实验三：参数估计、非参数估计

姓名：孙铭

学号：1711377

专业：计算机科学与技术

日期：2020年4月4日

摘要

本次实验要求实现基于数据集`HWData3.csv`的参数估计和非参数估计。实验指导中关于本次实验的三类要求已全部实现。本次实验代码及作业报告均由个人独立完成。具体实现功能参下。

> *实验基本要求：*
>
> > *（1）假设每类数据集均满足正态分布，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。*
> >
> > *（2）使用核函数方法估计数据集的分布，并使用似然率测试规则给出了分类性能结果。此外，为了选取合适的带宽参数值，我设计了实验，作出了不同核函数中带宽参数同分类准确度之间的关系图。并求出在不同核下，分类最大正确率及对应的最优带宽参数。*
>
> *实验中级要求：*
>
> > *使用最近邻决策分类已知数据集合，给出性能结果，并与参数估计方法和核函数方法的结果相比较。同时做出不同分类方法准确率结果比较水平直方图。*
>
> *实验高级要求：*
>
> > *对于一维情况，进行k-近邻概率密度估计，对数据集的第1类中的特征x_1，用程序画出$k=1,3,5$时的概率密度估计结果。*

以上是本次实验实现的功能，接下来将展开论述。

环境配置

本次作业使用语言版本为`Python 3.6.5 64bit`，处理器为`Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80GHz`，项目工程文件如下。

```
|   data
|   | HWData3.csv          # 数据集文件
|   pic                    # 保存实验结果图的文件
|   parametric.py          # 参数估计方法实现
|   non_parametric.py      # 核函数方法实现
|   nn_decision.py         # 最近邻决策实现
|   knn_estimate.py        # k-近邻概率密度估计
|   draw.py                # 直方图绘制
```

项目实现过程中用到的包及作用如下。

```
import csv                                  # 解析csv文件
import math                                 # 常用数学函数
import numpy as np                          # numpy数据结构
import matplotlib.pyplot as plt             # 绘制图像
from sklearn.model_selection import KFold         # 交叉验证
from sklearn.neighbors.kde import KernelDensity   # 核函数
from sklearn.preprocessing import StandardScaler # 数据标准化
```

以上是环境配置部分介绍。

数据处理

本次实验所使用的数据集为`HWData3.csv`，本数据集共包含150条数据。数据集变量组成为4个输入变量（特征变量x_1, x_2, x_3, x_4）和1个输出变量（标签变量y），各变量无特殊含义。

处理数据的代码如下，作用是将`.csv`文件中的数据导出，并将特征变量和标签变量保存为矩阵形式返回。

```
def load_data(filename):
    with open(filename, encoding='utf-8') as f:
        csv_reader = csv.reader(f)
        _data = [line for line in csv_reader]
    data = np.array(_data[1: ], dtype=float)
    ss = StandardScaler()
    X = ss.fit_transform(data[:, :-1])
    Y = data[:, -1]
    return X, Y
```

代码中使用了机器学习包`sklearn`中的`StandardScaler()`函数和`fit_transform()`函数。前者的作用是标准化数据，以保证每个维度的特征数据方差为1，均值为0；后者的作用是先对数据进行拟合，再标准化，使用该函数可以在保留样本特征的前提下实现数据标准化。

此外，在实验高级要求中，由于给出的要求是对数据集的第一类特征中的x_1进行概率密度估计，因此在实验处理数据中需要先对数据集进行切分，再进行数据标准化，实现代码如下。

```
data = np.array(_data[1: ], dtype=float)
Y = data[:, -1]
data_1 = data[np.where(Y == 1)]
ss = StandardScaler()
X = ss.fit_transform(data_1[:, :-1])
```

以上是本次实验涉及到的数据处理部分内容。

实验基本要求

（1）参数估计方法

计算高斯分布参数

根据作业要求，假定每类数据集均满足正态分布。由于特征数量总共有4种，故数据集符合多元高斯分布。对于多元高斯分布，概率密度公式为：

$$N(x|\mu,\Sigma)=\frac{1}{(2\pi)^{\frac{D}{2}}}\frac{1}{|\Sigma|^{\frac{1}{2}}}exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}\}$$

其中，μ为样本均值，Σ为协方差矩阵。

对应代码如下。其中，`np.linalg.det()`函数用来求矩阵的逆矩阵，`np.linalg.inv()`函数用来求矩阵对应的行列式的值。

```
def calculate_prob(ave, cov, X):
    sub = (X - ave).reshape(X.shape[0], 1)
    coef_1 = 1 / math.pow(2 * math.pi, X.shape[0] / 2)
    coef_2 = 1 / math.sqrt(np.linalg.det(cov))
    coef_3 = math.exp(-1/2 * np.dot(sub.T,
np.dot(np.linalg.inv(cov), sub)))
    return coef_1 * coef_2 * coef_3
```

在对于多元高斯分布的参数估计方法中，参数的最大似然估计就是样本均值的矢量和样本的协方差矩阵。

样本均值矢量求解公式为：

$$\hat{\mu}=\frac{1}{N}\sum_{k=1}^{N}x^k$$

样本协方差矩阵求解公式为：

$$\hat{\Sigma}=\frac{1}{N}\sum_{k=1}^{N}(x^k-\hat{\mu})(x^k-\hat{\mu})^T$$

对应参数求解部分代码如下。

```
# 估计参数
def estimated_param(X):
    ave = np.average(X, axis=0) # 均值
    sub = X - ave
    cov = np.empty((X.shape[1], X.shape[1])) # 协方差
    for i in range(X.shape[1]):
        for j in range(i + 1):
            cov[i, j] = cov[j, i] = np.matmul(sub[:, i],
sub[:, j]) / X.shape[0]
    return ave, cov
```

代码比较简单，基本原理在前文公式给出，这里不再赘述。

测试分类性能、计算准确率

对于分类性能的测试，目的是计算每个样例从哪个高斯分布采样的可能性大，即将样例分别代入三个分布的均值方差，概率最高的即为该样例的标签。

这里采用十折交叉验证的方式对分类性能进行测试，调用的是机器学习包`sklearn`中的`KFold()`函数，该函数中`n_splits`参数代表交叉验证的折数，`shuffle`参数值为`True`时，数据集随机切分，为`False`时代表数据集不随机切分。代码如下。

```
def cross_validate_param(X, Y):
    kf = KFold(n_splits=10, shuffle=False)
    acc_res = 0 # 计算平均正确率
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        params = []
        for i in range(1, 4):
            index = np.where(Y_train == i)
            params.append(estimated_param(X_train[index]))
        acc = 0
        for i in range(len(X_test)):
            prob = [calculate_prob(ave, cov, X_test[i])
for ave, cov in params]
            y_hat = prob.index(max(prob)) + 1
            if y_hat == Y_test[i]:
                acc += 1
        acc_res += acc / len(X_test)
    return acc_res / 10
```

关于代码注释的细节请参见`parametric.py`中的源码，实现原理较为简单，不再赘述。

对分类结果进行计算，执行测试部分代码如下。

```
if __name__ == "__main__":
    filename = "data/HWData3.csv"
    X, Y = load_data(filename)
    res = cross_validate_param(X, Y)
    print("多元高斯分布参数估计分类准确率为：%.2f" % (res *
100), '%')
```

测试结果如下。

```
[Running] python -u "c:\Users\15696\Desktop\课程\ML\实验三\code\parametric.py"
多元高斯分布参数估计分类准确率为：97.33 %

[Done] exited with code=0 in 1.688 seconds
```

可以看到，多元高斯分布参数估计分类结果的准确率为97.33%。

（2）核函数方法

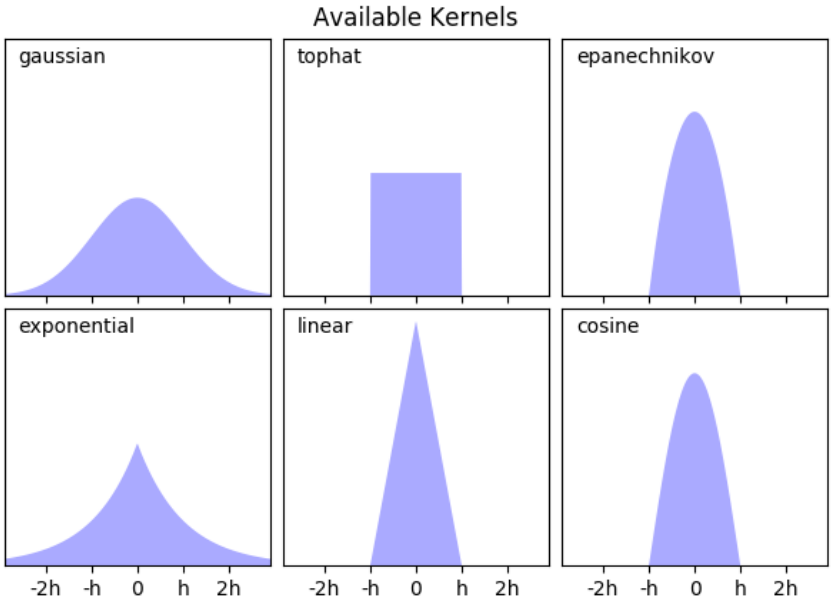
根据实验指导PPT中给出的内容，非参数概率密度估计的核心思路是，一个向量x落在区域R中的概率P为：$P = \int_R p(x)dx$，因此，可以通过统计概率P来估计概率密度函数$P(x)$。简单而言，就是在纸上画个圆，朝纸面均匀撒一把豆子，落在圆内的概率就是落在圆内豆子数量占纸上所有豆子数量的百分比。而核函数说白了就是纸上画出的图形。

非参数密度估计的一般形式如下。

$$p(x) = \frac{k}{NV} \quad where \begin{cases} V\ is\ the\ volume\ surrounding\ x \\ N\ is\ the\ total\ number\ of\ examples \\ k\ is\ the\ number\ of\ examples\ inside\ V \end{cases}$$

而核函数密度估计即固定V值，使用样本确定k值。核函数密度估计又称Parzen窗口估计，通常平滑核函数是径向对称且具有单峰的概率密度函数。

`sklearn.neighbors.KernelDensity`实现了scikit-learn中的核密度估计，它使用Ball Tree或KD Tree来进行高效查询。该方法实现的核密度估计可以用在任意维度上。此外，在`sklearn.neighbors.KernelDensity`中，实现了一些常见形状的核，如下图所示。



在本次实验中，用到的核为`['gaussian', 'tophat', 'epanechnikov', 'exponential', 'linear']`五种。其中，各个核的函数详细介绍请参见skearn官方中文文档。

关于核函数密度估计分类测试的原理，与前文参数估计的原理一致，即计算每个样例从哪个核函数采样的可能性大，即将样例分别代入三类标签数据对应的核函数，概率最高的即为该样例的标签。实现代码如下。

```
def cross_validate_nonparam(X, Y, kernel, bandwidth):
    kf = KFold(n_splits=10, shuffle=False)
    acc_res = 0 # 计算平均正确率
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
```

```
        Y_train, Y_test = Y[train_index], Y[test_index]
        _kd = []
        for i in range(1, 4):
            index = np.where(Y_train == i)
            _kd.append(KernelDensity(kernel=kernel,
bandwidth=bandwidth).fit(X_train[index]))
        acc = 0
        prob = np.array([item.score_samples(X_test) for
item in _kd]).T
        for i in range(len(X_test)):
            y_hat = prob[i].tolist().index(max(prob[i])) +
1
            if y_hat == Y_test[i]:
                acc += 1
        acc_res += acc / len(X_test)
    return acc_res / 10
```

在数学上，核是由带宽参数h控制的正值函数$K(x|h)$，给定核的形状后，在一组点$x_i, i = 1, 2...N$内的y点处的密度估计由下式给出：

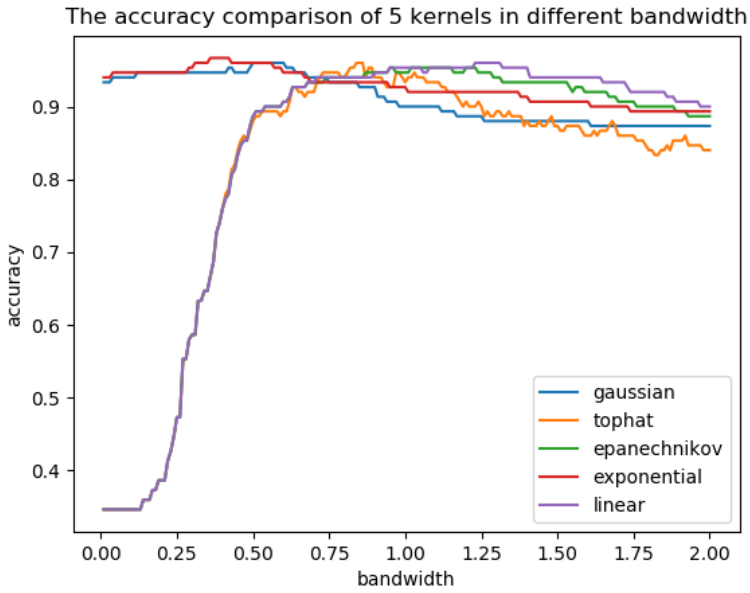
$$P_K(y) = \sum_{i=1}^{N} K(\frac{y - x_i}{h})$$

在核函数密度估计中，带宽参数（平滑参数）h决定了函数的“膨胀”程度，用来平衡结果中的偏差和方差的值。平滑参数的选择是十分重要的，较大取值将产生过渡平滑的密度估计，模糊了数据的空间结构；而较小取值将产生又长又尖的密度估计，解释比较困难。

根据实验要求，采用交叉验证得h值，h取值范围$(0.01, 2)$，通过取交叉验证的均值求得最优的h值，在利用最优的h验证分类结果（同参数估计）。前文给出了交叉验证的代码，这里再给出求解最优带宽参数的函数代码如下。

```
def fit_bandwidth(X, Y, kernel):
    acc = []
    _bandwidth = np.arange(0.01, 2.01, 0.01)
    for bw in _bandwidth:
        acc.append(cross_validate_nonparam(X, Y, kernel,
bw))
    acc = np.array(acc)
    bands_maxacc = _bandwidth[np.where(acc == max(acc))]
    return _bandwidth, acc, bands_maxacc
```

除此之外，为了进一步反应出对于不同核函数而言，带宽参数对分类准确率的影响情况，我设计了一组对比实验，实验代码请参见`non_parametric.py`文件，这里不再赘述。下图是实验结果图。



因此，对于不同核函数，在最优带宽参数下，分类准确率情况如下图所示。

```
[Running] python -u "c:\Users\15696\Desktop\课程\ML\实验三\code\non_parametric.py"
kernel=gaussian, 最大正确率96.0000 %
正确率最大时的bandwidth: [0.5  0.51 0.52 0.53 0.54 0.55 0.56 0.57 0.58 0.59 0.6 ]
kernel=tophat, 最大正确率96.0000 %
正确率最大时的bandwidth: [0.84 0.85 0.86]
kernel=epanechnikov, 最大正确率95.3333 %
正确率最大时的bandwidth: [0.95 0.96 1.02 1.03 1.04 1.05 1.06 1.07 1.08 1.09 1.1  1.11 1.12 1.13
 1.14 1.15 1.2  1.21 1.22 1.23]
kernel=exponential, 最大正确率96.6667 %
正确率最大时的bandwidth: [0.36 0.37 0.38 0.39 0.4  0.41 0.42]
kernel=linear, 最大正确率96.0000 %
正确率最大时的bandwidth: [1.23 1.24 1.25 1.26 1.27 1.28 1.29 1.3  1.31]

[Done] exited with code=0 in 12.681 seconds
```

从图示运行结果中可以很明显地看到在不同核下，最大分类正确率及相应的带宽参数大小。

以上是实验基本要求部分。

实验中级要求

接下来讲述实验中级要求部分，即利用最近邻决策分类已知数据集合，给出性能结果，并同参数估计方法和核函数密度估计方法的结果作对比。

根据实验指导，最近邻决策规则规则思路如下。

> *已知：N个样本组成的训练样本的集合，其中ω_i类包含N_i个样本。*
>
> *问题：对未知样本X_μ的分类决策。*
>
> *求解：*
>
> > *（1）以未知样本X_μ为中心构造一个体积为V的超球。*
> >
> > *（2）令球内共包含k个样本，其中包含k_i个ω_i类的样本。*
> >
> > *（3）（省略证明过程）后验概率为$P(\omega_i|x)=\frac{k_i}{k}$*

因此，关于最近邻决策规则（贝叶斯决策规则）有如下伪代码。

$$decide \quad \omega_i \ \ if \ \ g_i(x) > g_j(x) \ \ \forall j \neq i$$
$$where \quad g_i(x) = P(\omega_i|x) = \frac{k_i}{k}$$

上述最近邻决策过程实现代码如下。

```
# 计算欧氏距离
def euclidean_dist(data_1, data_2):
    return np.linalg.norm(data_1 - data_2, ord=2, axis=1)

# 最近邻决策规则
def nn_Decision(X_train, Y_train, X_test):
    y_hat = []
    for item in X_test:
        ed = euclidean_dist(X_train, item)
        y_hat.append(Y_train[ed.tolist().index(min(ed))])
    return np.array(y_hat, dtype=int)
```

利用十折交叉验证对最近邻决策分类结果进行测试代码如下。

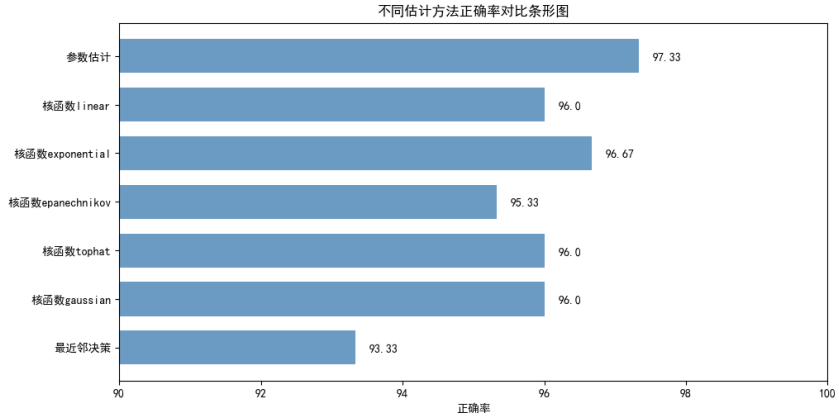
```
def cross_validate_nn(X, Y):
    kf = KFold(n_splits=10, shuffle=False)
    acc_res = 0
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        y_hat = nn_Decision(X_train, Y_train, X_test)
        acc = 0
        for i in range(len(X_test)):
            if y_hat[i] == Y_test[i]:
                acc += 1
        acc_res += acc / len(X_test)
    return acc_res / 10
```

使用最近邻决策规则分类准确率结果如下。

```
[Running] python -u "c:\Users\15696\Desktop\课程\ML\实验三\code\nn_decision.py"
最近邻决策分类准确率为: 93.33 %

[Done] exited with code=0 in 1.633 seconds
```

最后，为了更直观地比较多元高斯分布参数估计、各函数密度估计及最近邻决策对样本分类的准确率，我做出了使用不同分类方法的准确率水平直方图如下（图像代码见`draw.py`，这里不再赘述）。



从图中可以看到，使用多元高斯分布参数估计的方法分类正确率是最高的，为97.33%，而最近邻决策规则的分类正确率是最低的，为93.33%。

以上是实验中级要求部分。

实验高级要求

接下来是对实验高级要求的介绍，即对于一维情况，进行k-近邻概率密度估计，对数据集的第1类中的特征x_1，用程序画出$k=1,3,5$时的概率密度估计结果。

对于k-近邻概率密度估计，总体思路是逐渐扩大以估计点为中心区域的体积，直到该区域体积内包含k个数据点。密度估计表达式如下。

$$P(x)=\frac{k}{NV}=\frac{k}{N*c_D*R_k^D(x)}$$

其中，$R_k(x)$是估计点x到第k个最近邻数据点之间的距离，c_D为D维空间中单位超球的体积，即：

$$c_D=\frac{\pi^{\frac{D}{2}}}{(\frac{D}{2})!}=\frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)}$$

其中$\Gamma(x)$为伽马函数。

而根据题目要求，容易的值维数$D=1$，因此，带入到单位超球体积公式中有：

$$c_1=\frac{\pi^{\frac{1}{2}}}{\Gamma(\frac{1}{2}+1)}=\frac{\pi^{\frac{1}{2}}}{\frac{1}{2}\Gamma(\frac{1}{2})}=\frac{\pi^{\frac{1}{2}}}{\frac{1}{2}\pi^{\frac{1}{2}}}=2$$

因此，整合到密度估计表达式中可以得到：

$$P(x)=\frac{k}{NV}=\frac{k}{2N*R_k(x)}$$

基于上述公式，代码实现如下。值得指出，公式中我引入了eps，即精度误差，目的是当两点间距离为0时，能保证分母不为0，使得代码正常运行。

```python
def knn_estimate(X_train, X_test, k, eps):
    Distance = np.abs(X_train - X_test)
    dist = sorted(Distance.tolist())[k]
    V = 2 * dist + eps
    N = len(X_train)
    prob = k / (N * V)
    return prob
```

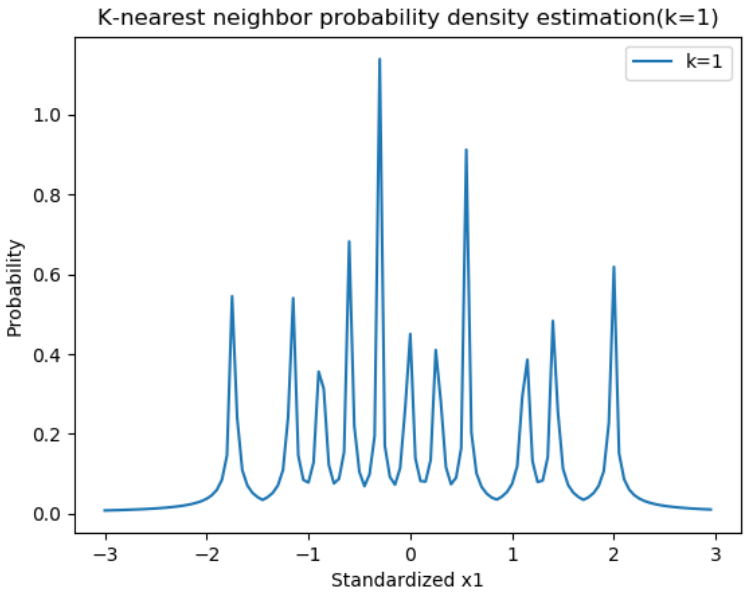
当$k=1,3,5$时，计算概率密度的代码为：

```python
X = load_data(filename)
X_1 = X[:, 0]
samples = np.arange(-3, 3, 0.05)
prob = [knn_estimate(X_1, i, k, eps) for i in samples]
```

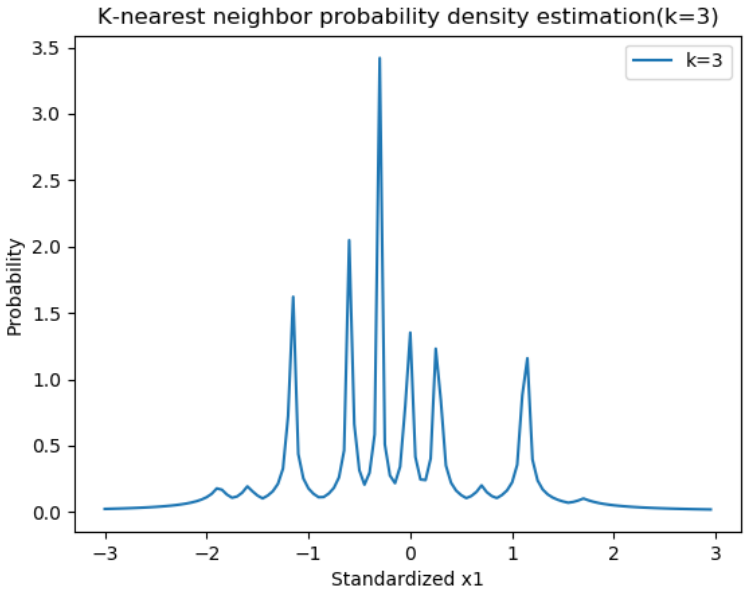
值得指出，这里对于eps和样本跨度（step）的选择，并非实验值，而是经验值。这里我测试了大量的数据，最后选定精度误差`eps=0.01`，样本点之间的跨度为`0.05`，在这样两个参数值下，能使得实验结果图更易于观察，避免精度误差和样本点跨度不合理造成的图像过于平滑或过于稀疏，从而使得对概率密度分布的观测出现误差。

下面给出实验结果图。

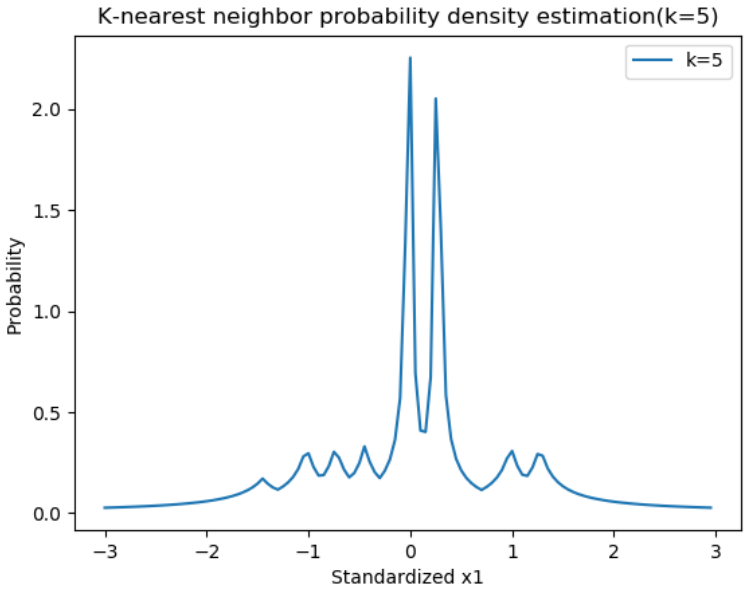
`k=1`时，概率密度估计结果如下。



`k=3`时，概率密度估计结果如下。



`k=5`时，概率密度估计结果如下。



以上是实验高级要求部分。

总结

以上是本次实验报告内容，感谢批阅！
