# 第六章

## 一、高斯贝叶斯分类器和高斯朴素贝叶斯分类器预测样本 x 的类别

1. 设密度 $x_1$，含糖率 $x_2$，好瓜 $C$（$c_1$=是，$c_2$=否）

由贝叶斯公式得 $p(c=c_j \mid x_1, x_2) = \dfrac{p(x_1, x_2 \mid c=c_j)\,p(c=c_j)}{p(x_1, x_2)} \propto p(x_1, x_2 \mid c=c_j)\cdot p(c=c_j)$

① 高斯贝叶斯.

多维高斯：$p(x_1, x_2 \mid c=c_j) = \dfrac{1}{2\pi}\dfrac{1}{\cdot|\Sigma_j|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j)\right)$

这里 $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ $\mu_j = \begin{bmatrix} \mu_{1j} \\ \mu_{2j} \end{bmatrix} \leftarrow \underset{\text{均值}}{\mu_{ij}} = \dfrac{1}{N(c=c_j)}\sum_{x_i \in c_j} x_i$ $\underset{\text{协方差}}{\Sigma_j} = \begin{bmatrix} cov(x_1, x_1) & cov(x_1, x_2) \\ cov(x_2, x_1) & cov(x_2, x_2) \end{bmatrix}$

$\downarrow$

使用numpy cov.方法

无偏估计

$p(c=c_1) = \dfrac{N(c=c_1)}{N_{all}} = \dfrac{8}{17}$ $p(c=c_2) = \dfrac{N(c=c_2)}{N_{all}} = \dfrac{9}{17}$

a. 当 $c=c_1$ 时

把样本数据代入得.

$p(0.5, 0.3 \mid c_1) = 9.490316446182263$ $p(c=c_1) = \dfrac{8}{17}$

$\therefore p(c=c_1 \mid 0.5, 0.3) = p(0.5, 0.3 \mid c_1)\cdot p(c=c_1) = 4.466031268791653 \Rightarrow P_{good}$.

b. 当 $c=c_2$ 时

把样本数据代入得

$p(0.5, 0.3 \mid c_2) = 2.8854275027464418$ $p(c=c_2) = \dfrac{9}{17}$

$\therefore p(c=c_2 \mid 0.5, 0.3) = p(0.5, 0.3 \mid c_2)\cdot p(c=c_2) = 1.527579266159881 \Rightarrow P_{bad}$

$\because P_{good} > P_{bad} \Rightarrow$ 样本 $x = \begin{bmatrix} 0.5 \\ 0.3 \end{bmatrix}$ 是好瓜.

② 高斯朴素贝叶斯

独立性假设.

$$P(x_1,x_2 \mid c=c_j) = P(x_1 \mid c=c_j)\,P(x_2 \mid c=c_j)$$

又 $P(x_i \mid c=c_j) = \frac{1}{\sqrt{2\pi}\,\sigma_{ij}} \exp\left(-\frac{(x_i-\mu_{ij})^2}{\sigma_{ij}^2}\right)$

均值 $\mu_{ij} = \frac{1}{N(c=c_j)} \sum_{x_i \in c_j} x_i$

方差 $\sigma_{ij}^2 = \frac{1}{N(c=c_j)} \sum_{x_i \in c_j} (x_i-\mu_{ij})^2$

$$\Rightarrow P(x_1,x_2 \mid c=c_j) = \prod_{i=1}^{2} \frac{1}{\sqrt{2\pi}\,\sigma_{ij}} \exp\left(-\frac{(x_i-\mu_{ij})^2}{\sigma_{ij}^2}\right)$$

$$= \frac{1}{2\pi} \frac{1}{|\Sigma_j|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1} (x-\mu_j)\right\}$$

→ 可以观察到和高斯贝叶斯公式一致，不过这里的 $\Sigma_j = \begin{bmatrix} \sigma_{1j}^2 & 0 \\ 0 & \sigma_{2j}^2 \end{bmatrix}$

这里 $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ $\mu_j = \begin{bmatrix} \mu_{1j} \\ \mu_{2j} \end{bmatrix}$

a. 当 $c=c_1$ 时 把数据代入得.

$P(0.5, 0.3 \mid c_1) = 11.289349211617608$ $\quad P(c=c_1) = \frac{8}{17}$

$P(c_1 \mid 0.5, 0.3) = 5.312634923114468$. $\Rightarrow P_{good\_naive}$

b. 当 $c=c_2$ 时 把数据代入得

$P(0.5, 0.3 \mid c_2) = 4.231758950693437$ $\quad P(c=c_2) = \frac{9}{17}$

$P(c_2 \mid 0.5, 0.3) = 2.2403429738965257$ $\Rightarrow P_{bad\_naive}$

$\because P_{good\_naive} > P_{bad\_naive} \Rightarrow$ 样本 $x = \begin{bmatrix} 0.5 \\ 0.3 \end{bmatrix}$ 是好瓜.

代码：

（1）读取样本数据，并分类好瓜和坏瓜

```python
def read_data(path):
    # 密度  含糖率 好瓜
    data = pd.read_csv(path, header=None, names=['x1', 'x2', 'c'])
    return data


```

```python
#  将 c=c1 和 c=c2 两种情况下的 x1 x2 分离
def split_data(path="data.txt"):
    data = np.array(read_data(path))
    x1_x2_good = np.array([x[0:2] for x in data if x[2] == '是'])
    x1_x2_bad = np.array([x[0:2] for x in data if x[2] == '否'])
    return x1_x2_good, x1_x2_bad
```

（2）求均值、协方差、方差

```python
 # 求均值 x[x1,x2]
def mean_value(x):
    x1_mean = np.mean(x[:, 0])
    x2_mean = np.mean(x[:, 1])
    return np.array([x1_mean, x2_mean])


# 求协方差 x[x1 x2] 2D 数据
def cov(x):
    x_cov = np.cov(x.astype(float).T)  # 默认无偏估计，即分母为 n-1
    return x_cov


# 方差 x[x1,x2]
def var_value(x):
    x1_var = np.var(x[:, 0])
    x2_var = np.var(x[:, 1])
    return np.array([x1_var, x2_var])
```

（3）高斯贝叶斯和高斯朴素贝叶斯通用的公式

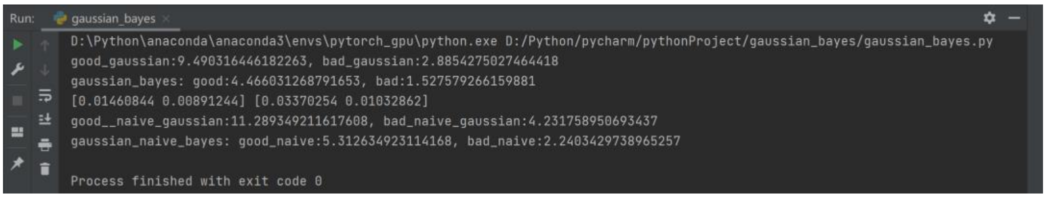
```python
#  高斯贝叶斯/高斯朴素贝叶斯
def gaussian_bayes(x_mean, x_cov, x_input):
 x_cov_det = np.linalg.det(x_cov)  # 协方差矩阵行列式
 x_cov_inv = np.linalg.inv(x_cov)  # 协方差矩阵逆矩阵
 temp = x_input - x_mean
 gaussian = 1 / (2 * np.pi) * 1 / (pow(x_cov_det, 0.5)) * np.exp(-
(1/2 * np.dot(np.dot(temp, x_cov_inv), temp)))
 return gaussian
```

（4）主函数调用

```python
def main():
 x_good, x_bad = read_data.split_data()  # x[x1,x2]
 # 均值
 x_good_mean = mean_value(x_good)
```

```python
x_bad_mean = mean_value(x_bad)
# 协方差矩阵
x_good_cov = cov(x_good)
x_bad_cov = cov(x_bad)
x = np.array([0.5, 0.3])
# 先验概率
p_good = len(x_good)/(len(x_good) + len(x_bad))
p_bad = len(x_bad)/(len(x_good) + len(x_bad))

# 高斯贝叶斯
good_gaussian = gaussian_bayes(x_good_mean, x_good_cov, x)
bad_gaussian = gaussian_bayes(x_bad_mean, x_bad_cov, x)
print("good_gaussian:%s, bad_gaussian:%s" % (good_gaussian, bad_gaussian))
good = good_gaussian * p_good
bad = bad_gaussian * p_bad
print("gaussian_bayes: good:%s, bad:%s" % (good, bad))

# 高斯朴素贝叶斯
# 方差
var_good = var_value(x_good)
var_bad = var_value(x_bad)
print(var_good, var_bad)
# 协方差矩阵
cov_good_naive = np.zeros((2, 2))
cov_good_naive[0][0] = var_good[0]
cov_good_naive[1][1] = var_good[1]
cov_bad_naive = np.zeros((2, 2))
cov_bad_naive[0][0] = var_good[0]
cov_bad_naive[1][1] = var_good[1]

good_naive_gaussian = gaussian_bayes(x_good_mean, cov_good_naive, x)
bad_naive_gaussian = gaussian_bayes(x_bad_mean, cov_bad_naive, x)
print("good__naive_gaussian:%s, bad_naive_gaussian:%s" % (good_naive_gaussian, bad_naive_gaussian))
good_naive = good_naive_gaussian * p_good
bad_naive = bad_naive_gaussian * p_bad
print("gaussian_naive_bayes: good_naive:%s, bad_naive:%s" % (good_naive, bad_naive))
```

运行截图：



## 二、Markov 模型

2. 由题可得状态转移矩阵 $A = \begin{bmatrix} 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix}$

第一次没中到第四次命中转移三次 $0 \longrightarrow 0 \longrightarrow 0 \longrightarrow 0$

$$A^3 = \begin{bmatrix} 0.722 & 0.278 \\ 0.695 & 0.305 \end{bmatrix}$$

∵ 第一次没中，第四次命中

∴ $A^3[1][0] = 0.695$ 是第四次命中的概率.

代码：

```python
# 返回矩阵的 n 阶结果
def get_matrix_pow(matrix, n):
    ret = matrix
    for i in range(n-1):
        ret = np.dot(ret, matrix)
    return ret


def main():
    A = np.array([[0.8, 0.2],
                  [0.5, 0.5]])
    A_3 = get_matrix_pow(A, 3)
    print("第一次没中且第四次射中的概率为：%.3f" % A_3[1][0])
```

三、使用 EM 求解 GMM

3. $$\underset{\theta}{\arg\max}\ \ln\left(\prod_{j=1}^{n} P(x_j|\theta)\right) = \underset{\theta}{\arg\max} \sum_{j=1}^{n} \ln\left(\sum_{i=1}^{k} P(y_j=i, x_j|\theta)\right)$$

构造目标函数 $\sum_{j=1}^{n}\sum_{i=1}^{k} P_{ji} \ln(P(y_j=i, x_j|\theta))$ $\quad P_{ji} = P(P_j=i|x_j, \theta^t)$

(1) 目标函数关于 $\mu_i$ 求偏导

$$\sum_{j=1}^{n} P_{ji} \frac{\partial \ln(P(y_j=i, x_j|\theta))}{\partial \mu_i} = \sum_{j=1}^{n} P_{ji} \frac{\partial \ln(P(y_j=i)\cdot P(x_j|y_j=i,\theta))}{\partial \mu_i}$$

$$= \sum_{j=1}^{n} P_{ji} \frac{\partial \ln(\pi_i P(x_j|y_j=i,\theta))}{\partial \mu_i} = \sum_{j=1}^{n} P_{ji} \frac{1}{\pi_i P(x_j|y_j=i,\theta)} \cdot \frac{\partial(\pi_i P(x_j|y_j=i,\theta))}{\partial \mu_i}$$

$$= \sum_{j=1}^{n} P_{ji} \frac{\pi_i P(x_j|y_j=i,\theta)}{\pi_i P(x_j|y_j=i,\theta)} \cdot \frac{\partial[-\frac{1}{2}(x_j-\mu_i)^T \Sigma_i^{-1}(x_j-\mu_i)]}{\partial \mu_i}$$

$$= \sum_{j=1}^{n} P_{ji} \cdot \frac{1}{2} \cdot \frac{\partial[(x_j-\mu_i)^T \Sigma_i^{-1}(x_j-\mu_i)]}{\partial(x_j-\mu_i)}$$

由 $\frac{\partial(x^TAx)}{\partial x} = (A+A^T)x$ 当 $A=A^T$ 时 $\frac{\partial(x^TAx)}{\partial x} = 2Ax$

得 原式 $= \sum_{j=1}^{n} P_{ji}\ \Sigma_i^{-1}(x_j-\mu_i) = 0$

即 $\Sigma_i^{-1} \sum_{j=1}^{n} P_{ji}(x_j-\mu_i) = 0$

$\because \Sigma_i^{-1} \neq 0 \quad \therefore \sum_{j=1}^{n} P_{ji} x_j = \sum_{j=1}^{n} P_{ji}\mu_i \Rightarrow \mu_i = \frac{\sum_{j=1}^{n} P_{ji} x_j}{\sum_{j=1}^{n} P_{ji}}$ 其中 $P_{ji} = P(P_j=i|x_j, \theta^t$

2) 目标函数关于 $\Sigma_i$ 求导

$$\sum_{j=1}^{n} P_{ji}\frac{\partial \ln(P(y_j=i,x_j|\theta))}{\partial \Sigma_i} = \sum_{j=1}^{n} P_{ji}\frac{\partial[\ln|\Sigma_i| + (x_j-\mu_i)^T\Sigma_i^{-1}(x_j-\mu_i)]}{\partial \Sigma_i}$$

又：$\frac{\partial|A|}{\partial A} = |A|\cdot A^{-1} \Rightarrow \frac{\partial \ln|\Sigma_i|}{\partial \Sigma_i} = \Sigma_i^{-1}$

$$(x_j-\mu_i)^T\Sigma_i^{-1}(x_j-\mu_i) = tr[(x_j-\mu_i)^T\Sigma_i^{-1}(x_j-\mu_i)] = tr[\Sigma_i^{-1}(x_j-\mu_i)^T(x_j-\mu_i)]$$

$$\frac{\partial tr(AB)}{\partial A} = B^T \Rightarrow \frac{\partial tr[\Sigma_i^{-1}(x_j-\mu_i)^T(x_j-\mu_i)]}{\partial \Sigma_i} = (x_j-\mu_i)(x_j-\mu_i)^T\frac{\partial \Sigma_i^{-1}}{\partial \Sigma_i}$$

$$= -(x_j-\mu_i)(x_j-\mu_i)^T\Sigma_i^{-2}$$

原式 $= \sum_{j=1}^{n} P_{ji}[\Sigma_i^{-1} - (x_j-\mu_i)(x_j-\mu_i)^T] = 0$

同乘 $\Sigma_i^2$ 得 $\sum_{j=1}^{n} P_{ji}[\Sigma_i - (x_j-\mu_i)(x_j-\mu_i)^T] = 0$

$$\Rightarrow \Sigma_i = \frac{\sum_{j=1}^{n} P_{ji}(x_j-\mu_i)(x_j-\mu_i)^T}{\sum_{j=1}^{n} P_{ji}}$$

(3) $\pi_i$. 此时这是一个有约束极值问题，约束条件为 $\sum_{i=1}^{k}\pi_i = 1$

构造拉格朗日函数.

$$L(\pi_1,\pi_2,\cdots,\pi_k,\lambda) = LL(D) + \lambda\left(\sum_{i=1}^{k}\pi_i - 1\right)$$

$$= \sum_{j=1}^{n}\ln\left(\sum_{i=1}^{K}P(y_j=i,x_j|\theta)\right) + \lambda\left(\sum_{i=1}^{k}\pi_i - 1\right)$$

$$= \sum_{j=1}^{n}\ln\left(\sum_{i=1}^{K}\pi_i P(x_j|y_j=i,\theta)\right) + \lambda\left(\sum_{i=1}^{k}\pi_i - 1\right)$$

$$\frac{\partial L}{\partial \pi_i} = \sum_{j=1}^{n}\frac{P(x_j|y_j=i,\theta)}{\sum_{i=1}^{K}\pi_i P(x_j|y_j=i,\theta)} + \lambda = 0$$

两边同乘 $\pi_i$ 得 $\sum_{j=1}^{n}(P_{ji} + \pi_i\lambda) = 0$ $\quad \sum_{i=1}^{K}\left[\sum_{j=1}^{n}(P_{ji}+\pi_i\lambda)\right] = \sum_{i=1}^{K}\sum_{j=1}^{n}P_{ji} + \lambda\sum_{i=1}^{K}\pi_i$

$$= n + \lambda = 0 \Rightarrow \lambda = -n$$

$$\therefore \sum_{j=1}^{n}(P_{ji} + \lambda\pi_i) = \sum_{j=1}^{n}(P_{ji} - n\pi_i) = 0 \Rightarrow \pi_i = \frac{1}{n}\sum_{j=1}^{n}P_{ji}.$$

代码：

（1）构建 GMM 类，包含初始化参数、EM 算法、迭代更新

```
class GMM:
    def __init__(self, k=2):
        self.k = k  # 定义聚类个数,默认值为 2
        self.p = None  # 样本维度
```

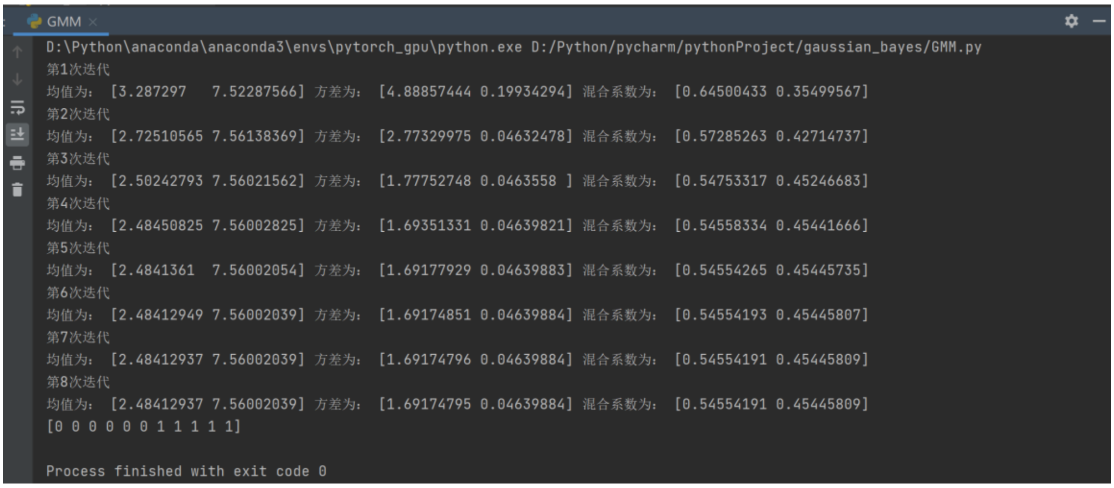
```
        self.n = None  # 样本个数
        # 声明变量
        self.params = {
            "πi": None,  # 混合系数 1*k
            "μ": None,  # 均值 k*p
            "cov": None,  # 协方差 k*p*p
            "pji": None  # 后验分布 n*k
        }

    # 初始化参数
    def init_params(self, init_μ):
        πi = np.ones(self.k) / self.k
        μ = init_μ
        cov = np.ones((self.k, self.p, self.p))
        pji = np.zeros((self.n, self.k))
        self.params = {
            "πi": πi,  # 混合系数 1*k
            "μ": μ,  # 均值 k*p
            "cov": cov,  # 协方差 k*p*p
            "pji": pji  # 后验分布 n*k
        }

    # 高斯公式
    def gaussian_function(self, x_j, μ_k, cov_k):
        one = -((x_j - μ_k) @ np.linalg.inv(cov_k) @ (x_j - μ_k).T) / 2
        two = -self.p * np.log(2 * np.pi) / 2
        three = -np.log(np.linalg.det(cov_k)) / 2
        return np.exp(one + two + three)

    # 计算 Pji 隐变量概率
    def E_step(self, x):
        πi = self.params["πi"]
        μ = self.params["μ"]
        cov = self.params["cov"]
        for j in range(self.n):
            x_j = x[j]
            pji_list = []
            for i in range(self.k):
                πi_k = πi[i]
                μ_k = μ[i]
                cov_k = cov[i]
                pji_list.append(πi_k * self.gaussian_function(x_j, μ_k, cov_k))
```

```
            self.params['pji'][j, :] = np.array([v / np.sum(pji_list) for v in pji_list])

    # 更新参数
    def M_step(self, x):
        μ = self.params["μ"]
        pji = self.params["pji"]
        for i in range(self.k):
            μ_k = μ[i]  # p
            pji_k = pji[:, i]  # n
            pji_k_j_list = []
            mu_k_list = []
            cov_k_list = []
            for j in range(self.n):
                x_j = x[j]  # p
                pji_k_j = pji_k[j]
                pji_k_j_list.append(pji_k_j)
                mu_k_list.append(pji_k_j * x_j)
            self.params['μ'][i] = np.sum(mu_k_list, axis=0) / np.sum(pji_k_j_list)
            for j in range(self.n):
                x_j = x[j]  # p
                pji_k_j = pji_k[j]
                cov_k_list.append(pji_k_j * np.dot((x_j - μ_k).T, (x_j - μ_k)))
            self.params['cov'][i] = np.sum(cov_k_list, axis=0) / np.sum(pji_k_j_list)
            self.params['πi'][i] = np.sum(pji_k_j_list) / self.n
        print("均值为：", self.params["μ"].T[0], end=" ")
        print("方差为：", self.params["cov"].T[0][0], end=" ")
        print("混合系数为：", self.params["πi"])

    # 迭代，返回聚类结果
    def fit(self, x, μ, max_iter=10):
        x = np.array(x)
        self.n, self.p = x.shape
        self.init_params(μ)

        for i in range(max_iter):
            print("第{}次迭代".format(i+1))
            self.E_step(x)
            self.M_step(x)
        return np.argmax(np.array(self.params["pji"]), axis=1)
```

（2）主函数调用

```python
def main():
    dataset = np.array([[1.0], [1.3], [2.2], [2.6], [2.8], [5.0], [7.3], [7.4], [7.5], [7.7], [7.9]])
    μ = np.array([[6], [7.5]])
    my_model = GMM(2)
    result = my_model.fit(dataset, μ, max_iter=8)
    print(result)
```

实验结果：



```
D:\Python\anaconda\anaconda3\envs\pytorch_gpu\python.exe D:/Python/pycharm/pythonProject/gaussian_bayes/GMM.py
第1次迭代
均值为： [3.287297   7.52287566] 方差为： [4.88857444 0.19934294] 混合系数为： [0.64500433 0.35499567]
第2次迭代
均值为： [2.72510565 7.56138369] 方差为： [2.77329975 0.04632478] 混合系数为： [0.57285263 0.42714737]
第3次迭代
均值为： [2.50242793 7.56021562] 方差为： [1.77752748 0.0463558 ] 混合系数为： [0.54753317 0.45246683]
第4次迭代
均值为： [2.48450825 7.56002825] 方差为： [1.69351331 0.04639821] 混合系数为： [0.54558334 0.45441666]
第5次迭代
均值为： [2.4841361  7.56002054] 方差为： [1.69177929 0.04639883] 混合系数为： [0.54554265 0.45445735]
第6次迭代
均值为： [2.48412949 7.56002039] 方差为： [1.69174851 0.04639884] 混合系数为： [0.54554193 0.45445807]
第7次迭代
均值为： [2.48412937 7.56002039] 方差为： [1.69174796 0.04639884] 混合系数为： [0.54554191 0.45445809]
第8次迭代
均值为： [2.48412937 7.56002039] 方差为： [1.69174795 0.04639884] 混合系数为： [0.54554191 0.45445809]
[0 0 0 0 0 0 1 1 1 1 1]

Process finished with exit code 0
```

可以看出在第 6 次迭代之后，参数已经趋于稳定。前六个分为一类，对应高斯分布为 N(2.484, 1.691)，后五个分为一类，对应高斯分布为 N(7.560, 0.046)