ML003 多元参数估计

杨起

1610299

摘要

在本次实验中， 我完成了基本-中级-高级要求，尝试了参数估计， 非参数估计---核密度估计， KNN算法，调节了参数， 对实验现象做出了分析。最后给出了我得到的对这个问题最佳方案： 基于高斯假设的多元参数估计。

1. 问题描述

每个样例的前 4 列表示特征,第五列表示标签。

- 假设每类数据均满足正态分布,使用参数估计方法估计数据集合的分布参数,并使用似然概率测试规则给出分类性能结果
- 使用平滑核函数方法估计数据集合的分布,并使用似然率测试规则给出分 类性能结果
- 使用最近邻决策分类已知数据集合,给出性能结果

2. 解决方法

2.1 解决思路

1. 多元高斯函数参数估计

- 按照类别计算多元正太分别的参数估计— 均值和协方差矩阵
- 测试分类性能
- 计算错误率

2.2 基本理论

2.2.1 多元参数估计

特征： 观测的属性， 一般来说是相关的， 不然没有必要做多元分析

标签： 离散的标签。 对应多元分类问题

对于一个d维的特征

均值向量(mean vector): μ的每个元素都是X的一列的均值

$$E(x) = \mu = [\mu_1, \ldots, \mu_n,]^T \tag{1}$$

样本均值(sample mean):

它的第i维是X的第i列(第i个特征)的平均值

$$m = \frac{\sum_{t=1}^{N} x^t}{N} \tag{2}$$

其中， 第i维

$$m_i = \frac{\sum_{t=1}^{N} x_i^t}{N} \tag{3}$$

协方差:

$$\sigma_{ij} = Cov(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)] \tag{4}$$

协方差矩阵(covaraiance matrix):

d维就有d个方差， 以及d(d - 1) /2 个协方差。 协方差矩阵记为:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \ldots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \ldots & \\ \ldots & & & \\ \sigma_{d1} & \sigma_{d2} & \ldots & \sigma_d^2 \end{bmatrix} \tag{5}$$

样本协方差(sample covariance)

$$s_i^2 = \frac{\sum_{t=1}^{N} (x_i^t - m_i)^2}{N} \tag{6}$$

$$s_{ij} = \frac{\sum_{t=1}^{N} (x_i^t - m_i)(x_j^t - m_j)}{N} \tag{7}$$

和一维的情况一致， 样本均值是均值向量的无偏估计， 样本协方差是协方差的有偏估计

在X为d维的情况下， 有：

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} exp[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})] \tag{8}$$

2.2.2 KNN算法

理论和实践都和作业1类似，区别只是验证方式改成了K折交叉检验。 简要原理如下

一个样本与特征空间中K个最接近它的点相似。 由此能得出分类相同， 算法主要分成两步：

1. 获取距离P点距离最近的K个点
2. K个点投票决定P的类别

2.2.3 非参数估计 --- 核密度估计

核密度估计（kernel density estimation）是在概率论中用来估计未知的密度函数，属于非参数检验方法之一。

$$\widehat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x-x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x-x_i}{h}\Big),$$

内核的bandwidth是一个自由参数，表现的是核估计的平滑度，它对结果估计有很大的影响。

2.3 算法分析—多元参数估计

由假设：每类数据均满足正态分布

我们可以使用上一节的公式直接计算样本协方差和样本均值

2.3.1 样本协方差矩阵估计

```
# 计算x和y的样本协方差
def cal_cov(x, y):
    m_x = np.mean(x)
    m_y = np.mean(y)
    return np.sum((x - m_x) * (y - m_y)) / len(x)

# 协方差矩阵
def cal_sigma(X_mat):
    col = np.shape(X_mat)[1]
    sigma = np.zeros((col, col))
    for i in range(col):
        for j in range(i, col):
            sigma[i, j] = cal_cov(X_mat[:, i], X_mat[:, j])

    for i in range(col):
        for j in range(0, i):
            sigma[i, j] = sigma[j, i]

    return sigma
```

2.3.2 特征向量X的标签估计

```
# 如公式8
def cal_P(x, sigma, mu):
    d = len(x)
    a0 = (2*np.pi)**(d/2) * np.sqrt(np.linalg.det(sigma))
    a1 = np.dot(np.transpose(x-mu), np.linalg.inv(sigma))
    a2 = np.dot(a1, (x - mu))
    return np.e**(-0.5*a2) / a0
```

2.3.3 测试性能---计算错误率

```
# 对test里的每一个X， 计算三类的概率， 概率最大的， 我们就判断他属于那一类
# 计算错误率
num_test = len(Y_test)
count_acc = 0
for i in range(num_test):
    x = X_test[i]
    y = Y_test[i]
    p_1 = cal_P(x,sigma_X_1,mu_X_1)
    p_2 = cal_P(x,sigma_X_2,mu_X_2)
    p_3 = cal_P(x,sigma_X_3,mu_X_3)
    predict_x = np.argmax([p_1, p_2, p_3]) + 1
    if predict_x == y:
        count_acc += 1
err_rate_list.append(1 - count_acc/num_test)
```

2.4 算法分析—KNN算法

```
# 同作业001 ， 找到K个最近邻点， 预测类别
def find_k_nearest(x_point, x_train, k):
    ":return K args for nearest"
    dis_vec = [distance.euclidean(x_point,  x_train[it]) for it in range(x_train.shape[0])]
    arr = np.argsort(dis_vec)
    return arr[: k]


def knn_predict(y_train, k_point_arr):
    near_class = y_train[k_point_arr]
    return stats.mode(near_class)[0][0]


# 5折交叉检验，性能测试
    for train_index, test_index in kf.split(X):
        '''省略了一些代码，详见hw003_knn.py'''

        for k in range(2, 12):
            # sklearn
            clf = neighbors.KNeighborsClassifier(k)
            clf.fit(X_train, Y_train)
            acc = clf.score(X_test, Y_test)
```

```
            acc_arr.append(acc)

            trueNum = 0
                        # my
            for i in range(X_test.shape[0]):
                point = X_test[i]
                trueValue = Y_test[i]
                # prediction = clf.predict([point])[0]
                arr = find_k_nearest(point, X_train, k)
                prediction = knn_predict(Y_train, arr)
                if trueValue == prediction:
                    trueNum = trueNum + 1
            acc_my = trueNum / X_test.shape[0]
            acc_my_arr.append(acc_my)
```

2.5 算法分析—平滑核函数使用

```
pattern1 = kde.KernelDensity(kernel='gaussian', bandwidth=d).fit(X_1)
'''略去一些代码， 见 kernel.py'''
'''性能测试使用的是5折交叉检验 三次实验测试的代码一致， 见2.3节-测试性能--计算错误率'''
    for i in range(num_test):
        x = X_test[i]
        y = Y_test[i]
        p_1 = pattern1.score_samples(np.mat(x))
        p_2 = pattern2.score_samples(np.mat(x))
        p_3 = pattern3.score_samples(np.mat(x))
        predict_x = np.argmax([p_1, p_2, p_3]) + 1
        if predict_x == y:
            count_acc += 1
            err_rate_list.append(1 - count_acc / num_test)
```

3. 实验分析

3.1 多元参数估计， 实验结果

使用5折交叉检验， 错误率为0.0273。

多元参数估计没有更多的参数可以调整， 在此再分析一下测试性能的假设。

我们是基于最大似然的思路去估计标签的， 即对一组特征X，P(X)最大，就取P(X)对应的假设标签。 （回顾2.2.1节，公式8）但是我们当前做做法是建立在1，2，3三种标签出现的频率相当的基础上的。

我们在公式8中求的并不是P(X),而是

$$P(X|C_i) \tag{9}$$

实际上应该有

$$P(X) = P(X|C_i) \times P(C_i) \tag{10}$$

只不过我们1，2，3标签出现的频率相同。 有

$$P(C_1) = P(C_2) = P(C_3) = 1/3 \tag{11}$$

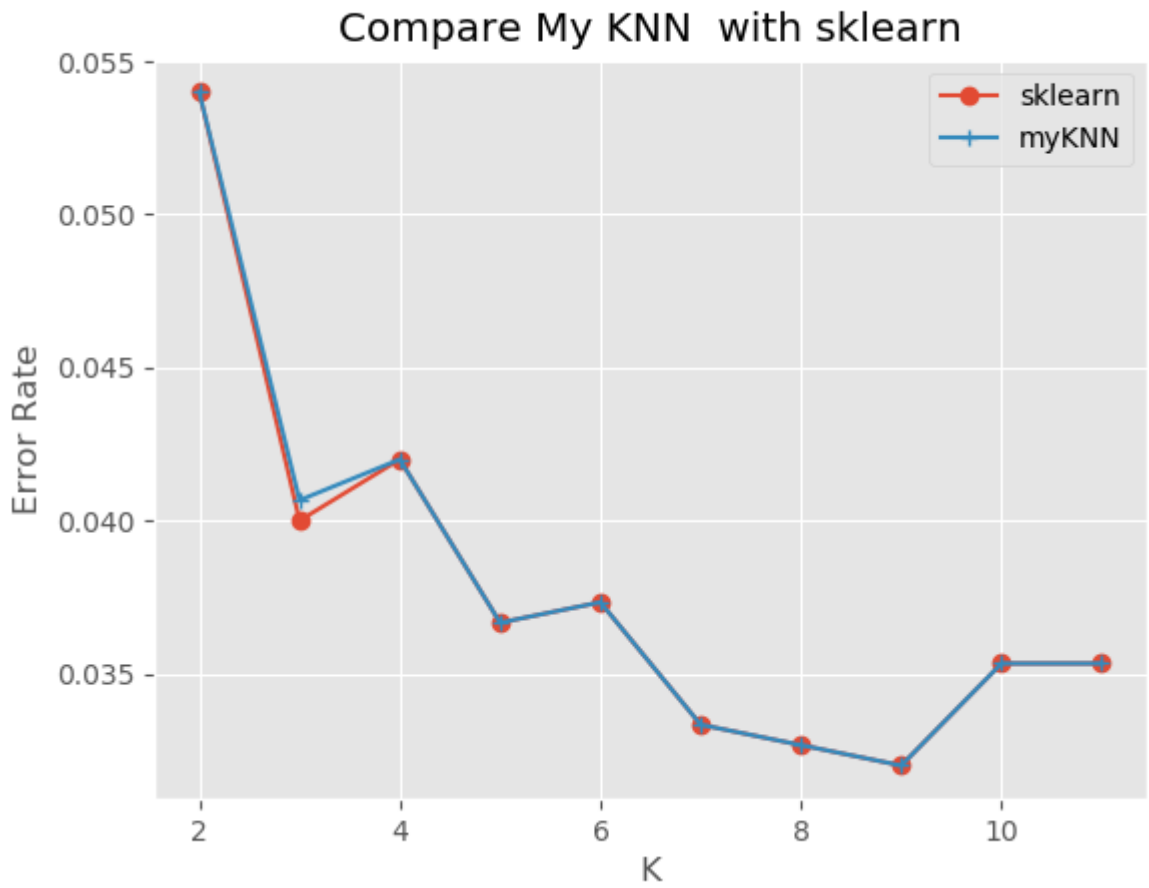
所以才可以直接使用公式8做最大似然估计。

3.2 KNN算法，实验结果和理论分析

我使用了自己在作业1中实现的KNN算法和sklearn中的比较， K的取值从2到12.

可以看出， K=9的时候错误率最小， 可以达到0.0320

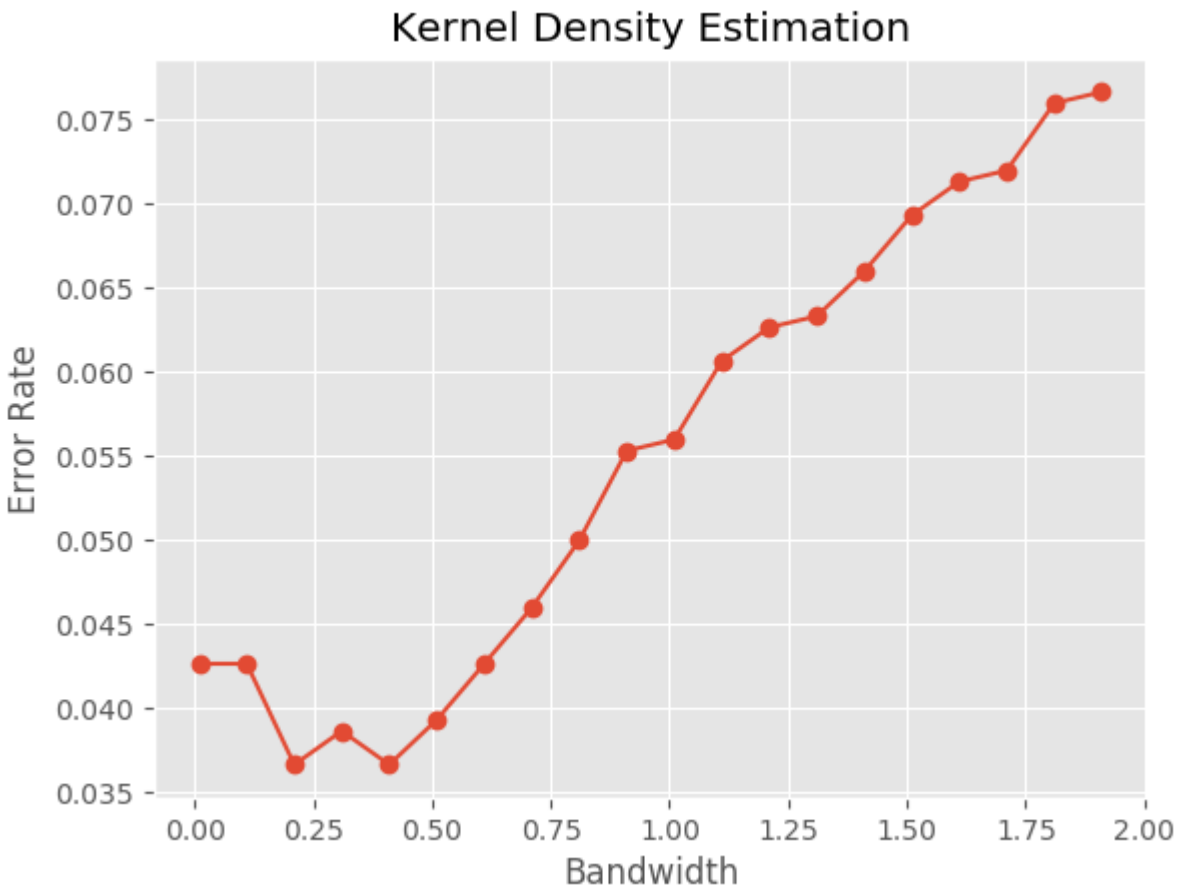
正如我们所知， KNN的K的选择是比较依赖实验和经验的。 在K取值从2到5这段里， 随着K增大， 错误率快速地下降。 之后错误率下降得就比较缓慢。 **这是因为最有价值的几个点基本已经被包含在5个最近邻点中。**



纵向比较， 即使K=9的时候，KNN算法的错误率=0.0320， 比多元参数估计的0.0273稍高。 而且还要搭上更大的计算量(主要是计算特征向量之间的距离，以及寻找K个最近邻点需要排序)， 有正确的假设的情况下， 多元估计在当前的数据规模下展现出了一定的优越性。

3.3 平滑核函数

比较了Bandwidth从0.01到2， **err rate**的变化。可以看出，在bandwidth=0.02和bandwidth=0.04的时候， err rate最小。 为0.0366。bandwidth的合适的大小是比较难从理论上解释的。



纵向比较， 使用高斯核做核密度估计的错误率也是最高的， 以下是三种方法的错误率

多元参数估计（高斯分布假设）	KNN算法	核密度估计（高斯核）
0.0273	0.0320	0.0366

多元参数估计还是稍微优秀一点。