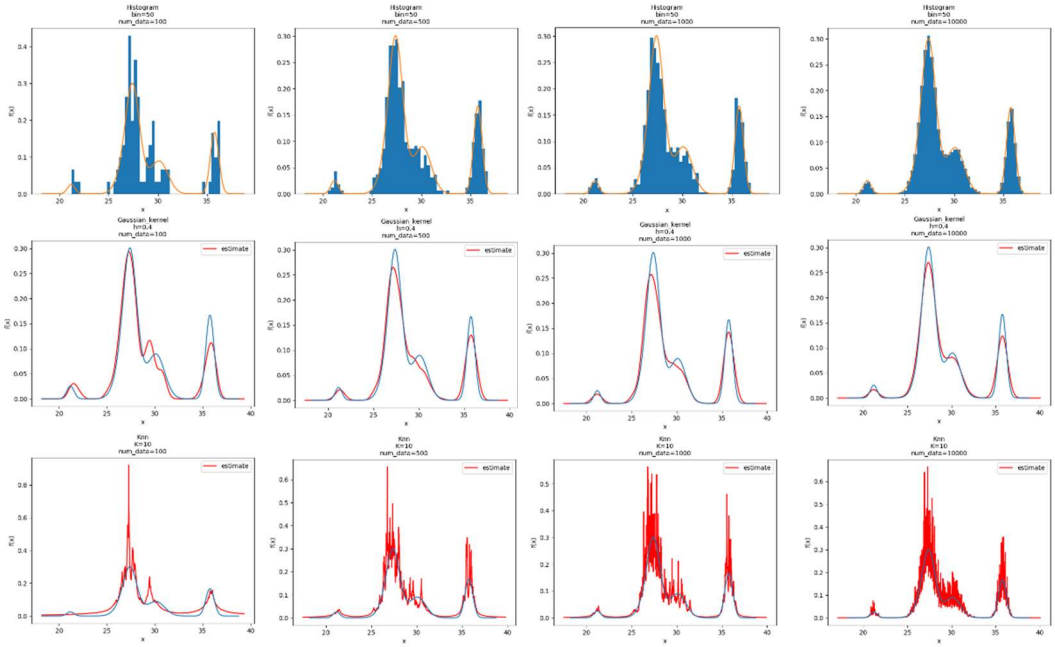
Assignment 1 Report

代码运行示例：

```
python source.py --func=hist --num_data=200 --bins=20
python source.py --func=kernel --num_data=200 --h=0.4
python source.py --func=knn --num_data=200 --k=10
python choose_bins.py --num_data=200
python best_h.py --num_data=200
```

一、数据量对密度估计的影响



100　　500　　1000　　10000

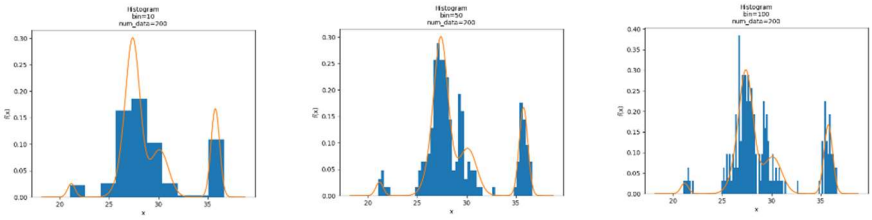
除方法一外，等间隔取 2000 个点

1.从理论上来说，**大数定律告诉我们当数据量越大时，频率越接近概率**（其中 Histogram 最能体现这一点）。因此这三个算法对密度的估计效果都应该**随着数据量的增大得到提升**

2.从实验结果来看，随着数据量的增大，三种算法的密度估计的效果也确实在一定程度上得到了提升。但也稍微**存在一些区别**：

- Histogram：对数据量的变化最为敏感，效果提升显著
- Kernel density estimation：对数据量不那么敏感，但效果也有提升
- Nearest neighbor：数据量的增大没有让预测曲线更加平滑，而是更加凸显了“尖峰”的存在（密度曲线在“尖峰”处变化更为剧烈），在“尖峰”以外的地方更加贴近真实情况

二、Histogram 方法和 bins 的选择

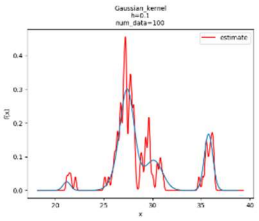


bins=10　　bins=50　　bins=100

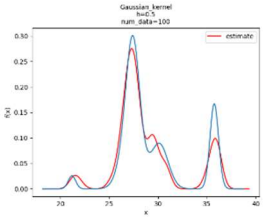
1. 从理论上来说，很难确定什么是最好的 bins，但有一点很显然，那就是 **bins 既不能太多也不能太少**。从我的角度出发，若 bins 太多，会导致对密度的估计波动很大，被噪声影响，但更能抓住局部特征；而 bins 太少，则导致密度估计过于平滑，无法提取局部特征（某些峰值消失）。
2. 我尝试了一种结合“交叉验证”和“极大似然”的方法，但发现 bins 的大小对结果的影响比较小，并且由于要求 bins 为整数　，因此未能通过此方法找到最好 bins。该方法的细节见下一个问题。
3. **Wiki 上给出了一些经验公式**，如:
   - **Square-root choice**
   $$\mathrm{k} = \lceil \sqrt{N} \rceil$$
   - **Sturges' formula**
   $$\mathrm{k} = \lceil \log_2 N \rceil + 1$$
   - **Rice Rule**
   $$\mathrm{k} = \lceil 2N^{1/3} \rceil$$
   - **Scott's normal reference rule**
   $$\mathrm{k} = \left\lceil \frac{\max x - \min x}{\frac{3.5\sigma}{N^{1/3}}} \right\rceil$$
   - **Shimazaki and Shinomoto's choice**
   $$\underset{h}{\operatorname{argmin}} \frac{2\bar{m} - v}{h^2}$$
   $$\bar{m} = \frac{1}{k}\sum_{i=1}^{k} m_i$$
   $$v = \frac{1}{k}\sum_{i=1}^{k} (m_i - \bar{m})^2$$
   $$k = \left\lceil \frac{\max x - \min x}{h} \right\rceil$$

   其中 N 为数据量，σ为标准差，k 为 bins 的数量，m_i表示在 Histogram 方法下第 i 个 bins 中元素的个数，choose_bins.py 中有这些公式的实现。
4. 在 N=200 时，前四个经验公式的值都在 10 左右，最后一个公式的值为 25。**但对于本数据集而言，bins 的值在 50 左右时结果较为平滑，并且能够体现出峰值。**通过改变 N 的值可以发现，**Shimazaki and Shinomoto's choice 方法**在多数情况下均表现得更好。

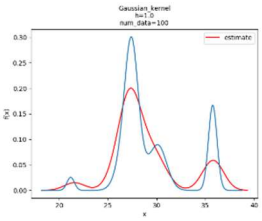
三、Kernel density estimation 方法和 h 的选择



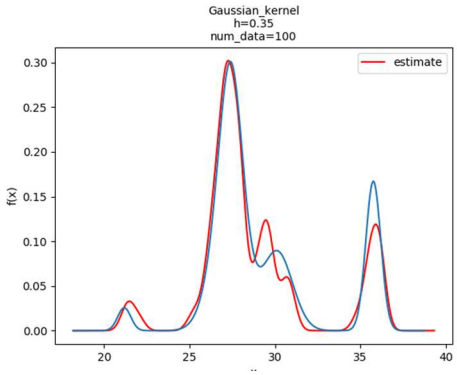
h=0.1



h=0.5



h=1.0



Num_data=100 时，h 的最优值为 0.35

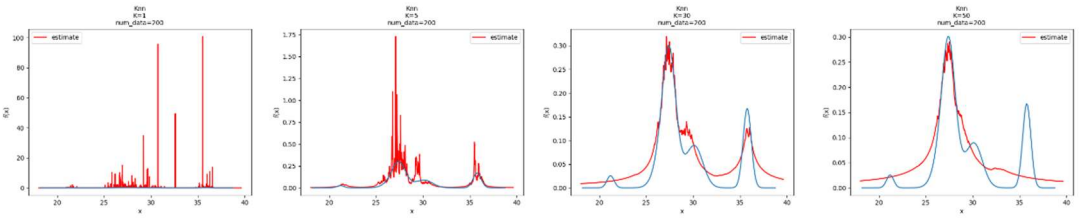
1. 通过改变 h，可以看到 h 的选择对密度估计的效果影响非常显著。
2. 通过查阅文献资料可以发现，有很多估计最佳 h 的方法，我**选择了一种易于理解并且容易实现的方法**来寻找最佳的 h，具体细节如下：
   - 对于每个样本点，用 kernel 方法计算出它的概率密度 p，但在计算时排除该点
   - 将每个样本点的概率密度相乘，得到关于 h 的函数 f(h)
   - 寻找使得 f(h)最大的 h 值

   对该方法的实现见 best_h.py。

   为什么该方法是对的呢？我没有找到对它的详细解释，这里给出我的理解。**既然这些样本点已经被获取到，说明这些样本点出现的概率应该很大**，因此通过使得它们概率密度的乘积最大，便可获得最好的 h。这应该是一种基于**极大似然**的思想。

   为什么在计算每个样本点的概率密度时要**排除自己**呢？这有一种交叉验证的感觉，朴素的说，我们能利用自己去估计自己的概率密度。从数学公式出发，可以看到，**公式** $\mathbf{p(x)} = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{\sqrt[2]{2\Pi h^2}} e^{-\frac{(x-x_n)^2}{2h^2}}$ **中，若** $\mathbf{x} = x_i$**，很明显 h->0 时，其结果越大**，但这很明显不符合事实。
3. 运用上述理论，再加上 scipy.optimize 中的 minimize 方法，**在 data_num=100 的情况下，可以发现 h 收敛到 0.35**。并且实际效果表现得还不错。（data_num 等于其它值时表现得也都很不错）

四、Nearest neighbor 方法



K=1　　K=5　　K=30　　K=50

1. 从图中可以看到，k 对密度估计的结果影响非常大。从我的角度出发，当 k 过小时，由于噪声的存在，导致密度估计波动很大，但更能抓住局部特征（峰值）；当 k 过大时，则导致密度估计过于平滑，无法提取局部特征（某些峰值消失）。
2. 假设样本点为{x1,x2,……xn},，且有 x1<=x2<=……<=xn。注意到当 x<=x1 时，离自己最近的 k 的点为 x1,x2,……xk，则相应的 V=2*(xk-x)，即 x=xk-V/2。此时有：

$$\int_{-\infty}^{+\infty} \boldsymbol{p(x)dx} > \int_{-\infty}^{\boldsymbol{x1}} \boldsymbol{p(x)dx} = \int_{\infty}^{\boldsymbol{2(xk-x1)}} \frac{\boldsymbol{k}}{\boldsymbol{NV}} \boldsymbol{d}\left(\boldsymbol{xk} - \frac{\boldsymbol{V}}{\boldsymbol{2}}\right) = \frac{\boldsymbol{k}}{\boldsymbol{2N}} \int_{\boldsymbol{2(xk-x1)}}^{\infty} \frac{\boldsymbol{1}}{\boldsymbol{V}} \boldsymbol{dV} = +\infty$$

由上式可知该方法得到的分布的概率密度积分不会收敛到 1。

参考文献

[1] https://en.wikipedia.org/wiki/Histogram

[2] http://176.32.89.45/~hideaki/res/histogram.html

[3] http://cran.irsn.fr/web/packages/kedd/vignettes/kedd.pdf