实验三：参数估计与最近邻决策

杨扬　计算机科学与技术　1611132

摘　要

本次实验是对参数估计方法的学习及实现，所使用的数据集是一个包含 4 维特征，3 类标签的数据集。本文档针对于此问题，分析数据集本身的特性，介绍了各种参数估计方法和最近邻决策的原理，结合于实验数据集本身设计了算法，并基于源程序对实验的设计进行了解析，最后对实验的结果进行了分析。简单来说，本文档在对实验所有要求均做出实现的同时，也成功地进行了多方面的拓展，以极高的质量完成了此次实验。

关键词：参数估计、平滑核函数、最近邻决策

目录

1 问题描述

参数估计（Parameter Estimation），是统计推断的一种。根据从总体中抽取的随机样本来估计总体分布中未知参数的过程。从估计形式看，区分为点估计与区间估计：从构造估计量的方法讲，有矩法估计、最小二乘估计、似然估计、贝叶斯估计等。要处理两个问题：（1）求出未知参数的估计量；（2）在一定信度（可靠程度）下指出所求的估计量的精度。信度一般用概率表示，如可信程度为 95%；精度用估计量与被估参数（或待估参数）之间的接近程度或误差来度量。

参数估计是统计学习领域中一系列估计方法的统称，在本学期的机器学习课程中，我们主要学习了最大似然估计的理论推导及应用实例，而本次实验中所采用的参数估计方法也正是最大似然估计。

本次实验的数据集文件是 *HWData3.csv*。具体来说，数据的存储形式如图 1 所示，文件中包含 150 行、5 列的浮点数字，列与列之间以逗号分割。其中各列数据并无实际意义，其中前 4 列为特征，最后 1 列为标签。150 行数据中根据标签的不同分为 3 组，每组各 50 行数据。此次实验需要我们基于前 4 列的特征对数据的概率分布进行参数的估计或进行最近邻决策分类，然后根据测试数据的特征计算出预测的标签，与最后 1 列的实际标签进行比较，并计算出精确率。

图 1: 数据存储形式

2 参数估计

本次实验的问题是对我们课上所学习的参数估计理论方法的实际应用。针对于此问题，我们需要基于实验要求中各种假设条件进行求解。

2.1 多元正态分布

首先，实验基本要求中假设 3 类数据均满足正太分布。简单回顾一下概率论中的最简单的一元正态分布，其概率密度的函数为如下形式：

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

公式中，σ 为样本的标准差，μ 为样本的均值。本次实验中 3 类数据样本均含有 4 维特征，因此，我们基于一元正态分布进行拓展，引入多元正态分布，首先，将特征对应的各个维度的数据点描述为：

$$x = [x_1, x_2, \cdots, x_d]^T$$

在本次实验中，d 的值即为 4。然后，各个维度数据的样本均值和方差定义为：

$$\mu_1, \mu_2, \cdots, \mu_d, \sigma_1, \sigma_2, \cdots, \sigma_d$$

涉及到多维形式的正态分布，还需考虑各个维度间的相关性，在统计学中，通常把这种相关性以协方差和相关系数这两个概念来进行刻画，其中应用更为广泛的是协方差这一概念。

由于课上和教材中对多元正态分布都进行了详细的介绍，因此在本文档中我们略去多元正态分布的中间推导过程。总之，将多维数据进行向量化后代入一元正态分布概率密度函数后，经过一系列的推导，可得到多元正态分布的概率密度函数为如下形式：

$$f(x) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-\mu)^T \Sigma^{-1} (X-\mu)}$$

其中，Σ 代表协方差矩阵，其第 i 行的第 j 个元素即表示第 i 个变量与第 j 个变量的协方差，而 $|\Sigma|$ 即表示对 Σ 进行行列式运算。

在得到多元正态分布的概率密度函数之后，我们即可应用课上所学到的最大似然估计的思想，来对函数中出现的各参数进行估计了。

由于正态分布中存在自然底数 e 的指数形式，我们通常采用求对数的方式来简化求解似然函数的过程，即：

$$\ln L(\mu, \Sigma) = \sum_{i=1}^{n} \ln f\left(x_i; \mu, \Sigma\right)$$

对数似然函数的函数性质仍与似然函数的性质一样，最大似然估计方法中为使此函数的的值达到最大，我们需要分别对 μ 和 Σ 分别进行求导，经过一系列化简后，我们可以得到各维 μ 和方差 σ 的估计值计算公式为：

$$\mu = \overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

$$\sigma = \frac{1}{n} \sum_{i=1}^{n} \left(x_i - \overline{x}\right)^2$$

其中 n 的值即为样本的数量。此时我们仍需解决协方差矩阵 Σ 的计算，我们已经得到各维样本的方差值了，根据概率论中的协方差计算公式即可计算出不同维度数据间的协方差，从而构造出协方差矩阵。

至此，我们便完成了对本实验中多元正态分布概率密度函数中各参数的估计了。

2.2 核密度估计

本次实验的另一基本内容是要求我们使用平滑核函数来进行参数估计，这便涉及到核密度估计的内容。简单来说，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，核密度估计方法不利用有关数据分布的先验知识，对数据分布不附加任何假定，是一种从数据样本本身出发研究数据分布特征的方法，因而，在统计学理论和应用领域均受到高度的重视。

由于这部分知识属于非参数估计的内容，因此这里我们不对其原理部分进行分析，实验中我们采用调用 sklearn 中的相关方法进行实现即可。

3 最近邻决策

本次实验的另一提高要求是最近邻决策分类算法的应用，这部分内容我们在实验一中已经进行过详细的实现了，这里我们对当时所设计的算法进行一个简单

的回顾。

基于最近邻决策的思想，我们将本实验的问题归为一个 3 分类的问题，因此算法的运行步骤如下：

- 计算待分类点与已知类别的点之间的欧氏距离
- 按照距离递增次序排序
- 选取与待分类点距离最小的 k 个点
- 确定前 k 个点所在类别的出现次数
- 返回前 k 个点出现次数最高的类别作为待分类点的预测分类

至此，本次实验中所有算法的理论基础便介绍完毕。

4 实验及源程序分析

4.1 实验环境

本次实验在 Ubuntu 18.04 下进行，实验所使用的编程语言为 Python 3.6，IDE 为 Pycharm 2018.3.2，主要使用了 numpy、pandas、matplotlib、sklearn 等 Python 包。

4.2 正态分布下参数估计源程序分析

由于此次实验的内容较少，实验数据规模也不算太大，所以我们在一个 py 文件里即可完成本次实验，具体来说，程序的步骤如下所示：

首先我们在 Python 中导入所需要用到的 Python 包：

```
import pandas as pd
import numpy as np
from sklearn.neighbors import kde
import matplotlib.pyplot as plt
```

然后我们需要导入相应的实验数据，*HWData3.csv* 文件中数据的分割方式为逗号，是一个正常的 *csv* 文件，且不含文件头，因此我们采用 *pandas* 包来读入数据：

```
def read_data(path):
    data = np.array(pd.read_csv(data_path, header=None))
    label = data[:, -1]
    data = data[:, :-1]
    return data, label
```

读入数据后，由于本次实验没有提供专门的测试集，所以我们需要进行训练集与验证集的划分，与前两次实验一样，我们仍旧采用多折交叉验证的方式。需注意的是，为保证验证集中各类数据出现概率均等，我们需要先把 3 类数据分开后再分别划分训练集和测试集：

```
def train_valid_split(data, label, fold, idx):
    sample1 = data[:50]
    label1 = label[:50]
    sample2 = data[50:100]
    label2 = label[50:100]
    sample3 = data[100:]
    label3 = label[100:]
    train_idx = []
    valid_sample = []
    valid_label = []
    for i in range(sample1.shape[0]):
        if i%fold == idx:
            valid_sample.append(sample1[i, :])
            valid_sample.append(sample2[i, :])
            valid_sample.append(sample3[i, :])
            valid_label.append(label1[i])
            valid_label.append(label2[i])
            valid_label.append(label3[i])
        else:
            train_idx.append(i)
    train_sample1 = sample1[train_idx, :]
    train_sample2 = sample2[train_idx, :]
    train_sample3 = sample3[train_idx, :]
    train_label1 = label1[train_idx]
    train_label2 = label2[train_idx]
    train_label3 = label3[train_idx]
    valid_sample = np.array(valid_sample)
    valid_label = np.array(valid_label)
    return train_sample1, train_sample2, train_sample3, \
```

然后我们便可开始参数估计的核心内容了，首先我们进行正态分布条件下的参数估计。根据前文分析得到的参数计算公式，定义参数计算函数为：

```python
def cal_para(sample):
    dim = sample.shape[1]
    mean = np.mean(sample, 0)
    sigma = np.cov(sample.T)
    det_sigma = np.linalg.det(sigma)
    return dim, mean, sigma, det_sigma
```

得到概率密度函数中的各参数后，我们构造概率密度函数公式，定义概率计算函数为：

```python
def cal_prob(vec, sample):
    dim, mean, sigma, det_sigma = cal_para(sample)
    prob = (1 / (((2*np.pi) ** (dim/2)) * np.sqrt(det_sigma))) * \
           np.exp((-(1/2) * (vec - mean)).dot(np.mat(sigma).I).dot(vec - mean))
    return prob
```

得到概率值后，我们基于实验要求中的似然概率测试规则，比较验证集中各条样本对应的 3 类分布的概率大小，将概率值最大的 1 类作为验证样本的预测标签：

```python
def pred_label(valid, train_sample1, train_sample2, train_sample3, method):
    label = []
    for i in range(valid.shape[0]):
        if method is 'likelihood':
            prob1 = cal_prob(valid[i, :], train_sample1)
            prob2 = cal_prob(valid[i, :], train_sample2)
            prob3 = cal_prob(valid[i, :], train_sample3)
        else:
            prob1 = cal_prob_smooth(valid[i, :], train_sample1)
            prob2 = cal_prob_smooth(valid[i, :], train_sample2)
            prob3 = cal_prob_smooth(valid[i, :], train_sample3)
        if max([prob1, prob2, prob3]) == prob1:
            label.append(1)
        elif max([prob1, prob2, prob3]) == prob2:
            label.append(2)
        else:
            label.append(3)
    return np.array(label)
```

为方便后续调用其他算法，我们定义一个交叉验证的借口函数，在此函数中完成交叉验证的内容，并计算出 10 折交叉验证情况下各算法的平均误差

```
def cross_valid(data, label, fold, method, k=3):
    acc = []
    for i in range(fold):
        train_sample1, train_sample2, train_sample3, \
        train_label1, train_label2, train_label3, \
        valid_sample, valid_label = train_valid_split(data, label, fold, i)
        if method is 'likelihood' or method is 'kernel':
            pred = pred_label(valid_sample, train_sample1, train_sample2,
                              train_sample3, method)
        elif method is 'knn':
            train_sample = np.concatenate((train_sample1, train_sample2, train_sample3),
                                          axis=0)
            train_label = np.concatenate((train_label1, train_label2, train_label3),
                                         axis=0)
            pred = pred_label_knn(valid_sample, train_sample, train_label, k)
        elif method is 'knn_weighted':
            train_sample = np.concatenate((train_sample1, train_sample2, train_sample3),
                                          axis=0)
            train_label = np.concatenate((train_label1, train_label2, train_label3),
                                         axis=0)
            pred = pred_label_knn_weighted(valid_sample, train_sample, train_label, k)
        else: return None
        correct = 0
        for j in range(valid_label.shape[0]):
            if pred[j] == valid_label[j]:
                correct += 1
        print(i, 'acc', method, correct/valid_label.shape[0])
        acc.append(correct/valid_label.shape[0])
    return np.mean(acc)
```

至此，我们便完成了多元正态分布情况下的参数估计。

4.3 核密度估计源程序分析

核密度估计的内容我们依靠 sklearn 中的方法来进行实现，因此代码较为简单，修改概率值的计算函数即可，实验中我们选用的平滑核函数为高斯核函数：

```
def cal_prob_smooth(vec, sample):
    model = kde.KernelDensity(kernel='gaussian', bandwidth=0.2).fit(sample)
    prob = np.exp(model.score_samples(vec.reshape(1, -1)))
    return prob
```

4.4 最近邻决策分类源程序分析

最近邻决策思想的首要内容是各数据点间的距离度量，与实验一时一样，这里我们仍采用欧式距离来实现：

```python
def cal_distance(vec1, vec2):
    return np.sqrt(np.sum(np.square(vec1 - vec2)))  # 计算欧式距离
```

然后将验证集中的数据逐条输入，计算与训练集中所有数据点的距离，选取最近的 k 个：

```python
def get_knn(vec, data, k):
    distances = []  # 存储目标点与训练样本中的点的距离
    for i in range(data.shape[0]):  # 与训练样本中的每一个点都需要计算距离
        distance = cal_distance(vec, data[i])
        distances.append(distance)
    distances = np.array(distances)
    idx = np.argsort(distances)[0:k]  # 取出前k个距离最小的点对应的索引
    distances = distances[idx]  # 取出k个点的距离，其余点可以省略
    return distances, idx
```

接下来是决策的过程，我们采用投票的机制来实现对验证数据点标签的预测：

```python
def pred_label_knn(test, train, label, k):
    preds = []  # 存储对待分类点预测的标签
    for i in range(test.shape[0]):
        vec = test[i]
        vote = np.zeros(3)  # 投票数组，用于判定k个点中哪类点最多
        distances, idx = get_knn(vec, train, k)  # 计算出k个近邻
        knn_label = label[idx]  # 在训练样本标签中取出k个近邻的标签
        for j in range(knn_label.shape[0]):
            if knn_label[j] == 1:
                vote[0] += 1  # 进行投票
            elif knn_label[j] == 2:
                vote[1] += 1
            else:
                vote[2] += 1
        pred = vote.argmax() + 1
        preds.append(pred)
    return np.array(preds)
```

与实验一时一样，我们也可以采用基于距离加权的方式来优化投票机制：

```python
def pred_label_knn_weighted(test, train, label, k=3):
    preds = []  # 存储对待分类点预测的标签
    for i in range(test.shape[0]):
        vec = test[i]
        vote = np.zeros(3)  # 投票数组，用于判定k个点中哪类点最多
        distances, idx = get_knn(vec, train, k)  # 计算出k个近邻
        knn_label = label[idx]  # 在训练样本标签中取出k个近邻的标签
        for j in range(knn_label.shape[0]):
            if distances[j] == 0:
                continue
            else:
                if knn_label[j] == 1:
                    vote[0] += 1/distances[j]  # 基于距离加权的投票
                elif knn_label[j] == 2:
                    vote[1] += 1/distances[j]
                else:
                    vote[2] += 1/distances[j]
        pred = vote.argmax() + 1
        preds.append(pred)
    return np.array(preds)
```

为选取一个合适的 k 值，我们定义一个选择 k 的函数，将 k 的测试范围定义为 1-20，并记录下 k 取其中各值时的分类精确率：

```python
def select_k(data, label, fold, method):
    k = [x for x in range(1, 21)]
    accs = []
    for i in k:
        acc = cross_valid(data, label, fold, method, i)
        accs.append(acc)
    best_k = np.argmax(accs) + 1
    return best_k, k, accs
```

至此，我们便完成了本次实验的源程序分析部分。

5 实验结果分析

实验中验证方式我设置为 10 折交叉验证，得到各种算法的平均预测准确率为如下：

	正态分布参数估计	核密度估计
Mean Accuracy	0.98	0.97
	最近邻决策	基于距离加权的最近邻决策
Mean Accuracy	0.98	0.98

表 1: 4 种算法 10 折交叉验证情况下的平均预测准确率

从上表可以看出，由于数据集本身比较符合高斯分布的特点，我们在假设其符合高斯分布时的预测准确率便已经达到了一个相当不错的程度，几乎对所有的验证数据都做出了正确的预测，而其错误我猜想可能是有部分的离群点。同样的，核密度估计也取得了相当不错的表现，与最大似然估计的性能差异极小，我猜想可能也是数据集本身的分布特点所导致的。

此外，对于两种最近邻决策分类算法，其平均准确率均达到了与假设正态分布情况下最大似然估计方法相同的程度，证明数据集在高维的几何空间中的确明显地划分成了 3 个部分，因此在最近邻决策算法的运行中才能被正确地划分开。另一方面，实验中 k 值对最终准确率的影响并不算太大，我在编写程序时记录了两种最近邻决策算法在 k 值为 1-20 范围内的平均准确率，并绘制其变化折线图如下：

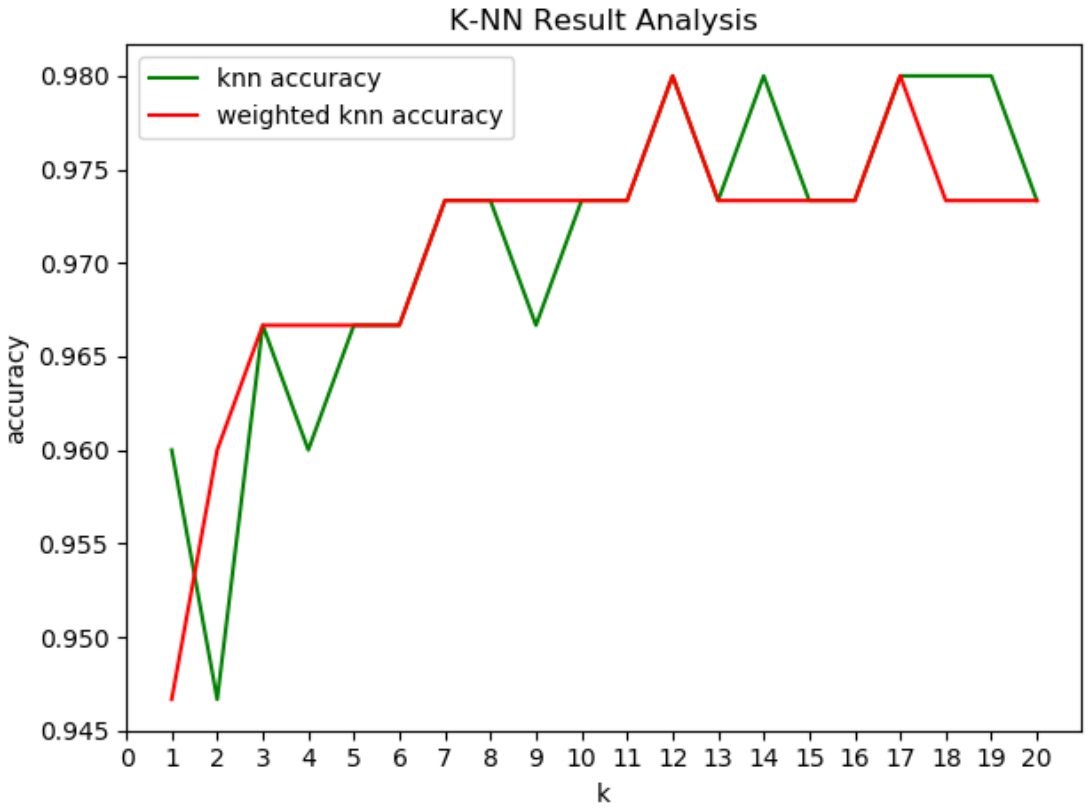

图 2: 梯度下降与随机梯度下降的 loss 值变化曲线图

由图 2 可以看出，分类准确率受 k 值的影响并不算太大，而基于距离加权的最近邻算法在准确率上也没有体现出更强的优越性，我猜想可能也是数据集的分布特点所导致，两种算法在实验中也很难取得更高的分类准确率，但目前的精度也已经相当不错了。

6 总结

参数估计是统计学习中极其重要的思想与方法，在其推导和代码实现中更能领略数学的无穷魅力。总的来说此次实验我收获颇丰，对机器学习也有了更深层次的理解。

参考文献

[1] Thomas M. Mitchell. 1997. Machine Learning (1 ed.). McGraw-Hill, Inc., New York, NY, USA.