataFrame(y_test,y_pred3)
df3=pd.DataFrame({"Actual":y_test,'Predicted':y_pred3})
print(df3)
```

```
        Actual  Predicted
410253       3          4
3662         3          3
207553       2          6
12488        2          2
324791       2          1
...        ...        ...
60529        2          6
459620       2          1
91310        3          3
307260       3          6
19600        3          1

[133107 rows x 2 columns]
```

- **K-nearest neighbors**

```
data['grade'].unique()
```

```
array(['E', 'B', 'A', 'D', 'C', 'F', 'G'], dtype=object)
```

```
[7] data['grade'].replace(['E','B','A','D','C','F','G'],[1,2,3,4,5,6,7],inplace=True)
    data.head()
```

	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	batch_enrolled	int_rate	grade	sub_grade	emp_title	...	collections_12_mths_
0	58189336	14350	14350	14350.0	36 months		19.19	1	E3	clerk	...	
1	70011223	4800	4800	4800.0	36 months	BAT1586599	10.99	2	B4	Human Resources Specialist	...	
2	70255675	10000	10000	10000.0	36 months	BAT1586599	7.26	3	A4	Driver	...	
3	1893936	15000	15000	15000.0	36 months	BAT4808022	19.72	4	D5	Us office of Personnel Management	...	
4	7652106	16000	16000	16000.0	36 months	BAT2833642	10.64	2	B2	LAUSD-HOLLYWOOD HIGH SCHOOL	...	

5 rows × 45 columns

```
[8]  X = data[['loan_amnt','funded_amnt','funded_amnt_inv']]
     y = data['grade']
```

```
[9]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X,y, random_state = 0)
```

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

```
#import required packages
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
from sklearn.metrics import mean_squared_error
from math import sqrt
```

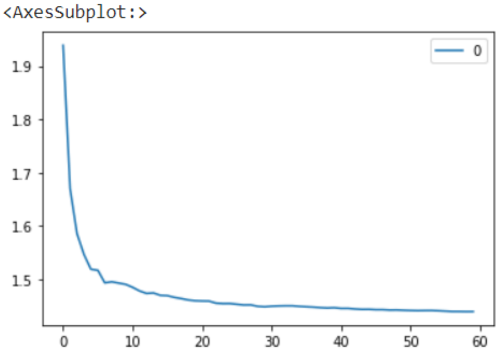
```
[12] rmse_val = [] #to store rmse values for different k
     for K in range(60):
         K = K+1
         model = neighbors.KNeighborsRegressor(n_neighbors = K)

         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
```

```
[13]
         # print('RMSE value for k= ' , K , 'is:', error)

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

<AxesSubplot:>



```
[14] #define the model
     classifier=KNeighborsClassifier(n_neighbors=60)
     #fit model
     classifier.fit(X_train, y_train)
     y_pred=classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[  598  3424   815  1882  3889     0     0]
 [  341 21710  3737  1793 10388     0     0]
 [  141 12364  3433   778  5479     0     0]
 [  728  8524  1585  3014  7226     0     0]
 [  561 18308  3284  2783 12051     0     0]
 [  283   914   262   735  1319     0     0]
 [   87   167    40   185   279     0     0]]
```

```
[53] from sklearn.metrics import accuracy_score
     print(accuracy_score(y_test,y_pred))
```

```
0.7262953864184453
```

```
data=pd.DataFrame({"Actual Status":y_test,'Predicted Status':y_pred})
print(data)
```

```
        Actual Status  Predicted Status
410253              3                 3
3662                3                 3
207553              2                 2
12488               2                 2
324791              2                 2
...               ...               ...
60529               2                 2
459620              2                 2
91310               3                 3
307260              3                 2
19600               3                 3

[133107 rows x 2 columns]
```

- **(Kernel) SVM**

```
[ ] from sklearn.svm import SVC # "Support vector classifier"
    svm = SVC(kernel='linear', random_state=0)
    svm.fit(X_train, y_train)
    y_svm_pred= svm.predict(X_test)
```

```
[ ] #Creating the Confusion matrix
    from sklearn.metrics import confusion_matrix
    cm_svm = confusion_matrix(y_test, y_svm_pred)
    print(cm_svm)
```

```
[[   0  835    0    0]
 [   0 4726    0    0]
 [   0 3529    0    0]
 [   0   16    0    0]]
```

```
[ ] from sklearn.metrics import accuracy_score
    print(accuracy_score(y_test,y_svm_pred))
```

```
0.5189984625521634
```


```
[ ] from sklearn.metrics import accuracy_score
    print(accuracy_score(y_test,y_svm_pred))
```

```
0.5189984625521634
```


```
svmdata=pd.DataFrame({"Actual Status":y_test,'Predicted Status':y_svm_pred})
print(svmdata)
```

```
       Actual Status  Predicted Status
8312               2                 2
60723              2                 2
29653              2                 2
2101               3                 2
65575              2                 2
...              ...               ...
33109              3                 2
66259              2                 2
55328              2                 2
75245              3                 2
95772              2                 2

[9106 rows x 2 columns]
```

- **<u>Decision tree</u>**

Decision Tree Classifier

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
```

```
[183] print(data['application_type'].unique())
```

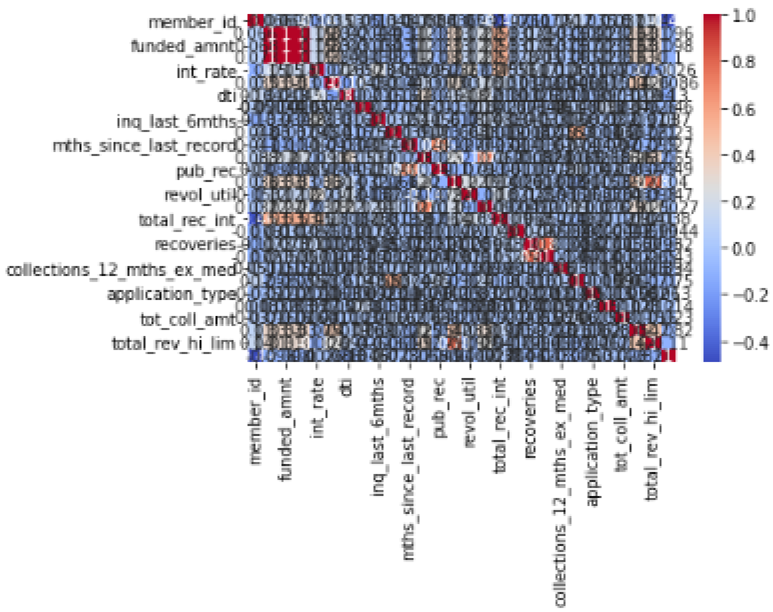
```
['INDIVIDUAL' 'JOINT']
```

```
[184] data=data.replace('INDIVIDUAL',1)
      data=data.replace('JOINT',0)
      data.head()
```

	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	batch_enrolled	int_rate	grade	sub_grade	emp_title	...	collections_12_mths_ex_med	mths_since_last
0	58189336	14350	14350	14350.0	36 months		19.19	E	E3	clerk	...	0.0	
1	70011223	4800	4800	4800.0	36 months	BAT1586599	10.99	B	B4	Human Resources Specialist	...	0.0	
2	70255675	10000	10000	10000.0	36 months	BAT1586599	7.26	A	A4	Driver	...	0.0	
3	1893936	15000	15000	15000.0	36 months	BAT4808022	19.72	D	D5	Us office of Personnel Management	...	0.0	
4	7652106	16000	16000	16000.0	36 months	BAT2833642	10.64	B	B2	LAUSD-HOLLYWOOD HIGH SCHOOL	...	0.0	

5 rows × 45 columns

```python
import seaborn as sns
import matplotlib.pyplot as plt
corr_matrix = data.corr(method='pearson')
# Plot the heatmap
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True)
plt.show()
```




```python
[186] X=data[['loan_amnt','funded_amnt']]
      y=data['application_type']
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
[187] clf = DecisionTreeClassifier(max_depth=3,criterion="entropy",random_state=100)
      clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=100)
```

```python
[188] y_pred=clf.predict(X_test)
```

```python
df=pd.DataFrame(y_test,y_pred)
```

+ Code + Text

```python
[190] df=pd.DataFrame({"Actual application type":y_test,'Predicted application type':y_pred})
```

```
df
```

	Actual application type	Predicted application type
447404	1	1
105437	1	1
418519	1	1
264650	1	1
32781	1	1
...	...	...
467499	1	1
86149	1	1
19718	1	1
483234	1	1
74966	1	1

106486 rows × 2 columns

```
[192] accuracy_score(y_test,y_pred)*100
```

```
99.93989820258062
```

```
[193] X=data[['application_type']]
      y=data['loan_status']
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[194] clf = DecisionTreeClassifier(max_depth=3,criterion="entropy",random_state=100)
      clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=100)
```

```
[195] y_pred=clf.predict(X_test)
      accuracy_score(y_test,y_pred)*100
```

```
76.46920721972842
```

```
df=pd.DataFrame(y_test,y_pred)
df=pd.DataFrame({"Actual loan status":y_test,'Predicted loan status':y_pred})
df
```

	Actual loan status	Predicted loan status
447404	0	0
105437	0	0
418519	1	0
264650	1	0
32781	1	0
...	...	...
467499	0	0
86149	0	0
19718	0	0
483234	1	0
74966	0	0

106486 rows × 2 columns

CONCLUSION: Thus we performed prediction of different attributes in our dataset and we also calculated its accuracy. We successfully implemented K-Nearest Neighbors (KNN) ,Naive Bayes, Support Vector Machines (SVMs), and Decision Tree Classification.