Aim: To implement Naïve Bayesian classification

Objective: Develop a program to implement Bayesian classification.

Theory:

The Naive Bayes is a classification algorithm that is suitable for binary and multiclass classification. Naïve Bayes performs well in cases of categorical input variables compared to numerical variables. It is useful for making predictions and forecasting data based on historical results.

The naïve Bayesian classifier, or simple Bayesian classifier, works as follows:

1) Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, X = (x1, x2,...,xn), depicting n measurements made on the tuple from n attributes, respectively, A1, A2,..., An.
2) Suppose that there are m classes, C1,C2,...,Cm. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naïve Bayesian classifier predicts that tuple X belongs to the class Ci if and only if
   P(Ci|X) > P(Cj|X) Thus we maximize P(Ci|X).The class Ci for which P(Ci|X) is maximized is called the maximum posteriori hypothesis.
   By Bayes' theorem,
   P(Ci|X) = P(X|Ci)*P(Ci)/ P(X)
3) As P(X) is constant for all classes, only P(X|Ci)P(Ci) need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, P(C1) = P(C2) = ⋯ = P(Cm), and we would therefore maximize P(X|Ci). Otherwise, we maximize P(X|Ci)P(Ci). Note that the class prior probabilities may be estimated by P(Ci)=|Ci,D|/|D|, where |Ci,D| is the number of training tuples of class Ci in D.

The equation: Posterior = Prior x (Likelihood over Marginal probability)
There are four parts:

- Posterior probability (updated probability after the evidence is considered)
- Prior probability (the probability before the evidence is considered)
- Likelihood (probability of the evidence, given the belief is true)
- Marginal probability (probability of the evidence, under any circumstance)

Bayes' Rule can answer a variety of probability questions, which help us (and machines) understand the complex world we live in.

Example:

Car No	Color	Type	Origin	Stolen
1	Red	Sports	Domestic	Yes
2	Red	Sports	Domestic	No
3	Red	Sports	Domestic	Yes
4	Yellow	Sports	Domestic	No
5	Yellow	Sports	Imported	No
6	Yellow	SUV	Imported	No
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	No
10	Red	Sports	Imported	Yes

P (yes) =5/10
P (No)=5/10

-Color:
P(Red/Y)=3/5 P(yellow/Y)=2/5
P(Red/N)=2/5 P(yellow/N)=3/5

-Type:
P(SUV/Y)=1/5 P(Sports/Y)=4/5
P(SUV/N)=3/5 P(Sports/N)=2/5

-Origin:
P(Domentic/Y)=2/5 P(Imported/Y)=3/5
P(Domentic /N)=3/5 P(Imported/N)=2/5

P(x|Yes).P(Yes)= 0.024
P(x|No).P(No)=0.072
So, Bayesian Classification Predicts the class "NO"

Code:

```
def table(s, r):  # cal count of selected tuples with yes and no
    l1 = []
    c = -1
    for x in r:
        c += 1
        if x == s:
            l1.append(st[c])
    pj = l1.count("yes")
    nj = l1.count("no")
    print(s, "for yes: ", pj, s, "for no : ", nj)
    return pj, nj

color = ['red', 'red', 'red', 'yellow', 'yellow',
         'yellow', 'yellow', 'yellow', 'red', 'red']
typ = ['sports', 'sports', 'sports', 'sports',
       'sports', 'SUV', 'SUV', 'SUV', 'SUV', 'sports',]
origin = ['Domestic', 'Domestic', 'Domestic', 'Domestic', 'Imported',
          'Imported', 'Imported', 'Domestic', 'Imported', 'Imported']
st = ['yes', 'no', 'yes', 'no', 'yes', 'no', 'yes', 'no', 'no', 'yes']
ty = st.count('yes')  # no of yes tuples
tn = st.count('no')  # no of no tuples
py = ty/len(st)  # p(yes/total no of tuples)
pn = tn/len(st)  # p(no/total no of tuples)
print('yes/total no of tuples :', py, '| no/total no of tuples :', pn)
y, n = table("red", color)  # X = color:red | type: SUV | origin :domestic
y1, n1 = table('SUV', typ)
y2, n2 = table("Domestic", origin)
pyx = (y*y1*y2*py)/(ty*ty*ty)  # p(X/yes)
pnx = (n*n1*n2*pn)/(tn*tn*tn)  # p(X/no)
print('for the tuple X = (color=red , type =SUV, origin = Domestic)')
print('p(x/yes) = ', pyx, ' p(x/no) = ', pnx)
if pyx > pnx:
    print("yes has highest probability")
else:
    print("no has highest probability")
```

Output:

```
PS D:\Vartak college\SEM 5\DWM\code> py .\naive_bayes.py
yes/total no of tuples : 0.5 | no/total no of tuples : 0.5
red for yes:  3 red for no :  2
SUV for yes:  1 SUV for no :  3
Domestic for yes:  2 Domestic for no :  3
for the tuple X = (color=red , type =SUV, origin = Domestic)
p(x/yes) =  0.024  p(x/no) =  0.072
no has highest probability
```

Conclusion:

Thus, we have learned to implement Naïve Bayesian classification using python. The Naive Bayes is a classification algorithm that is suitable for binary and multiclass classification. Naïve Bayes performs well in cases of categorical input variables compared to numerical variables.