NAME: SACHIN VISHWAKARMA

UID: 2019140066

BATCH: D

SUBJECT: AIML

EXPERIMENT: 5

AIM:

Naive Bayes Classifier algorithm implementation without using any inbuilt libraries.

use the dataset studied in Class

Class:

C1:buys_computer = ‘yes’

C2:buys_computer = ‘no’

Data to be classified:

X = (age <=30,

Income = medium,

Student = yes

Credit_rating = Fair)

steps:
1- calculate mean and variance for each column

2-calculate probability from Gaussian density function

3-calculate prior and posterior probabilities:

4-Make predictions

CODE:

```
# -*- coding: utf-8 -*-
"""aiml_exp5.ipynb"""

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# %matplotlib inline

import os

cwd = os.getcwd()  # Get the current working directory (cwd)
files = os.listdir(cwd)  # Get all the files in that directory
print("Files in %r: %s" % (cwd, files))

data = 'kaggle/input/adult-dataset/adult.csv'

df = pd.read_csv(data, header=None, sep=',\s')

# view dimensions of dataset
df.shape

# preview the dataset
df.head()

col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',
'marital_status', 'occupation', 'relationship',
             'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week',
'native_country', 'income']

df.columns = col_names

df.columns

#  again preview the dataset
df.head()

# view summary of dataset
df.info()

# find categorical variables
categorical = [var for var in df.columns if df[var].dtype=='O']

print('There are {} categorical variables\n'.format(len(categorical)))

print('The categorical variables are :\n\n', categorical)
```

```
# view the categorical variables
df[categorical].head()

# check missing values in categorical variables
df[categorical].isnull().sum()

# view frequency counts of values in categorical variables
for var in categorical:

    print(df[var].value_counts())

# view frequency distribution of categorical variables
for var in categorical:

    print(df[var].value_counts()/np.float(len(df)))

# check labels in workclass variable
df.workclass.unique()

# check frequency distribution of values in workclass variable
df.workclass.value_counts()

# replace '?' values in workclass variable with `NaN`
df['workclass'].replace('?', np.NaN, inplace=True)

# check labels in occupation variable
df.occupation.unique()

# # check frequency distribution of values in occupation variable
df.occupation.value_counts()

# replace '?' values in occupation variable with `NaN`
df['occupation'].replace('?', np.NaN, inplace=True)

# again check the frequency distribution of values in occupation variable
df.occupation.value_counts()

# check labels in native_country variable
df.native_country.unique()

# check frequency distribution of values in native_country variable
df.native_country.value_counts()

# replace '?' values in native_country variable with `NaN`
df['native_country'].replace('?', np.NaN, inplace=True)

# again check the frequency distribution of values in native_country variable
```

```python
df.native_country.value_counts()

# Check missing values in categorical variables again
df[categorical].isnull().sum()

# check for cardinality in categorical variables
for var in categorical:

    print(var, ' contains ', len(df[var].unique()), ' labels')

# find numerical variables
numerical = [var for var in df.columns if df[var].dtype!='O']

print('There are {} numerical variables\n'.format(len(numerical)))

print('The numerical variables are :', numerical)

# view the numerical variables
df[numerical].head()

# check missing values in numerical variables
df[numerical].isnull().sum()

# Declare feature vector and target variable
X = df.drop(['income'], axis=1)

y = df['income']

# split X and y into training and testing sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
random_state = 0)

# check the shape of X_train and X_test
X_train.shape, X_test.shape

# check data types in X_train
X_train.dtypes

# display categorical variables
categorical = [col for col in X_train.columns if X_train[col].dtypes == 'O']

categorical

# display numerical variables
numerical = [col for col in X_train.columns if X_train[col].dtypes != 'O']
```

```
numerical

# print percentage of missing values in the categorical variables in training
set
X_train[categorical].isnull().mean()

# print categorical variables with missing data
for col in categorical:
    if X_train[col].isnull().mean()>0:
        print(col, (X_train[col].isnull().mean()))

# impute missing categorical variables with most frequent value
for df2 in [X_train, X_test]:
    df2['workclass'].fillna(X_train['workclass'].mode()[0], inplace=True)
    df2['occupation'].fillna(X_train['occupation'].mode()[0], inplace=True)
    df2['native_country'].fillna(X_train['native_country'].mode()[0],
inplace=True)

# check missing values in categorical variables in X_train
X_train[categorical].isnull().sum()

# check missing values in categorical variables in X_test
X_test[categorical].isnull().sum()

# check missing values in X_train
X_train.isnull().sum()

# check missing values in X_test
X_test.isnull().sum()

# print categorical variables
categorical

X_train[categorical].head()

pip install --upgrade category_encoders

# import category encoders

import category_encoders as ce

# encode remaining variables with one-hot encoding

encoder = ce.OneHotEncoder(cols=['workclass', 'education', 'marital_status',
'occupation', 'relationship',
                                 'race', 'sex', 'native_country'])

X_train = encoder.fit_transform(X_train)
```

```python
X_test = encoder.transform(X_test)

X_train.head()

X_train.shape

X_test.head()

X_test.shape

cols = X_train.columns

from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

X_train = pd.DataFrame(X_train, columns=[cols])

X_test = pd.DataFrame(X_test, columns=[cols])

X_train.head()
# now we have X_train dataset ready to be fed into the Gaussian Naive Bayes
classifier.

# train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB


# instantiate the model
gnb = GaussianNB()


# fit the model
gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)

y_pred

from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,
y_pred)))
```

```
# Compare the train-set and test-set accuracy
# Now, I will compare the train-set and test-set accuracy to check for
overfitting.
y_pred_train = gnb.predict(X_train)

y_pred_train

# training set accuracy
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train,
y_pred_train)))

# Check for overfitting and underfitting
# print the scores on training and test set

print('Training set score: {:.4f}'.format(gnb.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(gnb.score(X_test, y_test)))

# check class distribution in test set

y_test.value_counts()

# check null accuracy score

null_accuracy = (7407/(7407+2362))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])

# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual
Negative:0'],
```

```
                                 index=['Predict Positive:1', 'Predict 
Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))

TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]

# print classification accuracy

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))

# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))

# print precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))

# Recall is a percentage of correctly predicted positive outcomes out of all 
the actual positive outcomes.
# It can be given as the ratio of true positives (TP) to the sum of true 
positives and false negatives (TP + FN).
# Recall is also called Sensitivity.
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))

# True Positive Rate is synonymous with Recall.
true_positive_rate = TP / float(TP + FN)


print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))

# False Positive Rate
```

```
false_positive_rate = FP / float(FP + TN)


print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))

# Specificity
specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))

# f1-score is the weighted harmonic mean of precision and recall.
# The best possible f1-score would be 1.0 and the worst would be 0.0.
# f1-score is the harmonic mean of precision and recall.
# print the first 10 predicted probabilities of two classes- 0 and 1

y_pred_prob = gnb.predict_proba(X_test)[0:10]

y_pred_prob

# store the probabilities in dataframe

y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Prob of - <=50K',
'Prob of - >50K'])

y_pred_prob_df

# print the first 10 predicted probabilities for class 1 - Probability of >50K

gnb.predict_proba(X_test)[0:10, 1]

# store the predicted probabilities for class 1 - Probability of >50K

y_pred1 = gnb.predict_proba(X_test)[:, 1]

# plot histogram of predicted probabilities


# adjust the font size
plt.rcParams['font.size'] = 12


# plot histogram with 10 bins
plt.hist(y_pred1, bins = 10)


# set the title of predicted probabilities
plt.title('Histogram of predicted probabilities of salaries >50K')
```

```
# set the x-axis limit
plt.xlim(0,1)


# set the title
plt.xlabel('Predicted probabilities of salaries >50K')
plt.ylabel('Frequency')

# The ROC Curve plots the True Positive Rate (TPR) against the False Positive
Rate (FPR) at various threshold levels.
# True Positive Rate (TPR) is also called Recall. It is defined as the ratio
of TP to (TP + FN).
# False Positive Rate (FPR) is defined as the ratio of FP to (FP + TN).
# plot ROC Curve

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred1, pos_label = '>50K')

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Gaussian Naive Bayes Classifier for Predicting
Salaries')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()

# compute ROC AUC

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred1)

print('ROC AUC : {:.4f}'.format(ROC_AUC))

# calculate cross-validated ROC AUC

from sklearn.model_selection import cross_val_score
```

```python
Cross_validated_ROC_AUC = cross_val_score(gnb, X_train, y_train, cv=5,
scoring='roc_auc').mean()

print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))

# Applying 10-Fold Cross Validation

from sklearn.model_selection import cross_val_score

scores = cross_val_score(gnb, X_train, y_train, cv = 10, scoring='accuracy')

print('Cross-validation scores:{}'.format(scores))

# compute Average cross-validation score

print('Average cross-validation score: {:.4f}'.format(scores.mean()))
```

OUTPUT:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	income
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States	<=50K
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	<=50K
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	<=50K
3	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States	<=50K
4	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba	<=50K

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	workclass_8	fnlwgt	education_1	education_2	education_3	education_4	education_5	educati
32098	45	1	0	0	0	0	0	0	0	170871	1	0	0	0	0	
25206	47	0	1	0	0	0	0	0	0	108890	1	0	0	0	0	
23491	48	1	0	0	0	0	0	0	0	187505	0	1	0	0	0	
12367	29	1	0	0	0	0	0	0	0	145592	1	0	0	0	0	
7054	23	1	0	0	0	0	0	0	0	203003	0	0	1	0	0	

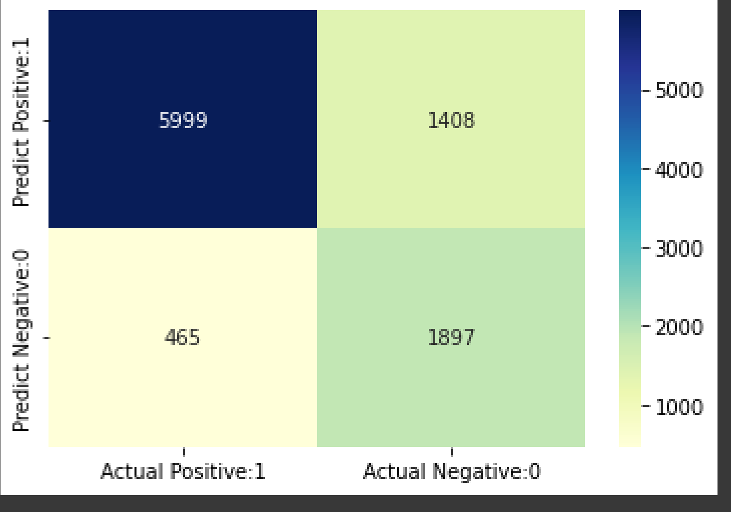
5 rows × 105 columns

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	workclass_8	fnlwgt	education_1	education_2	education_3	education_4	education_5	educati
22278	27	1	0	0	0	0	0	0	0	177119	0	1	0	0	0	
8950	27	1	0	0	0	0	0	0	0	216481	0	0	0	1	0	
7838	25	1	0	0	0	0	0	0	0	256263	0	0	0	0	0	
16505	46	1	0	0	0	0	0	0	0	147640	0	0	0	0	0	
19140	45	1	0	0	0	0	0	0	0	172822	0	0	0	0	0	

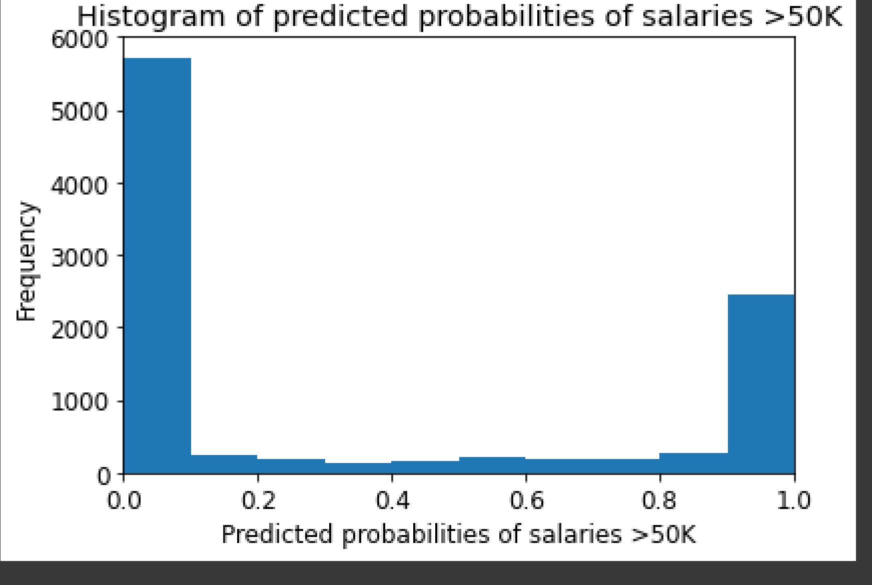
5 rows × 105 columns

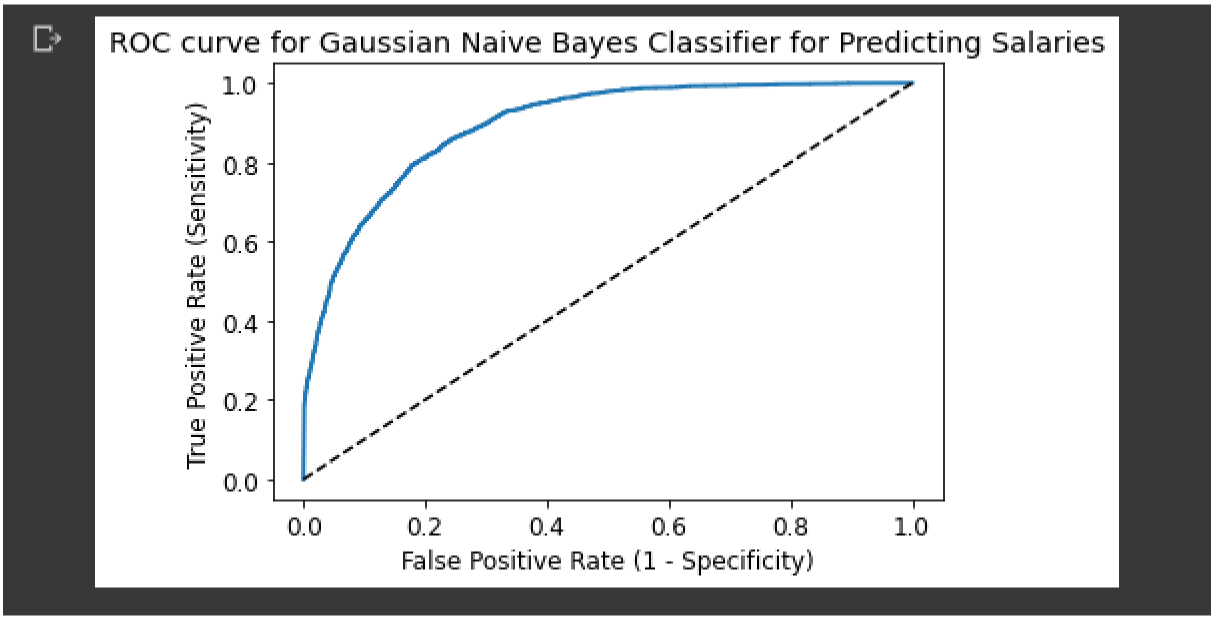
```
Confusion matrix

 [[5999 1408]
 [ 465 1897]]

True Positives(TP) =  5999

True Negatives(TN) =  1897

False Positives(FP) =  1408

False Negatives(FN) =  465
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe08d2869d0>
```




```
Text(0, 0.5, 'Frequency')
```




ROC curve for Gaussian Naive Bayes Classifier for Predicting Salaries
True Positive Rate (Sensitivity)
False Positive Rate (1 - Specificity)
0.0
0.2
0.4
0.6
0.8
1.0

CONCLUSION AND OBSERVATION:

- Using the mean cross-validation, we can conclude that we expect the model to be around 80.63% accurate on average.

- If we look at all the 10 scores produced by the 10-fold cross-validation, we can also conclude that there is a relatively small variance in the accuracy between folds, ranging from 81.35% accuracy to 79.64% accuracy. So, we can conclude that the model is independent of the particular folds used for training.

- Our original model accuracy is 0.8083, but the mean cross-validation accuracy is 0.8063. So, the 10-fold cross-validation accuracy does not result in performance improvement for this model.

- In this experiment, I build a Gaussian Naïve Bayes Classifier model to predict whether a person makes over 50K a year. The model yields a very good performance as indicated by the model accuracy which was found to be 0.8083.
- The training-set accuracy score is 0.8067 while the test-set accuracy to be 0.8083. These two values are quite comparable. So, there is no sign of overfitting.
- I have compared the model accuracy score which is 0.8083 with null accuracy score which is 0.7582. So, we can conclude that our Gaussian Naïve Bayes classifier model is doing a very good job in predicting the class labels.
- ROC AUC of our model approaches towards 1. So, we can conclude that our classifier does a very good job in predicting whether a person makes over 50K a year.
- Using the mean cross-validation, we can conclude that we expect the model to be around 80.63% accurate on average.
- If we look at all the 10 scores produced by the 10-fold cross-validation, we can also conclude that there is a relatively small variance in the accuracy between folds, ranging from 81.35% accuracy to 79.64% accuracy. So, we can conclude that the model is independent of the particular folds used for training.
- Original model accuracy is 0.8083, but the mean cross-validation accuracy is 0.8063. So, the 10-fold cross-validation accuracy does not result in performance improvement for this model.