机器学习第五次作业

1、贝叶斯分类器

（1）高斯贝叶斯分类器

1）计算均值和协方差

```python
def mean_value(X):
    X_mean = 0
    for i in range(len(X)):
        X_mean += X[i, :]
    X_mean = X_mean / len(X)
    return X_mean


def covariance(X, X_mean):
    n, p = X.shape
    cov = np.zeros((p, p))
    for i in range(n):
        cov += np.dot((X[i:i + 1, :] - X_mean).T, (X[i:i + 1, :] - X_mean))
    cov = cov / n
    return cov
```

2）计算高斯概率

```python
def gaussian_probability(X, x):
    n, p = X.shape
    X_mean = mean_value(X)
    X_cov = covariance(X, X_mean)
    X_cov_det = np.linalg.det(X_cov)
    X_cov_inv = np.linalg.inv(X_cov)
    one = 1 / ((2 * np.pi) ** (p / 2))
    two = 1 / (X_cov_det ** (1 / 2))
    three = np.exp((-1 / 2) * (x - X_mean) @ X_cov_inv @ (x - X_mean).T)
    X_gaussian = one * two * three
    return X_gaussian
```

3）分类

```python
def decision():
    X_good_gaussian = gaussian_probability(X_good, x)
    X_bad_gaussian = gaussian_probability(X_bad, x)
    good = p_good * X_good_gaussian
    bad = p_good * X_bad_gaussian
    if good >= bad:
        print("密度为{}， 含糖量为{}的瓜，高斯贝叶斯预测为好瓜".format(x[0], x[1]))
    else:
        print("密度为{}， 含糖量为{}的瓜，高斯贝叶斯预测为坏瓜".format(x[0], x[1]))
```

4）结果

```
gaussian_bayes_classifier
D:\Python38\python.exe D:/文件仓库/贝叶斯网/Bayesia
密度为0.5， 含糖量为0.3的瓜， 高斯贝叶斯预测为好瓜
高斯贝叶斯运行时间的一千倍为： 1.482248306274414
```

（2）朴素高斯贝叶斯分类器

1）计算方差

```
def variance(x, mean_value):
    var = 0
    for i in x:
        var += (i - mean_value) ** 2
    var = np.sqrt((var / len(x)))
    return var
```

2）计算条件概率

```
def conditional_probability():
    c = np.sqrt(2 * np.pi)
    p_density_good = ((1 / (c * var_density_good)) *
                      np.exp(-(x[0] - mean_density_good) ** 2 / var_density_good ** 2))
    p_density_bad = ((1 / (c * var_density_bad)) *
                     np.exp(-(x[0] - mean_density_bad) ** 2 / var_density_bad ** 2))
    p_sugar_good = ((1 / (c * var_sugar_good)) *
                    np.exp(-(x[0] - mean_sugar_good) ** 2 / var_sugar_good ** 2))
    p_sugar_bad = ((1 / (c * var_sugar_bad)) *
                   np.exp(-(x[0] - mean_sugar_bad) ** 2 / var_sugar_bad ** 2))
    return p_density_good, p_density_bad, p_sugar_good, p_sugar_bad
```

3）决策

```
def decision():
    p_density_good, p_density_bad, p_sugar_good, p_sugar_bad = conditional_probability()
    good = p_density_good * p_sugar_good * p_good
    bad = p_density_bad * p_sugar_bad * p_bad
    if good >= bad:
        print("密度为{}， 含糖量为{}的瓜，朴素高斯贝叶斯预测为好瓜".format(x[0], x[1]))
    else:
        print("密度为{}， 含糖量为{}的瓜，朴素高斯贝叶斯预测为坏瓜".format(x[0], x[1]))
```

4)结果

```
D:\Python38\python.exe D:/文件仓库/贝叶斯网/Bayesian-netw
密度为0.5， 含糖量为0.3的瓜，朴素高斯贝叶斯预测为好瓜
朴素高斯贝叶斯运行时间的一千倍为： 0.9975433349609375
```

（3）分析

对于高斯贝叶斯分类器和高斯朴素贝叶斯分类器，（密度=0.5， 含糖量=0.3）的瓜都被预测为好瓜。但是，朴素贝叶斯分类器的运行时间更短一些，对于大型数据，其运算时间短的优势就可以得到体现。

2、GMM

（1）证明

2.（1）对目标函数关于μ_i求偏导.

$$\sum_{j=1}^{n} p_{ji} \frac{\partial \ln(\sum_i p(x_j|y_j=i;\theta))}{\partial \mu_i}$$

$$= \sum_{j=1}^{n} p_{ji} \frac{1}{\sum_i p(x_j|y_j=i;\theta)} \frac{\partial \sum_i p(x_j|y_j=i;\theta)}{\partial \mu_i}$$

$$= \sum_{j=1}^{n} p_{ji} \frac{\sum_i p(x_j|y_j=i;\theta)}{\sum_i p(x_j|y_j=i;\theta)} \frac{\partial[-\frac{1}{2}(x_j-\mu_i)^T \Sigma_i^{-1}(x_j-\mu_i)]}{\partial \mu_i}$$

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

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

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

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

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

$\because \Sigma_i^{-1} \neq 0$ $\quad\therefore \sum_{j=1}^{n} p_{ji} x_j = \sum_{j=1}^{n} p_{ji}\mu_i$

$$\therefore \mu_i = \frac{\sum_{j=1}^{n} p_{ji} x_j}{\sum_{j=1}^{n} p_{ji}}$$

（其中$p_{ji}=p(y_i|x_j,\theta^t)$）

(2) 对目标函数关于 Σ_i 求导.

$$\Sigma^{(t+1)} = \arg\max_{\Sigma} \sum_{j=1}^{n} \sum_{i=1}^{k} \left(\log \frac{1}{(2\pi)^{\frac{d}{2}}} - \frac{1}{2} \log |\Sigma_i| - \frac{(x_j-\mu_i)^T \Sigma_i^{-1} (x_j-\mu_i)}{2} \right) p_{ji}$$

$$= \arg\max_{\Sigma} \sum_{i=1}^{k} \sum_{j=1}^{n} \left(\log|\Sigma_i| + (x_j-\mu_i)^T \Sigma_i^{-1} (x_j-\mu_i) \right) p_{ji}$$

$$\frac{\partial \Sigma^{(t+1)}}{\partial \Sigma_i} = \sum_{i=1}^{n} \frac{\partial}{\partial \Sigma_i} \left(\log|\Sigma_i| + (x_j-\mu_i)^T \Sigma_i^{-1} (x_j-\mu_i) \right) p_{ji}$$

$$= \sum_{i=1}^{n} p_{ji} \left(\Sigma_i^{-1} - (x_j-\mu_i)^T (x_j-\mu_i) \Sigma_i^{-2} \right) = 0$$

同时乘以 Σ_i^2 得

$$\sum_{i=1}^{n} p_{ji} \left(\Sigma_i - (x_j-\mu_i)^T (x_j-\mu_i) \right) = 0$$

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

其中 $\frac{\partial |A|}{\partial A} = |A| \cdot A^{-1}$ $\quad \frac{\partial \log|A|}{\partial A} = A^{-1}$

$$(x_j-\mu_i)^T \Sigma_i^{-1} (x_j-\mu_i) = tr\left((x_j-\mu_i)^T \Sigma_i^{-1} (x_j-\mu_i) \right)$$

$$= tr\left(\Sigma_i^{-1} (x_j-\mu_i)^T (x_j-\mu_i) \right)$$

$$\because \frac{\partial\, tr(AB)}{\partial A} = B^T$$

$$\therefore \frac{\partial\, tr\left(\Sigma_i^{-1} (x_j-\mu_i)(x_j-\mu_i)^T\right)}{\partial \Sigma_i} = (x_j-\mu_i)(x_j-\mu_i)^T \frac{\partial \Sigma_i^{-1}}{\partial \Sigma_i}$$

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

(3) 此问题是一个有约束极值问题，约束条件为 $\sum_{i=1}^{k} \alpha_i = 1$

构造Lagrange函数：

$$L(\alpha_1, \alpha_2, \cdots \alpha_k, \lambda) = LL(D) + \lambda\left(\sum_{i=1}^{k} \alpha_i - 1\right)$$

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

$$= \sum_{j=1}^{n} \ln\left(\sum_{i=1}^{k} (\alpha_i p(x_j | y_j = i; \theta)\right) + \lambda\left(\sum_{i=1}^{k} \alpha_i - 1\right)$$

$$\frac{\partial L(\alpha_1, \alpha_2, \cdots \alpha_k, \lambda)}{\partial \alpha_i} = \sum_{j=1}^{n} \frac{p(x_j | y_j = i; \theta)}{\sum_{i=1}^{k} (\alpha_i p(x_j | y_j = i; \theta))} + \lambda = 0$$

两边同时乘以 α_i 得 $\sum_{i=1}^{n} p_{ji} + \lambda \alpha_i = 0$

同时对 $i = 1, 2, 3, \cdots k$ 求和得

$$\sum_{i=1}^{k} \left(\sum_{j=1}^{n} p_{ji} + \lambda \alpha_i\right) = \sum_{i=1}^{k} \sum_{j=1}^{n} p_{ji} + \lambda \sum_{i=1}^{k} \alpha_i$$

$$= n + \lambda \cdot 1 = 0$$

$\therefore \lambda = -n$

$$\therefore \sum_{j=1}^{n} p_{ji} + \lambda \alpha_i = \sum_{j=1}^{n} p_{ji} - n \alpha_i = 0$$

$$\therefore \alpha_i = \frac{1}{n} \sum_{j=1}^{n} p_{ji}$$

（2）上机实验

1）初始化参数

```python
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 = {
        "pi": None,  # 混合系数1*k
        "mu": None,  # 均值k*p
        "cov": None,  # 协方差k*p*p
        "pji": None  # 后验分布n*k
    }

def init_params(self, init_mu):
    pi = np.ones(self.k) / self.k
    mu = init_mu
    cov = np.ones((self.k, self.p, self.p))
    pji = np.zeros((self.n, self.k))
    self.params = {
        "pi": pi,  # 混合系数1*k
        "mu": mu,  # 均值k*p
        "cov": cov,  # 协方差k*p*p
        "pji": pji  # 后验分布n*k
    }
```

2）计算高斯函数

```python
def gaussian_function(self, x_j, mu_k, cov_k):
    one = -((x_j - mu_k) @ np.linalg.inv(cov_k) @ (x_j - mu_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)
```

3）E 步，计算各混合成分的后验概率 pji

```python
def E_step(self, x):
    pi = self.params["pi"]
    mu = self.params["mu"]
    cov = self.params["cov"]
    for j in range(self.n):
        x_j = x[j]
        pji_list = []
        for i in range(self.k):
            pi_k = pi[i]
            mu_k = mu[i]
            cov_k = cov[i]
            pji_list.append(pi_k * self.gaussian_function(x_j, mu_k, cov_k))
        self.params['pji'][j, :] = np.array([v / np.sum(pji_list) for v in pji_list])
```

4）M 步，更新参数{均值， 协方差， 先验概率}

```
def M_step(self, x):
    mu = self.params["mu"]
    pji = self.params["pji"]
    for i in range(self.k):
        mu_k = mu[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['mu'][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 - mu_k).T, (x_j - mu_k)))
        self.params['cov'][i] = np.sum(cov_k_list, axis=0) / np.sum(pji_k_j_list)
        self.params['pi'][i] = np.sum(pji_k_j_list) / self.n
```

5）交替训练，返回聚类结果

```
def fit(self, x, mu, max_iter=10):
    x = np.array(x)
    self.n, self.p = x.shape
    self.init_params(mu)

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

6）结果

```
D:\Python38\python.exe D:/文件仓库/贝叶斯网/Bayesian-network/代码/GMM.py
第0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1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2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3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4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5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6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7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]
```

7）分析

可以看出在最后两次迭代时， 所有参数均为未，算法已经趋于稳定。前六个数据被分为第一类，对应的高斯分布为N—（2.4841， 1.6917)；后五个被分为一类，对应得高斯分布的均值为N—（7.5600， 0.0464)。

3、Markov

```python
import numpy as np

# 转移矩阵
A = np.array([[0.8, 0.2], [0.5, 0.5]])

res = A[1] @ A @ A @ A

print("射中的概率为%.4f， 射不中的概率为%.4f" % (res[0], res[1]))
```

markov

```
D:\Python38\python.exe D:/文件仓库/贝叶斯网,
射中的概率为0.7085， 射不中的概率为0.2915

Process finished with exit code 0
```

因此，第一次没射中，第四次射中的概率为 0.7085