Report for Assignment_1

Introduction

This is an assignment where we need to use **python** implementing three non-parametric estimation algorithms (including histogram method, kernel density estimate and the nearest neighbor method) to estimate the distribution of the given data set. The data set contains 10000 samples from an unknown 1-D distribution. We can access these data through calling function provided in handout. I implement all three algorithms and use matplotlib visualizing them. You can see figures through typing command on terminal and get more details through "python3 source.py -h". You can also use other three python code to verify my work, but most of them are duplicated.

Methods

Histogram method

I just make the original histogram method few changes.I devide the sampled_data into test_data and valid_data.There is a new variable **Loss**,which is defined:

$$Loss = \prod_{i=1}^{N} (p_{test}(i) - p_{valid}(i))^2$$

Where N refers to the number of bins, and p(i) is same as the definition in histogram method:

$$p(i) = \frac{n_i}{N\delta_i}$$

Here we regard the square of deviation between two distribution as the loss.Loss reflect the generalization error.

Kernel density estimators

Kernel density estimator(KDE) is a method depends on

$$p(\vec{x}) = \frac{K}{NV}$$

For a fixed data set which has N data,we can fix V and determine K from the data.We can also use the Gaussion kernel to smooth density model.

$$p(\vec{x}) = \frac{1}{N}\prod_{n=1}^{N}\frac{1}{(2\pi h^2)^{1/2}}exp\{-\frac{||\vec{x}-\vec{x}_n||}{2h^2}\}$$

KDE with Gaussion kernel gives a continuous function p(**x**).Here we choose evenly distributed and intensive point **x** and calculate the corresponding value **y**.Then we draw the points in the axis,using straight line to connect them.If the we take plenty of points,then the discrete point can nearly estimate the true KDE. I also partition the data set to vertify the training model.

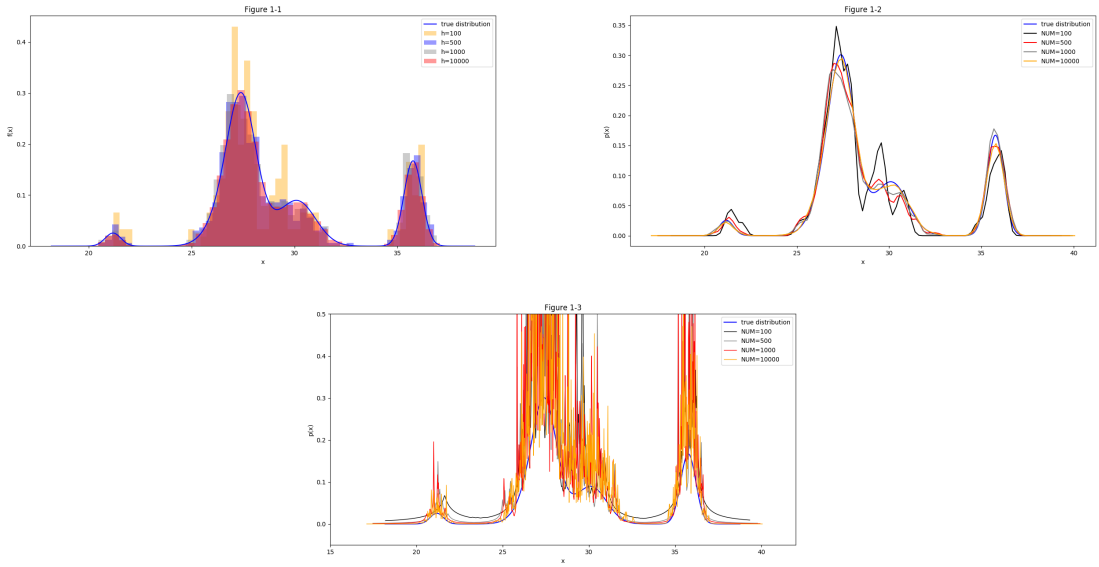
K Nearest-neighbour methods

We fix K and determine V from the data,which give rises to K Nearest-neighbour-methods(KNN).For each point x,we need to find the nearest k point in training set to calculate the corresponding value p(x).For each point,we get **d** denote the distance between the point x and the point t in training set.We sort the elements in **d**,get the k_{th} smallest number and take it as the V.

Discussion

1. About dataset size

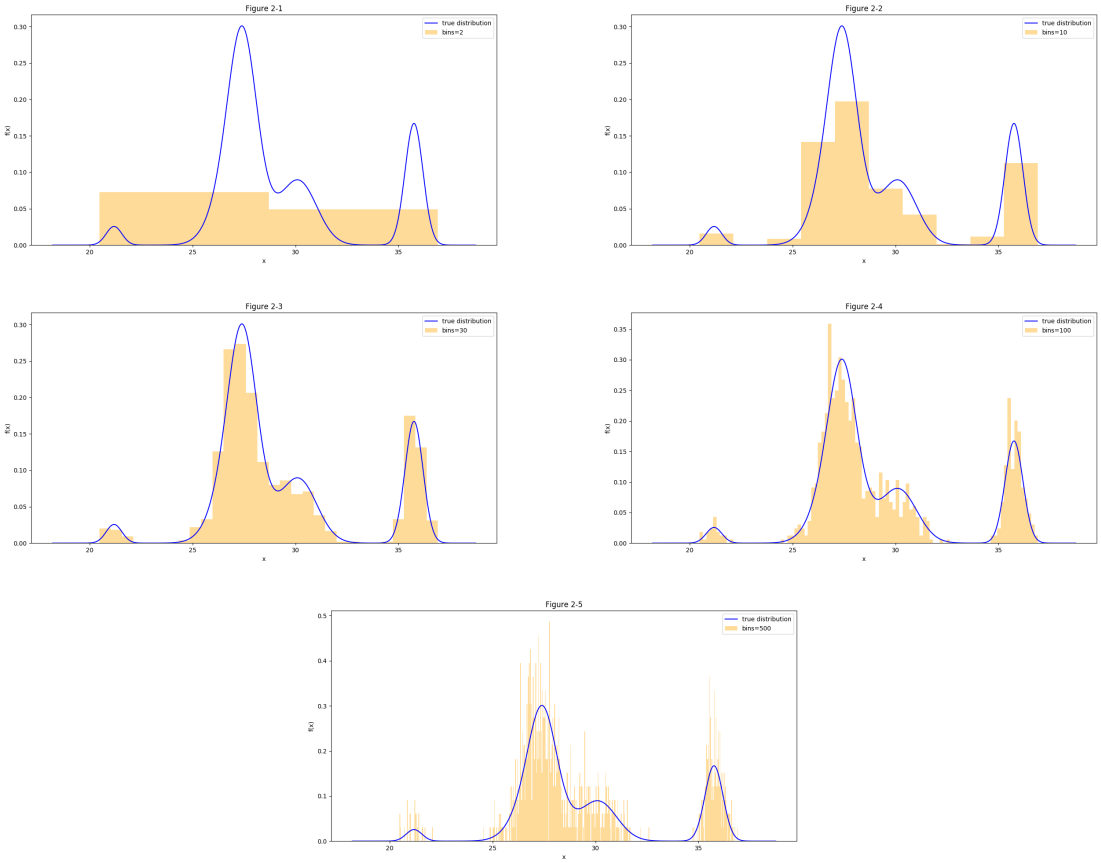
Intuitively,we could make the estimation better when we enlarge dataset size.According to **Law of Large Numbers**,the average of the results obtained from a large number of trials should be close to the expected value, and will tend to become closer as more trials are performed.Therefore,we can obtain an ideal model to describe the distribution,when we feed sufficient data to train the model.



> Figure1 Fix bins=50 in Histogram estimation, h=0.2 in KDE and k=3 in KNN.

Data visualization can help proving my opinion. Noting that models behave worse than others if we feed in only few data(100 data). However, the model built by KNN methods doesn't improve when the dataset increase. It can't well fit in with the true distribution and the model is steep. We can also notice that the model can't improve much when the size comes to a certain scale.I will take num_data=1000 for the following exploration.

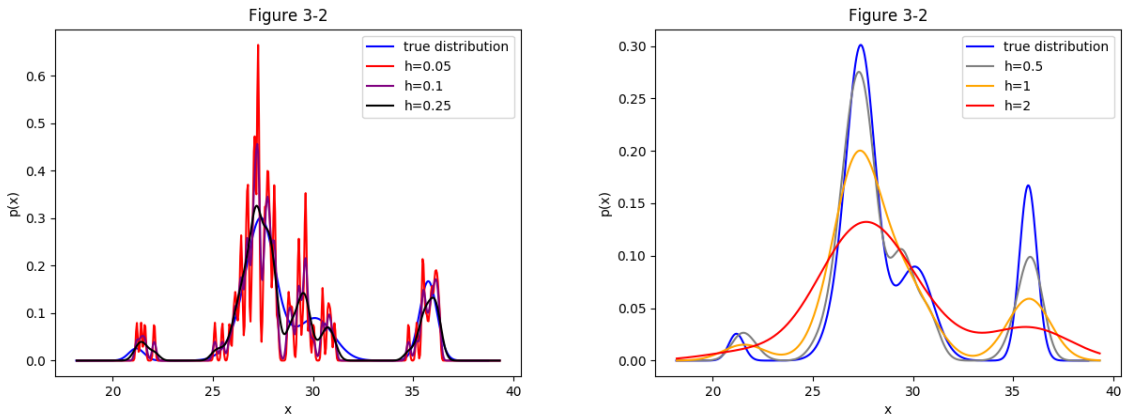
2. Histogram Estimation



> Figure 2 Illustration of histogram estimator applied to the same data. The best model is obtained for some intermediate values of bins.

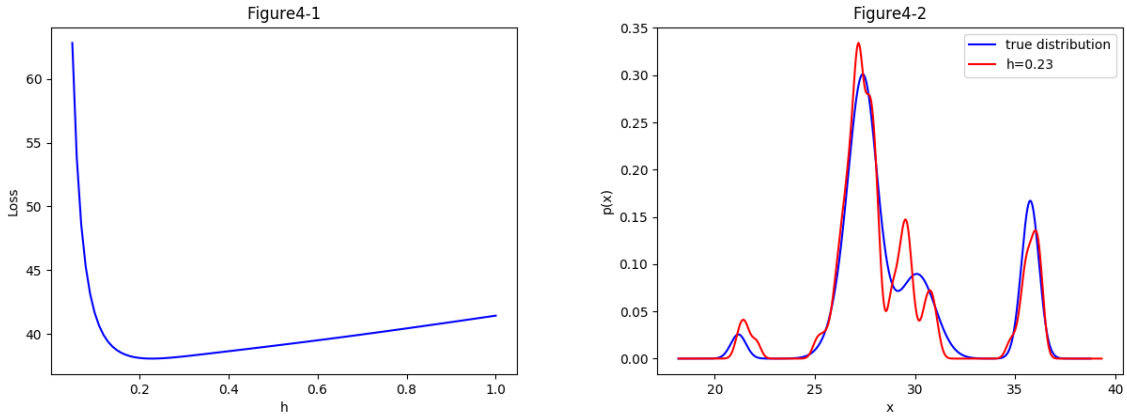
Notice that the figure behave well when bins take 30 and 100, the best choice of bins is the intermediate values briefly. But what is the definition of intermediate values? On one hand,if the bins is small, it probably can't extract feature. On the other hand, it would over-fitting if we set too much bins which lead to high **empty-bin rate**. We can partition the dataset and use **validation-cross** to choose the suitable quantity. Also, there are many **"rule of thumb"** can help figure out this problem.(Some useful methods of choosing the number of bins)

3. Kernel Density Estimation



> Figure 3 Kernel Density Estimation with Gaussion kernel. There are six curves whose h=0.05, 0.1, 0.25, 0.5, 1, 2 seperately.

The KDE showed in **Figure 3** allows us to visualize the difference among different distribution when we tune h. We can find that the curve would be steep if h is small and the curve would be too smooth to present the true distribution when h is large. When h is intermediate, such as h=0.25 and h=0.5, the curve wouldn't go up and down frequently (under-fitting) or have one and only maximum value (over-fitting). The curve can well present and predict the true distribution. So choose a suitable h for Gaussion kernel can effectively improve KDE model. But how to choose the best h?



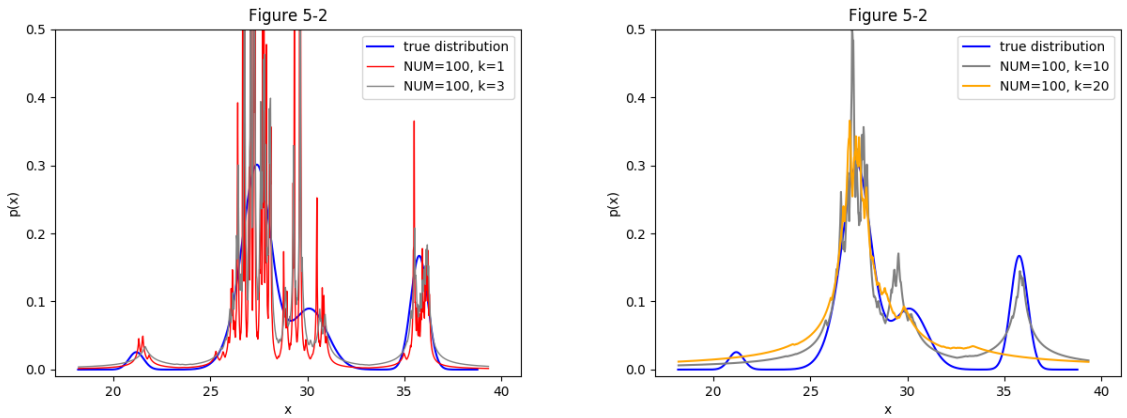
> Figure 4 Figure 4-1 display the result of Cross-validation using cross entropy to evaluate the estimation. The figure is an convex function which has a minimum, where h minimizes the cross entropy. Figure 4-2 show the best estimate I could achieve with **num_data**=100.

One simple but useful method is **Cross-validation**, which cut the dataset apart, using the vast majority of dataset for training and the rest for verification. In practice, I use 80% for training and another 20% for verification. I decide to use **cross entropy** to calculate the loss between training set and validation set. Here're the formula I choose to take

$$Loss = \sum_{n=1}^{N} p(x_n) \ln \hat{p}(x_n) \propto \sum_{n=1}^{N} \ln \hat{p}(x_n)$$

where N represents the size of validation set, $\hat{p}(x_n)$ represents the expected probability density of x_n. And for each point in the validation set, we assume that they are of the same probability. So what we need to do is calculating the sum of $\hat{p}(x_n)$ when h takes different value. Figure 4 show that the loss function is convex and it gets the minimum when **h=0.23**.

4. K Nearest Neighbor Methods



> Figure 5 Some distribution drawed by KNN. The curve is sharply peaked and can hardly behave well in presenting the true distribution.

Here are a simple way to prove that KNN does not always yield a valid distribution. For simplicity, we take 1-D distribution as an example. Assume that there are n data in total, which are sorted beforehand, so we have n point $x_1, x_2 \ldots, x_n$ and guarantee $x_1 \leq x_2 \leq \ldots \leq x_n$. For every x in $(-\infty, x_1)$, it's easy to proof that $p(x) = \frac{K}{2N(x_k - x)}$.

$$\int_{-\infty}^{+\infty} p(x)dx \geq \int_{-\infty}^{x_1} p(x)dx = \int_{-\infty}^{x_1} \frac{K}{2N(x_k - x)}dx = \frac{K}{2N}(-\ln(x_k - x)|_{-\infty}^{0}) \to +\infty$$

When data is 2-D, 3-D or even higher dimension, the prove is the same. So it's hard to choose a coefficient properly to make KNN distribution a valid distribution. But we can still regard it as an efficient method of non-parameters estimate. We can also choose a finite interval for KNN to normalize the function, which I think an method to optimize the KNN.