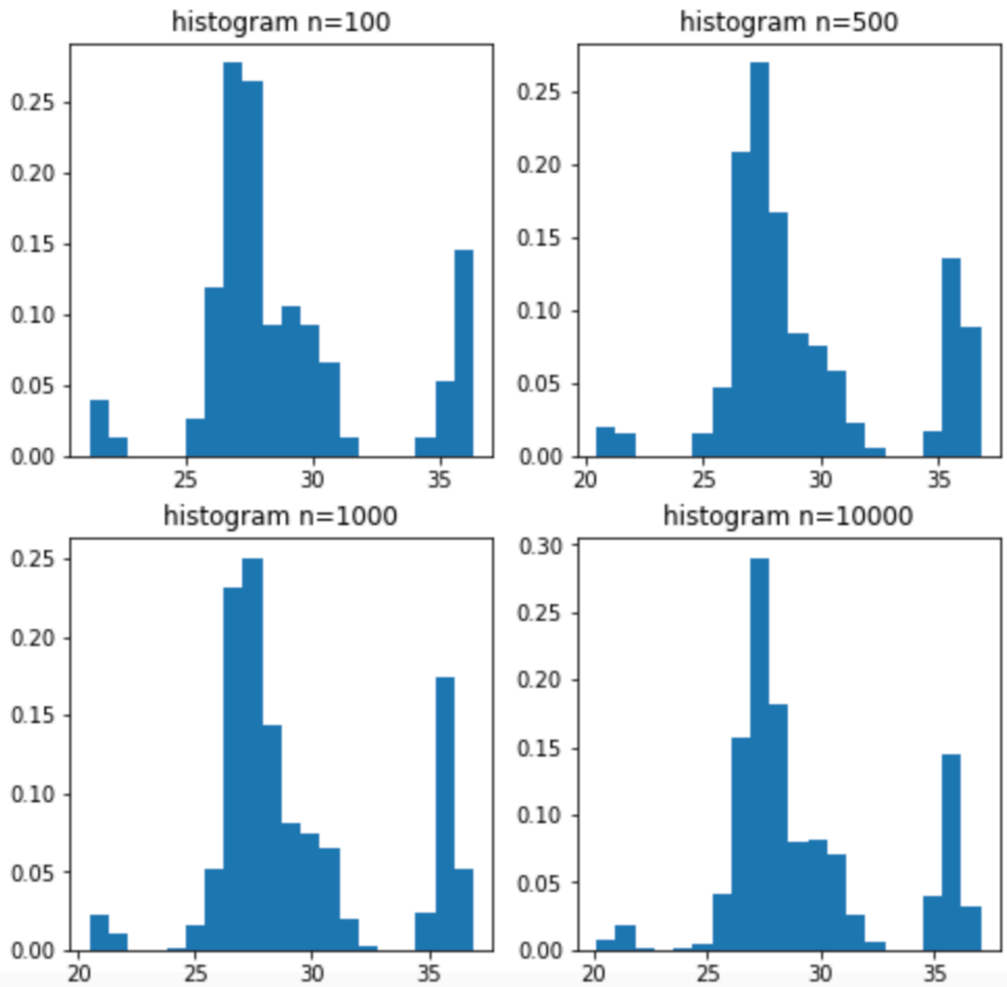
lab1实验报告

采样数量变化对估计结果的影响

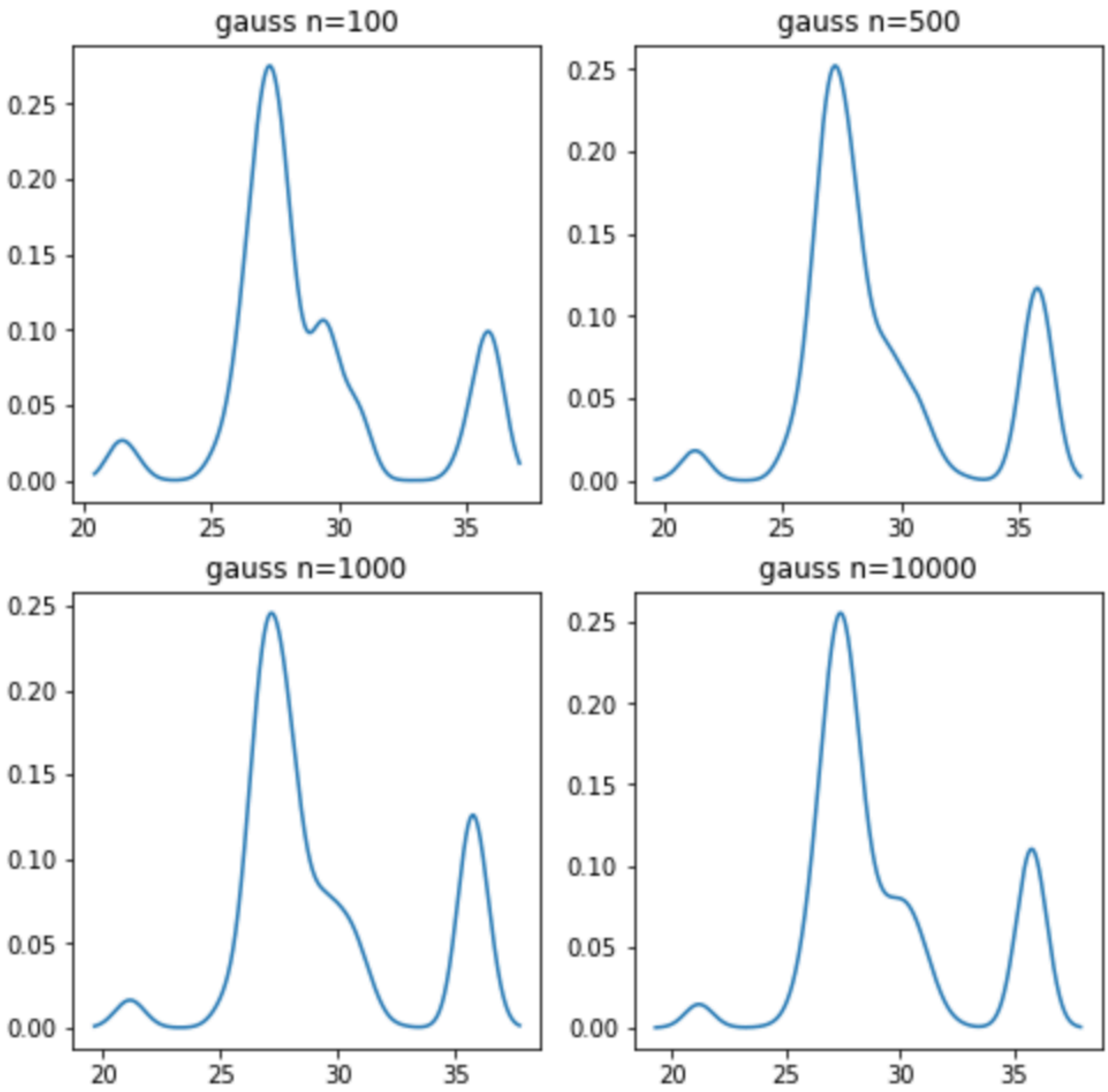
从经验上来说，采样的数据越多，估计出的概率密度分布也会更贴近真实分布，下面分别对三种算法的实验结果进行分析：

直方图



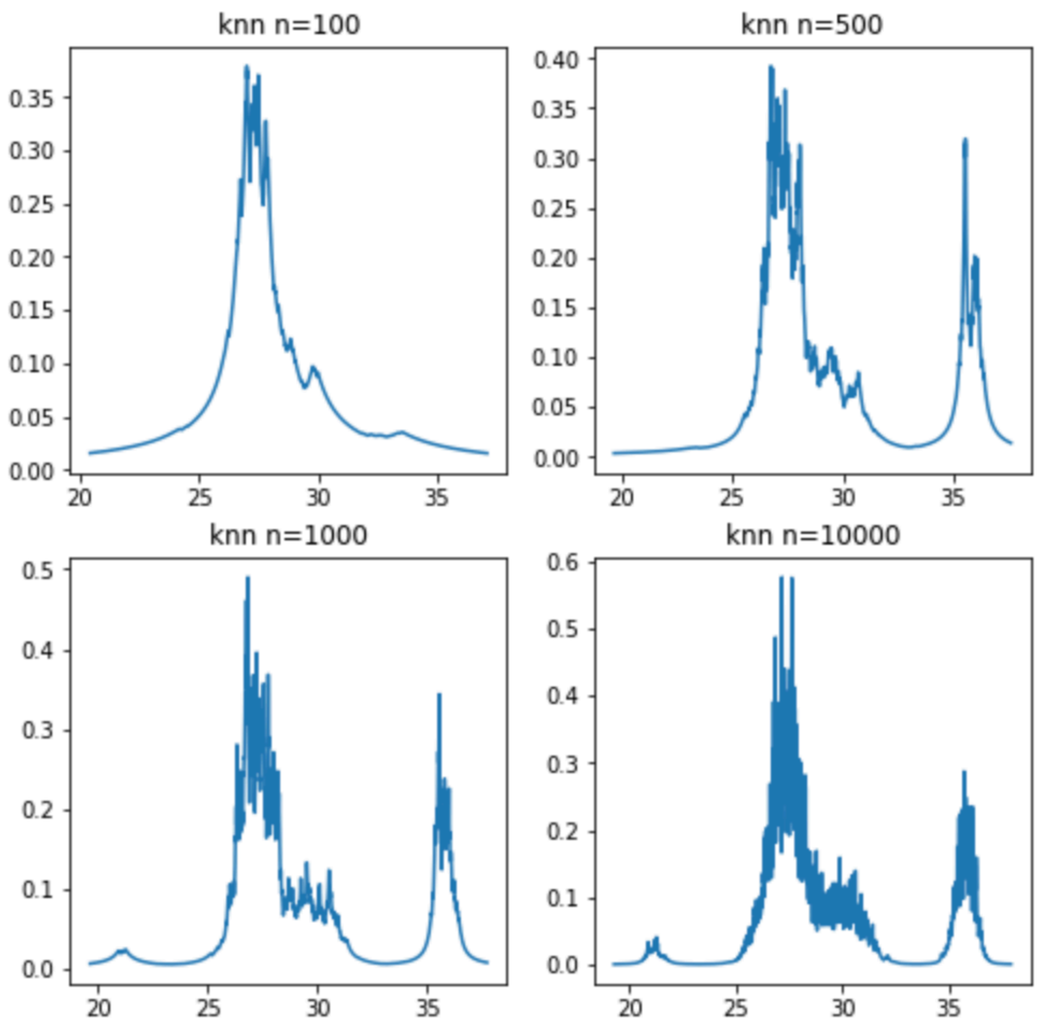
固定bins的数量为20，n分别取100，500，1000，10000。我们可以从图中发现，随着n的增加，直方图估计的分布也更加贴近真实分布，不过从n=500开始，估计的分布就已经较为准确，当n很大时，牺牲一定的精确度换取时间也是一种选择。

高斯核



固定参数h的值为0.5，结果如上图所示，可以发现高斯核估计的准确程度是随n增加而增加的，不过同样地，当n较小时（例如100）的效果也很接近真实分布。

k近邻



固定k的值为20，发现随着n的增加，虽然更贴近真实分布，但分布中的噪声也相应增加，而n较小时虽然曲线比较光滑，但分布中的一些峰也被光滑掉了。相比较之下，n=500是一个不错的选择，贴近真实分布的同时也减少了噪声。

总结

上述三种算法，当n增加时，估计得到的分布都会更加贴近真实分布，但当n的数量级较大时，全部采样的复杂度可能难以承受，此时牺牲一部分精度，提升算法的效率也是不错的选择。另外对于k近邻算法，n增大的同时噪声也会增大，此时可以考虑选择一个折衷大小的n。

直方图

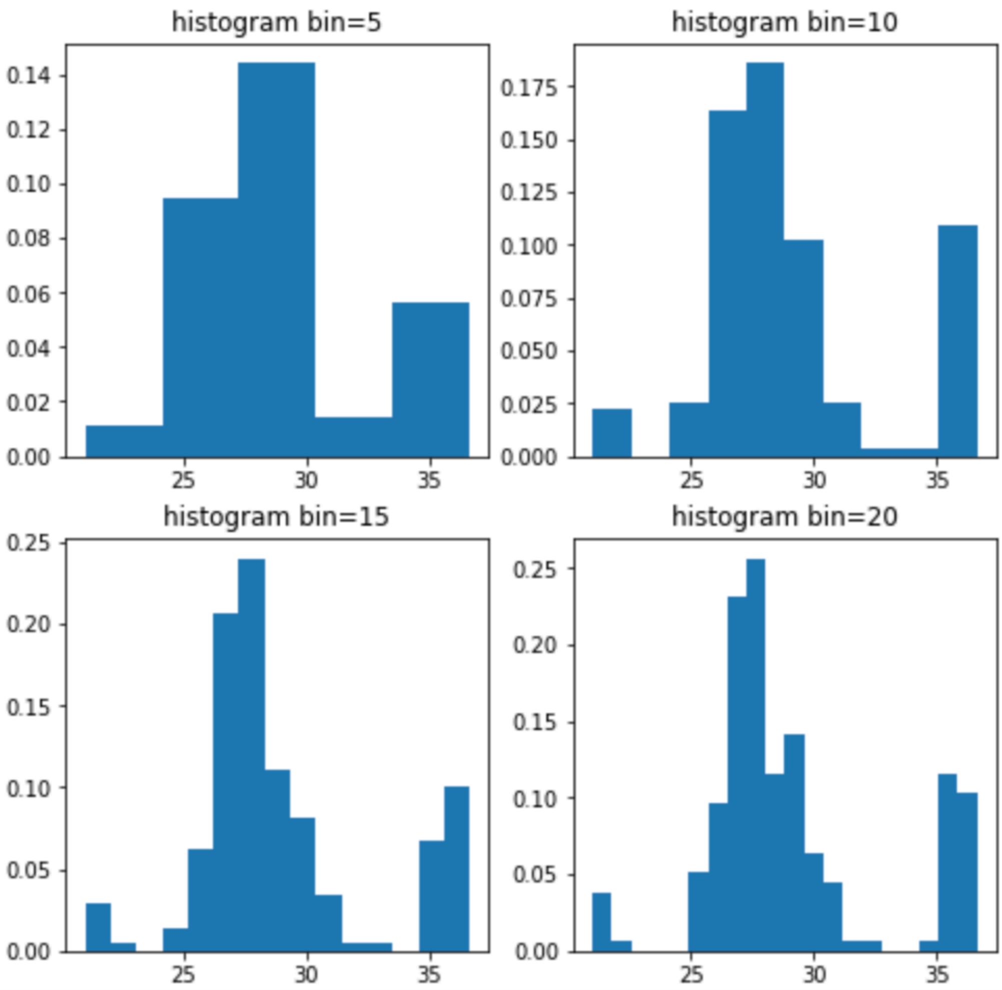
参数bins的选择

对于bins参数的选择，我们可以首先考虑一些经验性的规则，有如下几种：

1、Sturges's formula: $bins = \lceil \log_2 n \rceil + 1$

2、Square-root choice: $bins = \lceil \sqrt{n} \rceil$

根据这些不同规则得到的h值，我们确定一些候选值5、10、15、20(固定n为200)，打印它们的分布进行观察：

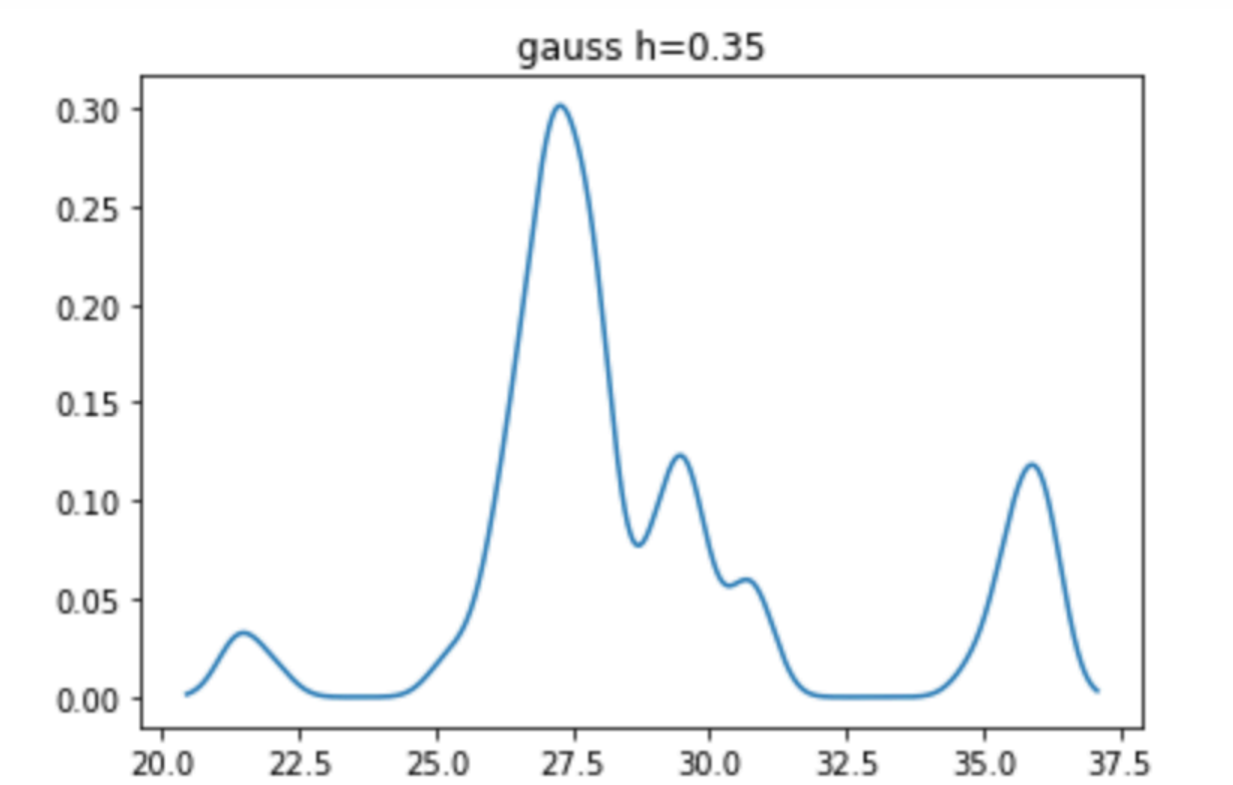


评价一个bins值的好坏可以从生成分布的光滑程度来判断，有过多尖刺的分布无法反映真实分布的结构，而过于光滑的分布则无法描述真实分布的重要性质（如峰的存在）。在折衷的考虑下，我们选取15作为最终的bins值。

高斯核

参数h的选择

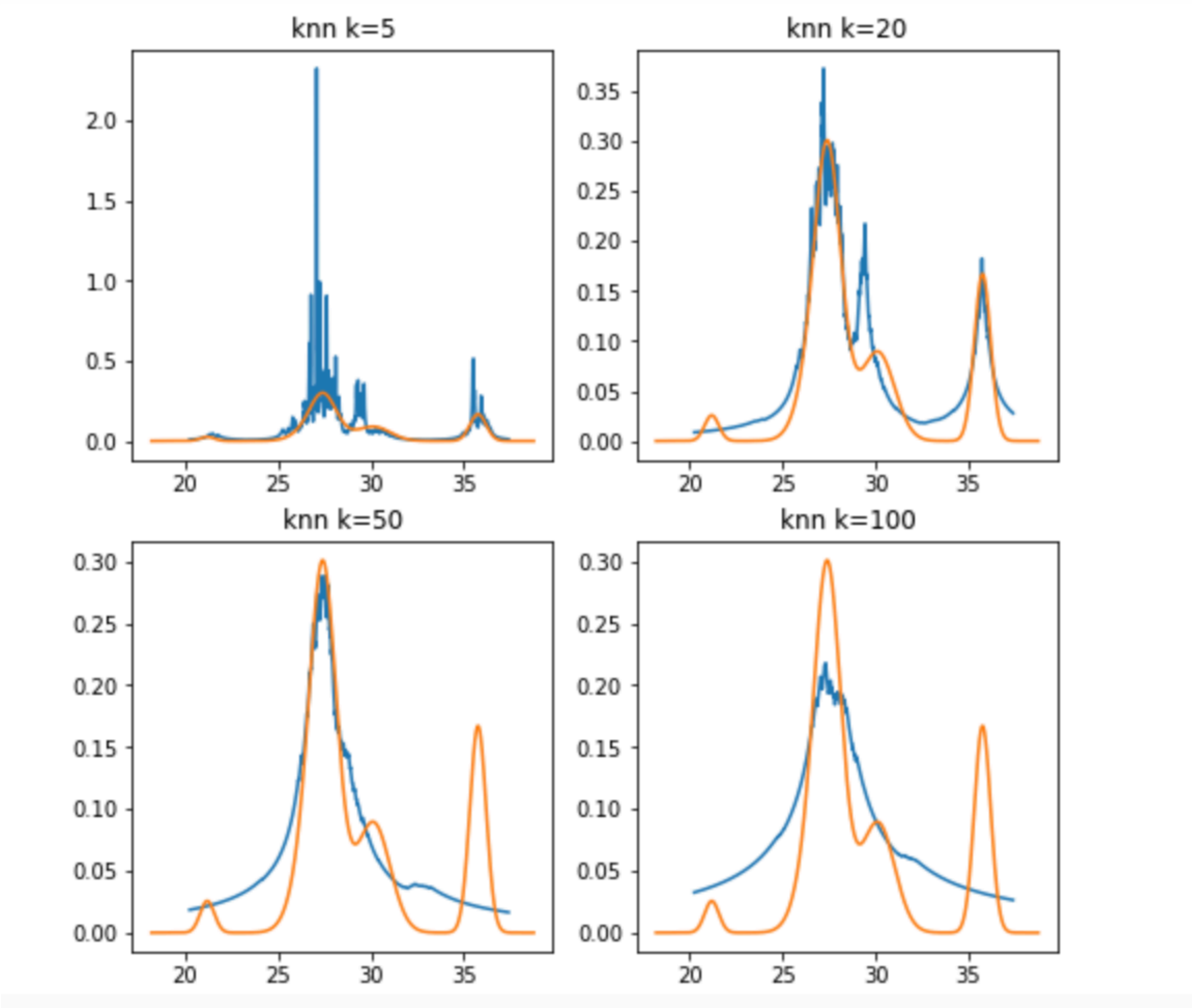
对于h，我们可以采用k折交叉验证的方式来选取参数。而k折交叉验证，就是将数据集平均分为k个部分，每轮选择一个部分作为测试集，其余作为训练集。对于测试集中的x, 利用训练集中的点根据高斯核计算x的似然值，对测试集中的每个点计算似然值的log并相加，和越大则选取的h所算得的分布越接近真实分布。首先我们通过随机取值大致确定h的范围为$[0.1, 1]$, 在区间内均匀取点作为候选h计算似然，最终似然值最高的h便是我们选择的参数，为0.35，n固定为100。效果如下：



k近邻

实验效果

固定n为200, 分别选取k为5, 20, 50, 100，并与真实分布比较，得到结果如下：



其中k=50效果最好，k较小时噪声很大，k较大时则平滑掉了峰的性质。

不收敛的理论证明

假设$x_1, x_2, \ldots, x_n$已经按照大小排序，则x_n为数据集中的最大值

$\int_{x_n}^{+\infty} p(x)\,dx = \int_{x_n}^{+\infty} \frac{k}{N*2*(x-x_{n-k+1})}\,dx = \frac{k}{2*N} * (\ln +\infty - \ln(x_n - x_{n-k+1}))$

积分显然发散，若k为1，我们可以取积分下界稍微大于x_n。

又$\int_{-\infty}^{+\infty} p(x)\,dx \geq \int_{x_n}^{+\infty} p(x)\,dx$,所以生成的并不是有效的分布。

自由探索

在这一节中，我们主要讨论近邻估计中算法的时间复杂度。最为朴素的实现方法是对于每一个需要估计概率密度的x，遍历一遍数据集并计算每一个数据点与x的差值进行排序，这样的时间复杂度是$O(nmlogn)$，其中n代表数据集大小，m代表需要估计的x个数，显然当n较大时，这样的复杂度是难以承受的。我们可以考虑利用二分查找对该算法进行优化，首先对数据集进行排序，对于每一个x，二分查找到最大的小于x的数组位置pos，令$l = pos + 1, r = pos$, 最后将l和r向数组的两边延伸，直到$r - l + 1 = k$，代码如下：

```
while r - l + 1 < k:
      if l == 0:
          r += 1
      elif r == num_data - 1 or x - sampled_data[l - 1] < sampled_data[r +
1] - x:
          l -= 1
      else:
          r += 1
```

这样算法的复杂度是$O(nlogn + mlogn + mk)$, 一般情况下k的值都比较小，所以当n较大时复杂度可以看作 $O(nlogn)$, 相对于朴素算法有了明显的提升。朴素和二分算法分别实现在代码中的knn_estimation和fast_knn_estimation函数中。