Experiment No. 3	

Experiment No. 3
Implementation of Bayes classifier for recommendation
Date of Performance:
Date of Submission:
Marks:
Sign:

Aim: Implementation of Bayes classifier for recommendation

Objective: Able to perform naïve bayes classifier for implementation recommendations system.

Theory: Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset.

One of the most simple and effective classification algorithms, the Naïve Bayes classifier aids in the rapid development of machine learning models with rapid prediction capabilities.

Naïve Bayes algorithm is used for classification problems. It is highly used in text classification. In text classification tasks, data contains high dimension (as each word represent one feature in the data). It is used in spam filtering, sentiment detection, rating classification etc. The advantage of using naïve Bayes is its speed. It is fast and making prediction is easy with high dimension of data.

Implementation:

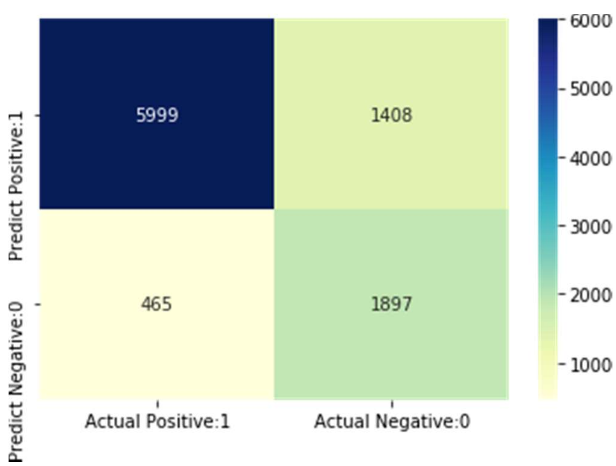
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import warnings
warnings.filterwarnings('ignore')
data = '/kaggle/input/adult-dataset/adult.csv'
df = pd.read_csv(data, header=None, sep=',\s')
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
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)
df[categorical].isnull().sum()
for var in categorical:
    print(df[var].value_counts())
for var in categorical:
    print(df[var].value_counts()/np.float(len(df)))
df.workclass.unique()
df.workclass.value_counts()
df['workclass'].replace('?', np.NaN, inplace=True)
df.workclass.value_counts()
df.occupation.unique()
df.occupation.value_counts()
df['occupation'].replace('?', np.NaN, inplace=True)
df.occupation.value_counts()
df.native_country.unique()
df.native_country.value_counts()
df['native_country'].replace('?', np.NaN, inplace=True)
df.native_country.value_counts()
df[categorical].isnull().sum()
for var in categorical:
```

```
    print(var, ' contains ', len(df[var].unique()), ' labels')
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)
df[numerical].head()
df[numerical].isnull().sum()
X = df.drop(['income'], axis=1)
y = df['income']
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)
X_train.shape, X_test.shape
X_train.dtypes
categorical = [col for col in X_train.columns if X_train[col].dtypes == 'O']
numerical = [col for col in X_train.columns if X_train[col].dtypes != 'O']
X_train[categorical].isnull().mean()
for col in categorical:
    if X_train[col].isnull().mean()>0:
        print(col, (X_train[col].isnull().mean()))
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)
X_train[categorical].isnull().sum()
X_test[categorical].isnull().sum()
X_train.isnull().sum()
X_test.isnull().sum()
import category_encoders as ce
encoder = ce.OneHotEncoder(cols=['workclass', 'education', 'marital_status', 'occupation', 'relationship',
                                 'race', 'sex', 'native_country'])
X_train = encoder.fit_transform(X_train)
X_test = encoder.transform(X_test)
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])
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
from sklearn.metrics import accuracy_score
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
y_pred_train = gnb.predict(X_train)
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
print('Training set score: {:.4f}'.format(gnb.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(gnb.score(X_test, y_test)))
y_test.value_counts()
null_accuracy = (7407/(7407+2362))
print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
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])
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]
classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('Classification error : {0:0.4f}'.format(classification_error))
precision = TP / float(TP + FP)
print('Precision : {0:0.4f}'.format(precision))
recall = TP / float(TP + FN)
print('Recall or Sensitivity : {0:0.4f}'.format(recall))
true_positive_rate = TP / float(TP + FN)
print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
false_positive_rate = FP / float(FP + TN)
print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))
specificity = TN / (TN + FP)
print('Specificity : {0:0.4f}'.format(specificity))
y_pred_prob = gnb.predict_proba(X_test)[0:10]
```

```
y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Prob of - <=50K', 'Prob of - >50K'])
gnb.predict_proba(X_test)[0:10, 1]
y_pred1 = gnb.predict_proba(X_test)[:, 1]
plt.rcParams['font.size'] = 12
plt.hist(y_pred1, bins = 10)
plt.title('Histogram of predicted probabilities of salaries >50K')
plt.xlim(0,1)
plt.xlabel('Predicted probabilities of salaries >50K')
plt.ylabel('Frequency')
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()
from sklearn.metrics import roc_auc_score
ROC_AUC = roc_auc_score(y_test, y_pred1)
print('ROC AUC : {:.4f}'.format(ROC_AUC))
from sklearn.model_selection import cross_val_score
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))
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))
print('Average cross-validation score: {:.4f}'.format(scores.mean()))
```

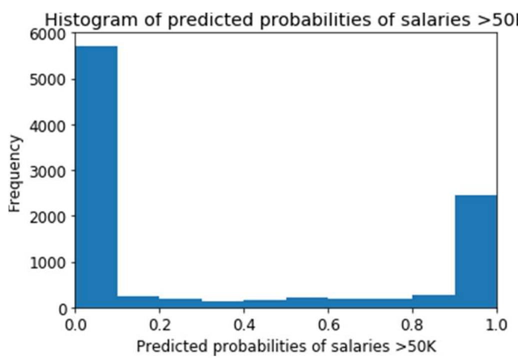
Output:

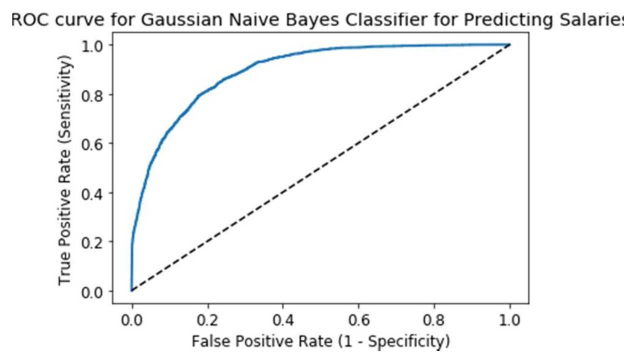
```
Model accuracy score: 0.8083
Training-set accuracy score: 0.8067
Training set score: 0.8067
Test set  score:  0.8083
Null accuracy score: 0.7582
Confusion matrix
 [[5999 1408]
 [ 465 1897]]
True Positives(TP) = 5999
True Negatives(TN) = 1897
False Positives(FP) = 1408
False Negatives(FN) =  465
```



	precision	recall	f1-score	support
<=50K	0.93	0.81	0.86	7407
>50K	0.57	0.80	0.67	2362
accuracy			0.81	9769
macro avg	0.75	0.81	0.77	9769
weighted avg	0.84	0.81	0.82	9769

```
Classification accuracy : 0.8083
Classification error : 0.1917
Precision : 0.8099
Recall or Sensitivity : 0.9281
True Positive Rate : 0.9281
False Positive Rate : 0.4260
Specificity : 0.5740
```






```
ROC AUC : 0.8941
Cross validated ROC AUC : 0.8938
Cross-validation scores:[0.81359649 0.80438596 0.81184211 0.80517771 0.
79640193 0.79684072
 0.81044318 0.81175954 0.80210619 0.81035996]
Average cross-validation score: 0.8063
```

Conclusion:

In conclusion, implementing the Bayes classifier for recommendation systems offers a probabilistic framework for personalized suggestions. Leveraging Bayesian inference, it efficiently predicts user preferences based on past behavior and item attributes. Its simplicity and effectiveness make it a valuable tool for enhancing recommendation accuracy and user satisfaction.