Assignment 1

This assignment is about the exploration of non-parametric methods for density estimation, including histogram method, kernel density estimation and k nearest neighbor method.

0. Overview

0.1 Structure

- source python file
  - `source.py`
- solution files
  - `histogram.py`: implementation of histogram method
  - `kernel.py`: implementation of kernel density estimation
  - `knn.py`: implementation of k nearest neighbor
- other file
  - `plot.py`: my wrapper of the gm1d plot function for showing gm1d in each subplot

0.2 Packages

- Packages including `numpy`, `matplotlib`, `scipy` are used for implementing algorithms.
- In addition, `argparse` is used for organizing the program, which can be installed via:

```
pip install argparse
```

0.3 Usage

- There are example usages in the following context
- Show the help message via:

```
python source.py -h
```

0.4 Parameters

- `n`: the amount of sample data
- `b`: the amount of bins in histogram method
- `h`: the parameter `h` in Gaussian kernel
- `k`: the amount of nearest neighbors

1. histogram method

1.1 Implementation

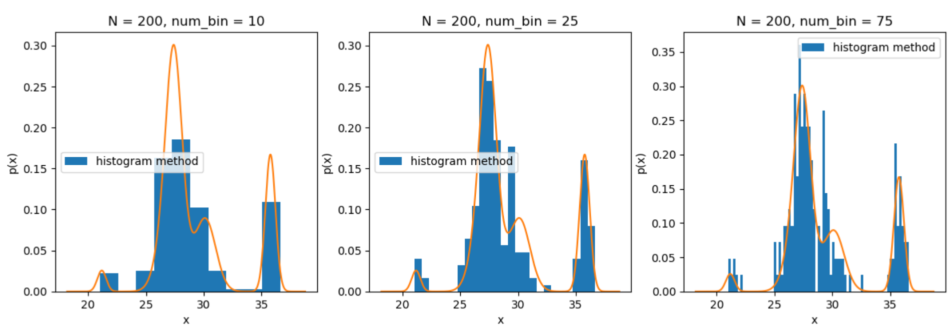
- This can be easily implemented with `hist` function in `matplotlib.pyplot`.
- Example usage (replace `200` and `25` with desired values of `n` and `b`):

```
python source.py --algorithm histogram_method --n 200 --b 25
```

1.2 Vary num_data

- By increasing `n` in the usage above, the result histogram **gets less spiky, captures more underlying features and gets closer to the original distribution**.
- As `n` grows, `b` can be increased for a certain amount which makes the histogram better without making the histogram spiky.

1.3 Vary the number of bins



- This is the result of `N = 200` and `b = 10 or 25 or 75`, which can be obtained with the following command:

```
python source.py --algorithm histogram_result
```

- The effect of `b`:
  - If `b` is too small as shown on the left, the histogram is **too smooth** and **captures little details** of the original distribution (orange curve).
  - If `b` is too large as shown on the right, the histogram gets **too spiky** with **a lot of structures that are not presented** in the original distribution.
  - The best `b` is some intermediate value as shown in the middle, which **captures the details without being spiky**.

1.4 Find the optimal number of bins

This is achieved with Shimazaki and Shinomoto's choice.

Implementaion

- generate an array of bin widths

- calculate the cost of each bin width with this formula:

$$C(\Delta) = \frac{2k - v}{\Delta^2}, where\ k\ is\ mean, v\ is\ variance\ and\ \Delta\ is\ bin\ width$$

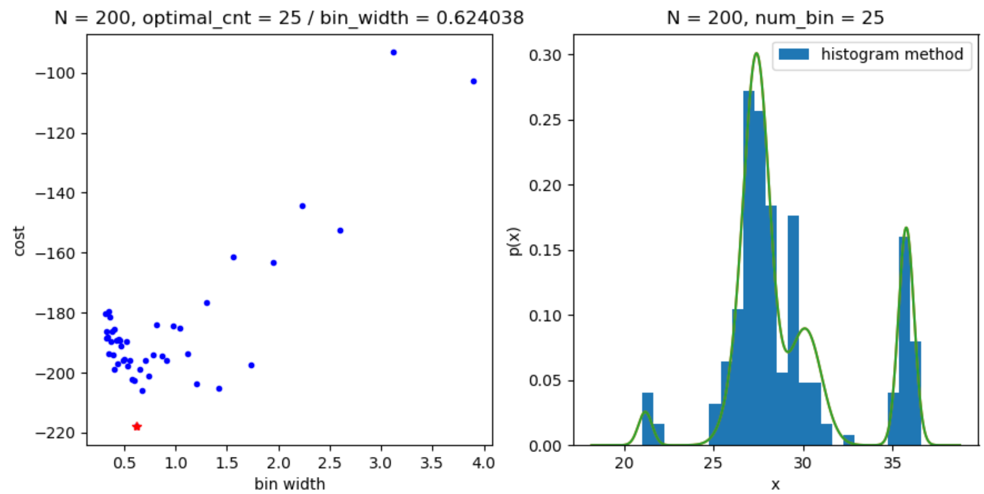
- The bin width that achieves the lowest cost is the best bin width

Usage

- The following usage finds the best `b` with a given `n`:

```
python source.py --algorithm optimal_bin --n 200
```

Result



If `n = 200`, `b = 25` is the best result, which is shown above: + [4, 5, 6, ..., 50] number of bins are tested + on the left, the cost of each bin width is shown in the scatter plot . The histogram with the optimal bin width is shown on the right

1.5 Other methods for finding the optimal amount

- Square-root choice

$$b = \lceil \sqrt{n} \rceil$$

- Sturges' formula

$$b = \lceil \sqrt{\log_2 n} \rceil + 1$$

- Rice Rule

$$b = \left\lceil 2n^{\frac{1}{3}} \right\rceil$$

2. kernel density estimate

2.1 Implementation

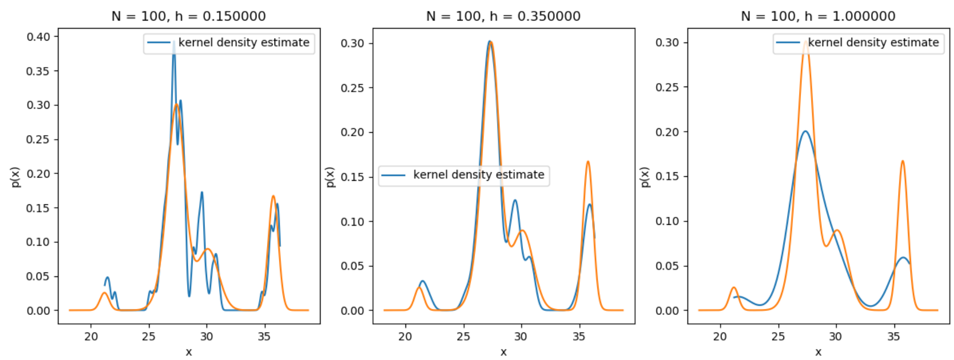
- Implemented with numpy operation based on Gaussian kernel
- Example usage (replace `200` and `0.35` with desired values of `n` and `h`):

```
python source.py --algorithm kernel_density_estimate --n 200 --h 0.
35
```

2.2 Vary num_data

By increasing `n` in the usage above, the result curve **captures more underlying features and gets closer to the original distribution**.

2.3 Vary h



- This is the result of `N = 100` and `h = 0.15 or 0.35 or 1`, which can be obtained with the following command:

```
python source.py --algorithm kde_result
```

- The effect of `h`:
  - If `h` is too small as shown on the left, the curve gets **too spiky** with **a lot of structures that are not presented** in the original distribution.
  - If `h` is too large as shown on the right, the curve is **too smooth** and **captures little details** of the original distribution.
  - The best `h` is some intermediate value as shown in the middle, which **captures the details without being spiky or too smooth**.

2.4 Find the optimal h

This is achieved with the idea of maximum likelihood.

Implementation

- Test different value of `h` to achieve the highest probability:

$$L(h) = \prod_{i=1}^{m} \frac{1}{N} \sum_{n=1, n\,!=i}^{N} \frac{1}{(2\pi h^2)^{\frac{1}{2}}} e^{-\frac{(x_i - x_n)^2}{2h^2}}$$

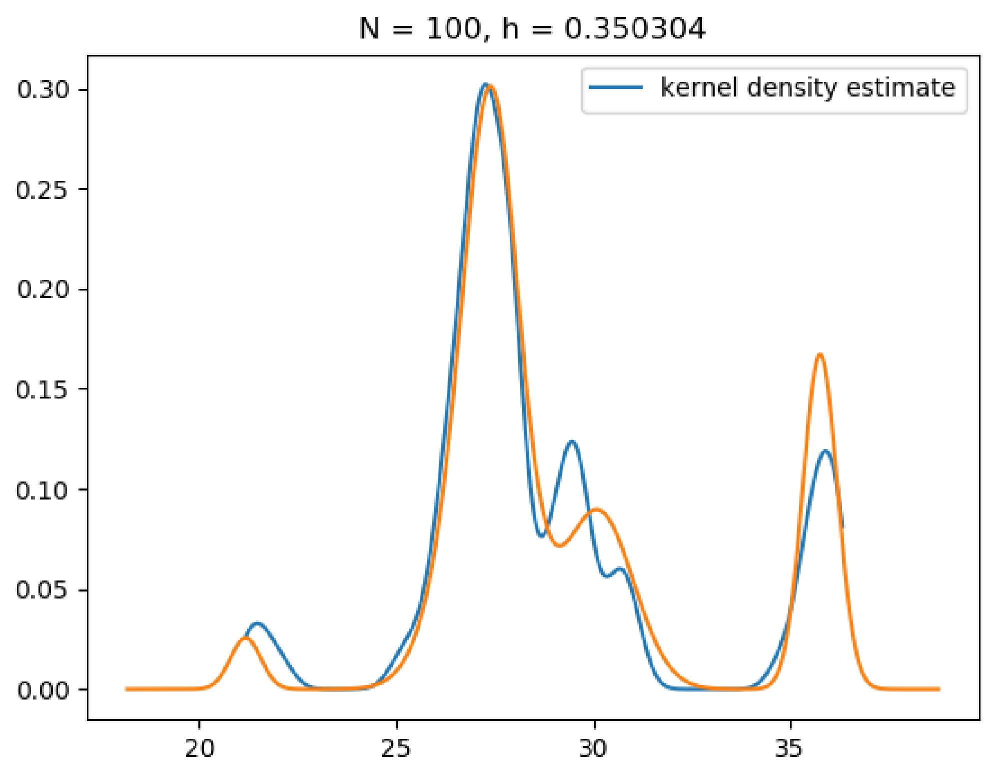
- This can be achieved easily with `minimize` in `scipy.optimize`

Usage

- The following usage finds the best `h` with a given `n`:

```
python source.py --algorithm optimal_h --n 200
```

Result



If `n = 100`, `h = 0.35` is the best result, which is shown above.

3. k nearest neighbor method (kNN)

3.1 Implementation

kNN is implemented with the following 3 methods: * Naive approach: + sort sample data and test data + set a window with fixed width `k` on sample data + for each test data, slide the window to include the k nearest neighbors + get V with the following formula

- Matrix operation approach:

- implemented with numpy

- KD tree:
  - implemented with `KDTree` in `scipy.spatial`

3.2 Usage

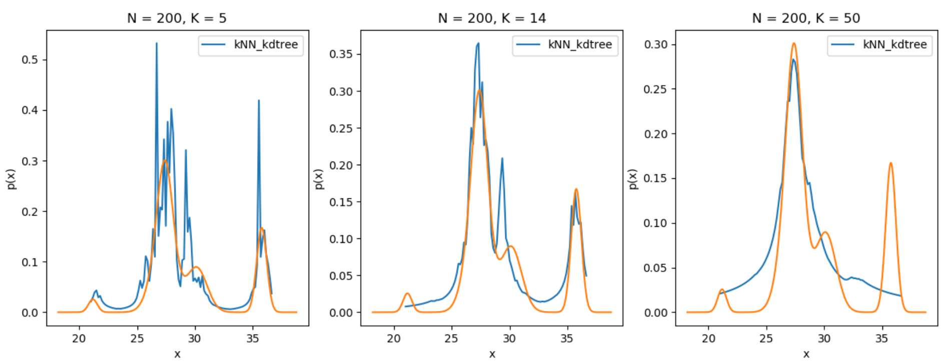
Example usage (replace `200` and `14` with desired values of `n` and `k`):

```
python source.py --algorithm kNN_kdtree --n 200 --k 14
```

3.3 Vary num_data

By increasing `k` in the usage above, the result curve **captures more underlying features and gets closer to the original distribution**.

3.4 Vary k



- This is the result of `N = 100` and `k = 5 or 14 or 50`, which can be obtained with the following command:

  ```
  python source.py --algorithm kNN_result
  ```

- The effect of `k`:
  - If `k` is too small as shown on the left, the curve gets **noisy decision boundaries** with **a lot of structures that are not presented** in the original distribution.
  - If `k` is too large as shown on the right, the curve gets **over-smoothed boundaries** and **captures little details** of the original distribution.
  - The best `k` is some intermediate value as shown in the middle, which is **large enough to minimize error rate** and __small enough to only include nearby samples.

3.5 Find the optimal k

This is achieved with the rule of thumb.

Implementation

- Get the best k simply with:

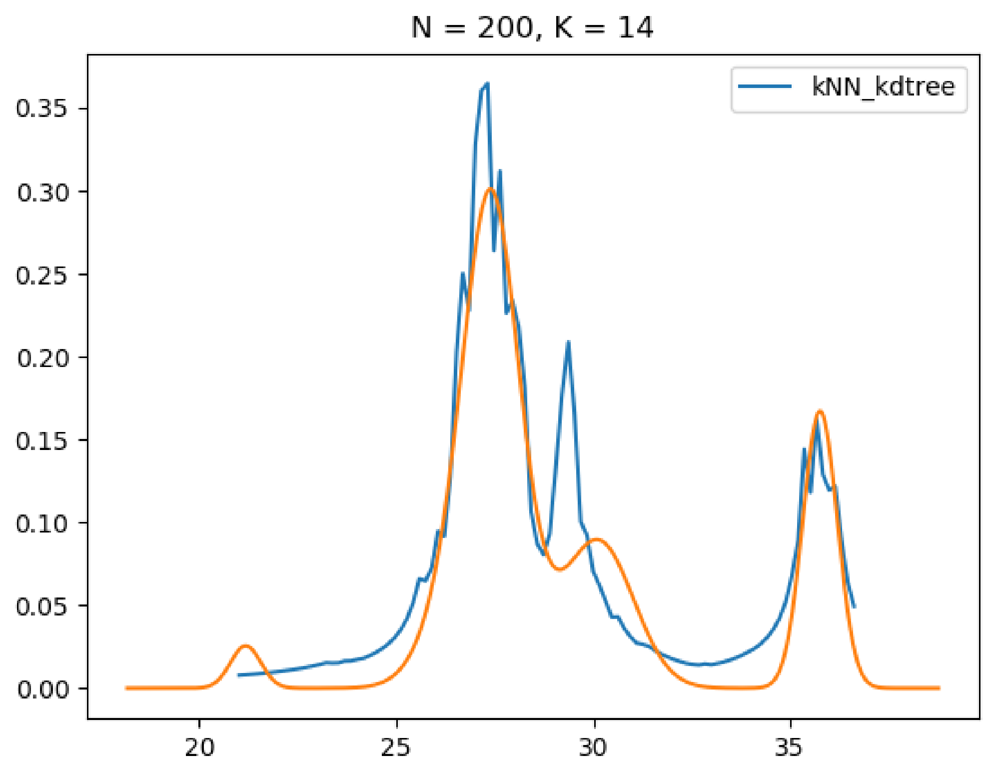
$$k = \sqrt{n}$$

Usage

- The following usage finds the best k with a given n:

```
python source.py --algorithm optimal_K --n 200
```

Result



If n = 100, k = 14 is the best result, which is shown above.

3.6

Assume {x1, x2, … xn} is the sample data, if the test data x > xn, the sum of probability mass over all the space won't converge to 1 as this is O(lnx).