Bayes Classifier on Iris Dataset

张栋玮 19373703 Open in GITHUB

Abstract

Naive Bayes is a generative classification algorithm which is based on bayesian theorem and conditional independence assumption. By using bayesian theorem, we can compute posterior probability with prior probabilities and observations. Bayesian theorem is shown as follows.

$$P(\omega_i|X)=\frac{P(X|\omega_i)P(\omega_i)}{\sum_{j=1}^{c}P(X|\omega_j)P(\omega_j)} \qquad \text{i=1,…, c}$$

Let the series of decision actions as $a_1, a_2, .., a_c$, the conditional risk $R(a|X)$ of decision action a_i with loss function $\lambda(a,w)$ can be computed by

$$R(a_i|X)=\sum_{j=1,j\neq i}^{c}\lambda(a_i,\omega_j)P(\omega_j|X), \qquad \text{i=1,…, c}$$

Thus the minimum risk Bayesian decision can be found as

$$a_k^* = \text{Arg min}_i\ R(a_i|X), \qquad \text{i=1,…, c}$$

1. Algorithm

1.1 Data Preprocess

Here I load the Iris dataset from sklearn.datasets. Then split it into train set and test set with $test\_size = 0.2, random\_state = 3$.

In [1]:
```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split

# Load data
def load_data():
    iris = datasets.load_iris()
    x = iris.data
    y = iris.target
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2, random_state
    return x_train,x_test,y_train,y_test
```

1.2 Bayes Class

This is the python class named bayes. It contains four parts with details in comments.

- Initialization and training
- Predictions
- Computation of accuracy
- Data print

In [2]:
```
class Bayes():

    # Initialize and train
    def __init__(self, x_train, y_train):
        self.categories = len(np.unique(y_train))  # 3 classes in the dataset
        self.total_col = x_train.shape[1]   # Number of Attributes used
        self.partial = []   # Split proportion
        self.mean = np.zeros([self.categories,self.total_col]) # Initialization
        self.var = np.zeros([self.categories,self.total_col])

        # Compute the proportion, variance and mean of each class
        for i in range(self.categories):
            temp = x_train[np.nonzero(i==y_train)] # Select i_th class
            self.partial.append(len(temp)/len(x_train))
            self.mean[i,:] = np.mean(temp,axis=0,keepdims=True)
            self.var[i,:] = np.var(temp,axis=0,keepdims=True)


    # Make predictions
    def predict(self, x_test,y_test):
        result = []
        eps = 1e-10
        for i in x_test:
            x = np.tile(i,(3,1))

            # Compute the Gaussian pdf
            num = -(x-self.mean+eps)**2
            den = 2*self.var+eps
            _exp = np.exp(num/den)
            # _exp = np.exp(-(x-self.mean)**2/(2*self.var+eps))

            # Compute the posterior possibilities
            p = _exp/(np.sqrt(2*np.pi*self.var)+eps)
            # Change the possibilities into log() mode
            log_p = np.sum(np.log(p),axis=1)
            prob = np.log(self.partial)+log_p
            result.append(np.argmax(prob))
        return result

    # Compute the accuracy
    def acc(self, y_test, y_pred):
        acc = np.count_nonzero(y_test==y_pred)
        return acc/len(y_pred)

    # Print parameters
    def printPara(self):
        print(f"The dataset has {self.categories} categories and {self.total_col} at
        print("Proportion of each class:")
        print(self.partial)
```

```
        print("Mean:")
        print(self.mean)
        print("Var:")
        print(self.var)
```

2. Test

In this part, I first computed the parameters of the whole Iris dataset. From the parameters, it is clear that Attribute 3 and 4 are more significant in classification. By doing the following experiments, it shows that using Attribute 3 and 4 to classify the dataset gives a better performance, though it is relative to the split of dataset.

In [3]:
```
# Using all 4 attributes
x_train,x_test,y_train,y_test=load_data()
TotalAttr = Bayes(x_train,y_train)
TotalAttr.printPara()
y_pred = TotalAttr.predict(x_test,y_test)
acc = TotalAttr.acc(y_test,y_pred)
print(f'Accuracy: {acc:.2f}')
```

```
The dataset has 3 categories and 4 attributes.
Proportion of each class:
[0.3333333333333333, 0.3333333333333333, 0.3333333333333333]
Mean:
[[5.03   3.4325 1.465  0.2375]
 [5.93   2.7875 4.2675 1.335 ]
 [6.5525 2.9875 5.5325 2.01  ]]
Var:
[[0.1011     0.13219375 0.032775   0.01134375]
 [0.2351     0.10009375 0.21819375 0.040275  ]
 [0.38749375 0.10309375 0.28519375 0.0649    ]]
Accuracy: 0.97
```

In [4]:
```
# Only take attribute 3 and 4 into consideration
x_train,x_test,y_train,y_test=load_data()
x_train = x_train[:,2:]
x_test = x_test[:,2:]
Attr34 = Bayes(x_train,y_train)
Attr34.printPara()
y_pred = Attr34.predict(x_test,y_test)
acc = Attr34.acc(y_test,y_pred)
print(f'Accuracy: {acc:.2f}')
```

```
The dataset has 3 categories and 2 attributes.
Proportion of each class:
[0.3333333333333333, 0.3333333333333333, 0.3333333333333333]
Mean:
[[1.465  0.2375]
 [4.2675 1.335 ]
 [5.5325 2.01  ]]
Var:
[[0.032775   0.01134375]
 [0.21819375 0.040275  ]
 [0.28519375 0.0649    ]]
Accuracy: 1.00
```

3. Minimum Risk Bayes

In this part, I first implement minimum risk bayes dicision on given data. Then test it on the Iris dataset.

In [5]:
```
w1=[-3.9847,-3.5549,-1.2401,-0.9780,-0.7932,-2.8531,-2.7605,-3.7287,
-3.5414,-2.2692,-3.4549,-3.0752,-3.9934, -0.9780,-1.5799,-1.4885,
-0.7431,-0.4221,-1.1186,-2.3462,-1.0826,-3.4196,-1.3193,-0.8367,
-0.6579,-2.9683]
w2 =  [2.8792,  0.7932,1.1882,3.0682,4.2532,0.3271,0.9846,2.7648,2.6588]

w = [*w1,*w2]
y = (np.sign(w)+1)/2    #
total = len(w)
pw = [0.9,0.1]
loss = [[0,1],[6,0]]
var = [np.var(w1),np.var(w2)]
mean = [np.mean(w1),np.mean(w2)]
```

In [6]:
```
def GaussPdf(x,mean,var):
    eps=1e-10
    _exp = np.exp(-(x-mean)**2/(2*var+eps))
    p = _exp/(np.sqrt(2*np.pi*var)+eps)
    return p

# x: data  w: class
def risk_pred(x):
    den = GaussPdf(x,mean[0],var[0])+GaussPdf(x,mean[1],var[1])
    num = [loss[0][1]*pw[1]*GaussPdf(x,mean[1],var[1]), loss[1][0]*pw[0]*GaussPdf(x,
    risk = num/den
    prediction = np.argmin(risk)
    return prediction
```

In [7]:
```
r_pred = []
for i in w :
    pred = risk_pred(i)
    r_pred.append(pred)
correct = np.count_nonzero(r_pred==y)
acc = correct/total
print(f"Accuracy: {acc:.2f}")
```

```
Accuracy: 0.89
```

Reference

[1] Statistical Pattern Recognition Lab, SASEE

[2] 朴素贝叶斯算法

[3] 朴素贝叶斯处理鸢尾花数据集分类