Experiment :

AIM : Write a program to implement Naive bayes Algorithm

THEORY :

The Naive Bayes algorithm is an intuitive method that uses the probabilities of each attribute belonging to each class to make a prediction. It is the supervised learning approach you would come up with if you wanted to model a predictive modeling problem probabilistically.

Naive bayes simplifies the calculation of probabilities by assuming that the probability of each attribute belonging to a given class value is independent of all other attributes. This is a strong assumption but results in a fast and effective method.

The probability of a class value given a value of an attribute is called the conditional probability. By multiplying the conditional probabilities together for each attribute for a given class value, we have a probability of a data instance belonging to that class.

To make a prediction we can calculate probabilities of the instance belonging to each class and select the class value with the highest probability.

Naive bases is often described using categorical data because it is easy to describe and calculate using ratios. A more useful version of the algorithm for our purposes supports numeric attributes and assumes the values of each numerical attribute are normally distributed (fall somewhere on a bell curve). Again, this is a strong assumption, but still gives robust results.

BAYES THEOREM

- Given a hypothesis h and data D which bears on the hypothesis:

$$P(h \mid D) = \frac{P(D \mid h)\,P(h)}{P(D)}$$

- P(h): independent probability of h: **prior probability**
- P(D): independent probability of D
- P(D|h): conditional probability of D given h: **likelihood**
- P(h|D): conditional probability of h given D: **posterior probability**

- Based on Bayes Theorem, we can compute the **Maximum A Posterior** (MAP) hypothesis for the data
- We are interested in the best hypothesis for some space H given observed training data D.

$$h_{MAP} \equiv \underset{h \in H}{\operatorname{argmax}} P(h \mid D)$$
$$= \underset{h \in H}{\operatorname{argmax}} \frac{P(D \mid h) P(h)}{P(D)}$$
$$= \underset{h \in H}{\operatorname{argmax}} P(D \mid h) P(h)$$

H: set of all hypothesis.

Note that we can drop P(D) as the probability of the data is constant (and independent of the hypothesis).

- Now assume that all hypothesis are equally probable a prior, i.e. $\mathbf{P(h_i) = P(h_j)}$ for all h_i, h_j belong to H.
- This is called assuming a **uniform prior**. It simplifies computing the posterior:

$$h_{ML} = \underset{h \in H}{\arg\max} P(D \mid h)$$

- This hypothesis is called the **maximum likelihood hypothesis**.

- The classification problem may be formalized using a-posterior probabilities:
- **P(C|X)** = prob. that the sample tuple $\mathbf{X=<x_1,\ldots,x_k>}$ is of class **C**.
- E.g. P(class=N | outlook= sunny, windy=true,…)
- Idea: assign to sample **X** the class label **C** such that **P(C|X) is maximal**
- Bayes theorem:

$$\mathbf{P(C|X) = P(X|C) \cdot P(C) / P(X)}$$

- P(X) is constant for all classes
- P(C) = relative freq of class C samples
- C such that P(C|X) is maximum = C such that P(X|C)·P(C) is maximum
- Problem: computing P(X|C) is unfeasible!

- **Bayes classification**

$$P(C \mid \mathbf{X}) \propto P(\mathbf{X} \mid C)P(C) = P(X_1, \cdots, X_n \mid C)P(C)$$

Difficulty: learning the joint probability

- **Naive Bayes classification**

-Assumption that **all input features are conditionally independent!**

$$\begin{aligned} P(X_1, X_2, \cdots, X_n \mid C) &= P(X_1 \mid X_2, \cdots, X_n, C)P(X_2, \cdots, X_n \mid C) \\ &= P(X_1 \mid C)P(X_2, \cdots, X_n \mid C) \\ &= P(X_1 \mid C)P(X_2 \mid C) \cdots P(X_n \mid C) \end{aligned}$$

-MAP classification rule: for $\mathbf{x} = (x_1, x_2, \cdots, x_n)$

$$[P(x_1 \mid c^*) \cdots P(x_n \mid c^*)]P(c^*) > [P(x_1 \mid c) \cdots P(x_n \mid c)]P(c), \quad c \neq c^*, c = c_1, \cdots, c_L$$

- Algorithm: Discrete-Valued Features

-Learning Phase: Given a training set **S**,

For each target value of c_i $(c_i = c_1, \cdots, c_L)$

$\hat{P}(C = c_i) \leftarrow$ estimate $P(C = c_i)$ with examples in **S**;

For every feature value x_{jk} of each feature X_j $(j = 1, \cdots, n; k = 1, \cdots, N_j)$

$\hat{P}(X_j = x_{jk} \mid C = c_i) \leftarrow$ estimate $P(X_j = x_{jk} \mid C = c_i)$ with examples in **S**;

Output: conditional probability tables; for X_j, $N_j \times L$ elements

-Test Phase: Given an unknown instance $\mathbf{X}' = (a_1', \cdots, a_n')$

Look up tables to assign the label c^* to $\mathbf{X}'$ if

$$[\hat{P}(a_1' \mid c^*) \cdots \hat{P}(a_n' \mid c^*)]\hat{P}(c^*) > [\hat{P}(a_1' \mid c) \cdots \hat{P}(a_n' \mid c)]\hat{P}(c), \quad c \neq c^*, c = c_1, \cdots, c_L$$

- **Algorithm: Continuous-valued Features**

  - Numberless values for a feature

  - Conditional probability often modeled with the normal distribution

$$\hat{P}(X_j \mid C = c_i) = \frac{1}{\sqrt{2\pi\sigma_{ji}}} \exp\left(-\frac{(X_j - \mu_{ji})^2}{2\sigma_{ji}^2}\right)$$

μ_{ji} : mean (avearage) of feature values X_j of examples for which $C = c_i$

σ_{ji} : standard deviation of feature values X_j of examples for which $C = c_i$

- **Learning Phase:** for $\mathbf{X} = (X_1, \cdots, X_n)$, $C = c_1, \cdots, c_L$

  Output: $n \times L$ normal distributions and $P(C = c_i)$ $i = 1, \cdots, L$

- **Test Phase:** Given an unknown instance $\mathbf{X}' = (a_1', \cdots, a_n')$

  -Instead of looking-up tables, calculate conditional probabilities with all the normal distributions achieved in the learning phrase

  -Apply the MAP rule to make a decision

Gaussian naive Bayes

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. For example, suppose the training data contains a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class. Let μ_k be the mean of the values in x associated with class C_k, and let σ_k^2 be the variance of the values in x associated with class C_k. Suppose we have collected some observation value v. Then, the probability *distribution* of v given a class C_k, $p(x = v \mid C_k)$, can be computed by plugging v into the equation for a Normal distribution parameterized by μ_k and σ_k^2. That is,

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Another common technique for handling continuous values is to use binning to discretize the feature values, to obtain a new set of Bernoulli-distributed features; some literature in fact suggests that this is necessary to apply naive Bayes, but it is not, and the discretization may throw away discriminative information

CODE:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('sales.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
#Printing the classification report
from sklearn.metrics import classification_report,accuracy_score
print(classification_report(y_test, y_pred))
print("Accuracy is " ,accuracy_score(y_test,y_pred)*100)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step
= 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('yellow', 'brown')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('yellow', 'brown'))(i), label = j)
```

```
plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()


# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step
= 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('yellow', 'brown')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('yellow', 'brown'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

OUTPUT :

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0

Confusion Matrix

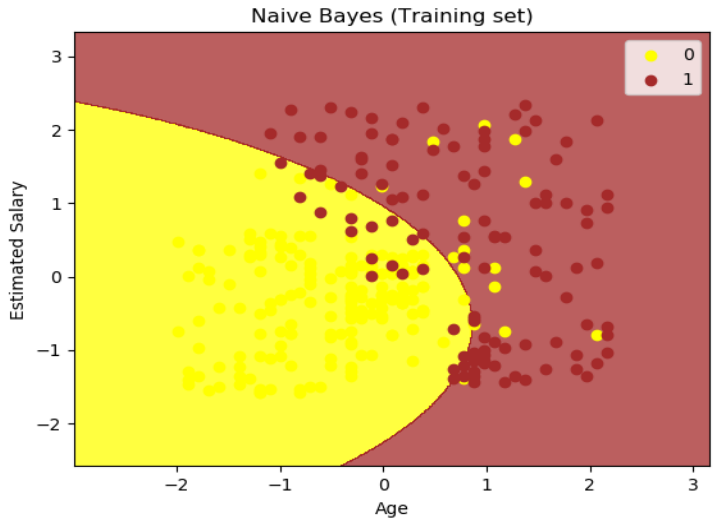
	0	1
0	65	3
1	7	25

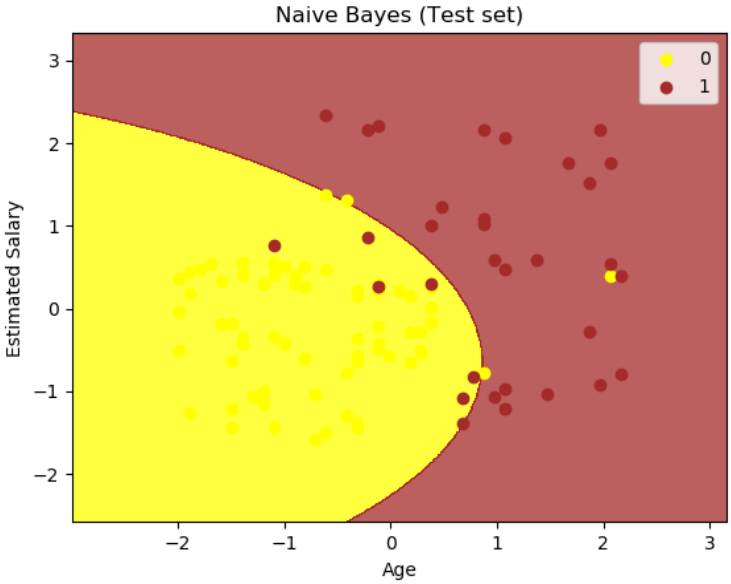
```
In [8]: from sklearn.metrics import classification_report
   ...: print(classification_report(y_test, y_pred))
             precision    recall  f1-score   support

          0       0.90      0.96      0.93        68
          1       0.89      0.78      0.83        32

avg / total       0.90      0.90      0.90       100
```

Accuracy is 90.


Naive Bayes (Training set)
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3


Naive Bayes (Test set)
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3

DISCUSSION :

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable**.**

Despite the fact that the far-reaching independence assumptions are often inaccurate, the naive Bayes classifier has several properties that make it surprisingly useful in practice. In particular, the decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one-dimensional distribution. This helps alleviate problems stemming from the curse of dimensionality, such as the need for data sets that scale exponentially with the number of features. While naive Bayes often fails to produce a good estimate for the correct class probabilities, this may not be a requirement for many applications.

FINDING AND LEARNING :

In this experiment we implemented naïve bayes(Gaussian distribution for continuous variable) using sklearn library in python on a sales dataset which classifies whether a person will by a certain product given his/her age and salary. And We got an accuracy of 90%.