Concept

The plain Bayesian classification model is a simple way to construct classifiers.The plain Bayesian classification model divides the problem into two categories, feature vectors and decision vectors, and assumes that the feature variables of the problem all act independently of each other on the decision variables, i.e., the features of the problem are all uncorrelated with each other. Despite this oversimplification assumption, the plain Bayesian classification model can exponentially reduce the complexity of Bayesian network construction, and also better handle the noise and irrelevant attributes of the training samples, so the plain Bayesian classification model still has efficient applications in many real-world problems, such as intrusion detection and text classification.

Principle

Bayesian theory originates from Bayes' theorem and Bayes' postulate proposed by Bayes. Bayes' theorem introduces the prior probability, and the posterior probability is calculated from the prior probability and the class of conditional probability expressions. Suppose there are random variables x and y，p(x，y) denotes their joint probability，p(x|y) and p(y|x) denote the conditional probability，where p(y|x) is the posterior probability, while p(y) is called the prior probability of y. The joint probability and conditional probability of x and y satisfy the following relationship：

$$p(y,x) = p(y|x)p(x) = p(x|y)p(y)$$

After the exchange we get：

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

The above formula is Bayes' theorem, which provides a way to calculate the posterior probability p(y|x) from the prior probability p(y).

Assuming that the characteristic vector of the problem is X = {x1, x2, ..., xn} and that X1, X2, ..., Xn} are independent of each other, then p(x|y) can be decomposed into the product of multiple vectors:

$$p(x|y) = \prod_{i=1}^{n} p(x_i, y)$$

Then this problem can be solved by the plain Bayesian classifier:

$$p(y|x) = \frac{p(y)\prod_{i=1}^{n} p(x_i,y)}{p(x)}$$

where p(x) is a constant and the prior probability p(y) can be estimated by the proportion of samples from each class in the training set. Given Y = y, if the classification of test sample x is to be estimated, the posterior probability of y obtained from the plain Bayesian classification is：

$$p(y=Y|x) = \frac{p(y=Y)\prod_{i=1}^{n} p(x_i|y=Y)}{p(x)}$$

So finally it is enough to find the maximum class y of $p(y=Y)\prod_{i=1}^{n} p(x_i|y=Y$

Example1

In [13]:
```
# import the Plain Bayesian Module
from sklearn.naive_bayes import MultinomialNB
import numpy as np
# The features of the custom test dataset x x have 4 dimensions
x=np.array([[0,1,0,1],
           [1,1,1,0],
           [0,1,1,0],
           [0,0,0,1],
           [0,1,1,0],
           [0,1,0,1],
           [1,0,0,1]]
          )
# y is the label corresponding to x
y=np.array([0,1,1,0,1,0,0])
clf=MultinomialNB()
clf.fit(x,y)
# Enter the next day into the model
Next_Day=[[0,0,1,0]]
pre=clf.predict(Next_Day)
pre2=clf.predict_proba(Next_Day)
# Output model prediction results
print("prediction results: ",pre)
# Output model predicted classification probabilities
print("probabilities: ",pre2)
# There are two predicted probability values which correspond to the probab
ility that the categorization is 0 and 1.
# The predicted result is the categorization with the higher probability va
lue.
```

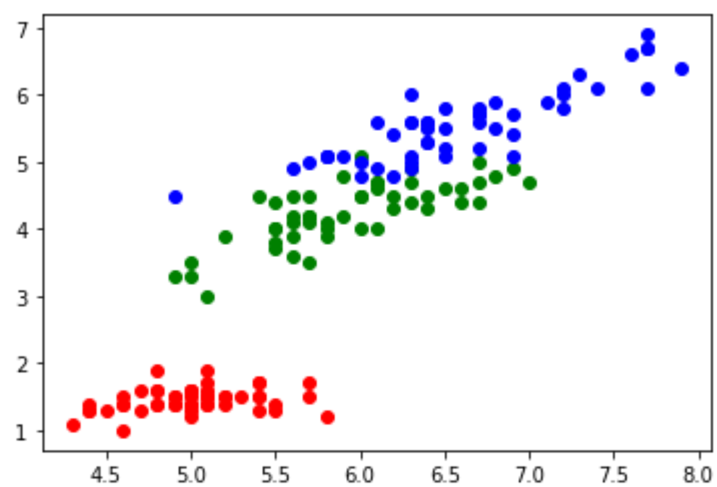
```
prediction results:  [1]
probabilities:  [[0.25 0.75]]
```

Example2

Using the iris dataset in sklearn.dataset

In [15]:
```
#  import the Plain Bayesian Module
from sklearn.naive_bayes import GaussianNB
import numpy as np
import pandas as pd
from pandas import Series,DataFrame
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
%matplotlib inline
muNB = GaussianNB()
iris=load_iris()
X, y = load_iris(return_X_y=True)
```

In [16]:
```
iris.data.shape  # Iris data type is 150*4 150 rows The 4 columns of each r
ow represent the 4 characteristics of the iris
```
Out[16]:
```
(150, 4)
```
In [17]:
```
print('Target values/label values for the Iris dataset: \n', iris.target)
print('Name of the Iris dataset features: \n', iris.feature_names)
print('Name of the target value of the Iris dataset: \n', iris.target_names)
```
```
Target values/label values for the Iris dataset:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Name of the Iris dataset features:
 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal widt
h (cm)']
Name of the target value of the Iris dataset:
 ['setosa' 'versicolor' 'virginica']
```
In [18]:
```
# Select two-dimensional data to see the distribution of the data
for i in range(len(X)):
    if y[i]==0:
        plt.scatter([X[i][0]], [X[i][2]],c="r")
    elif y[i]==1:
        plt.scatter([X[i][0]], [X[i][2]],c="g")
    else:
        plt.scatter([X[i][0]], [X[i][2]],c="b")

plt.show()
```


In [19]:
```
# train_test_split function is used to divide the data set, according to te
st_size=0.2,
# 80% of the data will be divided into training set and 20% of the data wil
l be divided into test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
ndom_state=0)
```

In [20]:
```
# Training data
muNB.fit(X_train, y_train)
```
Out[20]:
```
GaussianNB()
```
In [21]:
```
muNB.score(X_test, y_test) # Prediction Accuracy
```
Out[21]:
```
0.9666666666666667
```

Please use the Bayesian model to train the load_breast_cancer dataset in sklearn

In [26]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.naive_bayes import GaussianNB
```
In [27]:
```
muNB = GaussianNB()
breast=load_breast_cancer()
X, y = load_breast_cancer(return_X_y=True)
```
In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
ndom_state=0)
```
In [29]:
```
muNB.fit(X_train, y_train)
```
Out[29]:
```
GaussianNB()
```
In [30]:
```
muNB.score(X_test, y_test) # Prediction Accuracy
```
Out[30]:
```
0.9298245614035088
```
In [ ]:
```

```
In [ ]:
```

```