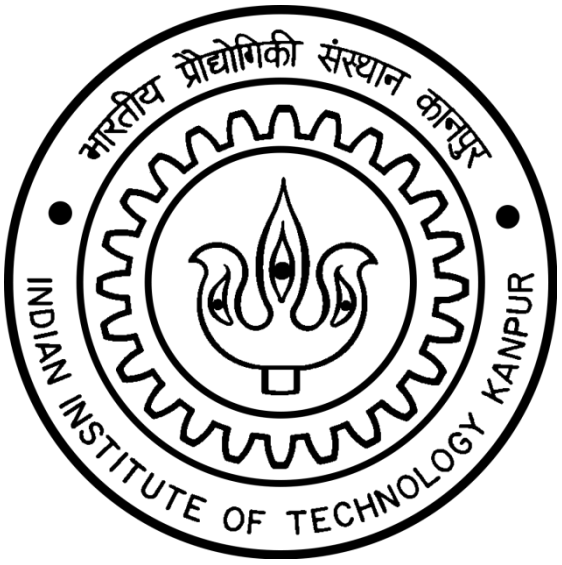


MTH 393A Project Report

Name: Aditya Jain

Roll no: 160047

Mentor: Prof. (Dr.) Subhra Sankar Dhar

Project Title:

Quantiles in Non-parametric Regression (Local constant estimators)

Declaration

I hereby declare that the work presented in the project report entitled "Quantiles in Non-parametric Regression (Local Constant Estimators)" contains my own ideas in my own words. At places, where ideas and words are borrowed from other sources, proper references, as applicable, have been cited. To the best of my knowledge this work does not emanate or resemble to other work created by person(s) other than mentioned herein.

The work was created on Thursday of April 4, 2019.

Name and Signature Name and Signature

Date:

Contents

1 Abstract

In this project, we study two local constant estimators for non-parametric regression namely Nadaraya Watson estimator and Local Constant Quantile estimator. The main aim of the project is to do empirical comparison of the two methods on some practical data set.

Key Words : Local constant estimators, Conditional Quantile, Kernel Smoothing

2 Introduction

Some Definitions:[3]

Quantiles:

Let Y be a real valued random variable. $F_Y(y)$ = P[Y<=y] be the CDF.

Then, τ^{th} quantile of Y is : $Q_Y(\tau) = \inf\{y : F_Y(y) >= \tau\}$; $\tau \in (0, 1)$

Define the loss function as : $\rho_\tau(y)$ = y (τ - $I_{(y<0)}$) ; I is indicator function

We can find $Q_Y(\tau)$ by *minimizing the expected loss of Y-u wrt u.* Let q_τ be the τ^{th} quantile of Y.

Then, $\min_u E(\rho_\tau(Y-u))$ to obtain $F_Y(q_\tau) = \tau$.

Conditional Quantiles:

Suppose the τ^{th} conditional quantile function is $Q_{Y|X}(\tau) = X\beta_\tau$.

Given the distribution function of Y, β_τ can be obtained as :

$\beta_\tau = argmin_{\beta \in R^k} E(\rho_\tau(Y - X\beta))$

Quantile Regression:

It is a type of regression analysis commonly used in statistics and econometrics. It *estimates conditional quantiles* of the response variable.

Applications of Quantile Regression :

1. When Conditional Quantile functions are of interest.
2. Assumptions of linear regression (OLS) are NOT applicable.
3. Dependent variable is bi-modal or multi-modal.
4. Dependent variable is highly skewed.
5. Quantile Regression estimates are robust against outliers.

Let $(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)$ be n data points.

Suppose the data is regarded as realizations from the model :

$Y_i = m(X_i) + \epsilon_i$; i = 1(1)n

$m(x)$ is the unknown regression function (mean function) to be estimated. $\epsilon_i's$ are the observation errors.

The aim of regression analysis is to estimate and analyse the unknown regression function $m(X)$ and reduce the observational errors to concentrate on mean dependence of Y on X. This curve approximation procedure is called **"smoothing"**[2].

Approximation of mean function can be done in 2 ways:[2]

1) **Parametric approach** :

- Assumes a pre-specified functional form for $m(X)$
- Fully described by a finite set of parameters.
- E.g. - Polynomial Regression

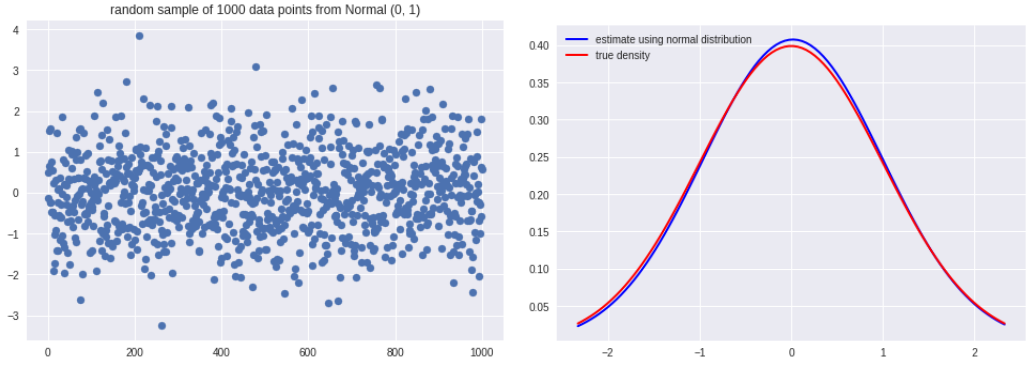
2) **Non-parametric approach** :

- No reference to a specific form.
- Pre-selected parametric model may be too restricted or too low-dimensional to fit unexpected features.
- Flexible tool in analyzing unknown regression relationships.
- Main purposes:
  - Explore general relationship between 2 variables.
  - Finding spurious or outlier observations.
  - Flexible method to substitute missing values.

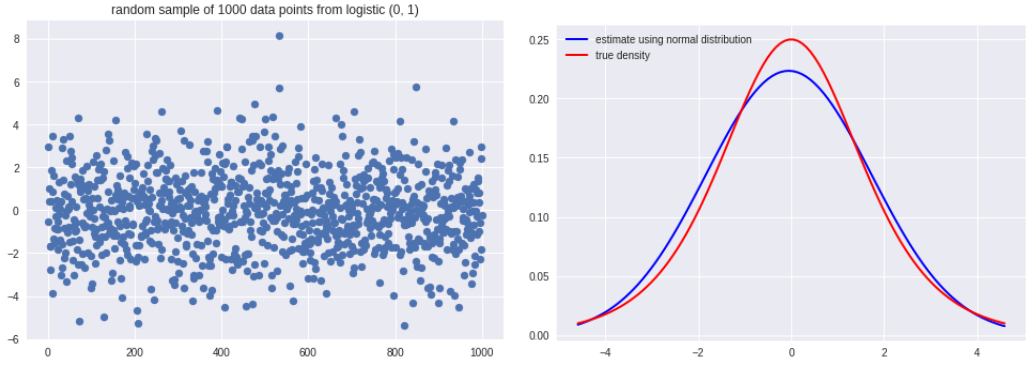
Note that there are some cases where these parametric techniques work well and some cases where they don't. Let us consider three examples to understand this.

In each example, we take a random sample of 1000 data points from some distribution, estimate the density of population using normal distribution with mean and variance same as the sample data and then compare it with the true density. [5]

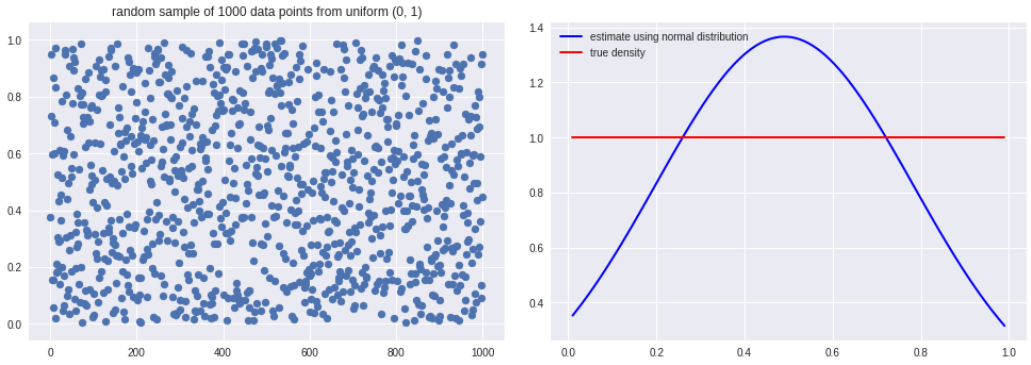
Case I: (parametric technique works very well)



Case II: (parametric technique works satisfactorily)



Case III: (parametric technique doesn't work well)



Non-parametric models[5] :

- Histogram

  Let h = bin-width ; N = sample-size

  Let $\hat{f}_{hist}(x)$ be the estimate of the density function 'f'.

  Then, $\hat{f}_{hist}(x) = \frac{1}{\text{N}} \frac{\text{No. of } x_i's \text{ in same bin as x}}{\text{width of bin containing x}}$

  $= \frac{1}{\text{Nh}} \sum_{i=1}^{N} \mathbf{1}\{x_i \in [x - h/2, x + h/2]\}$

- Kernel Density Estimator

  Note that histogram assumes constant density for all $x_i's$ in the bin containing x. Let $x_1, x_2, ..., x_n$ be a random sample from a population of density 'f'.

  Let $\hat{f}_{kernel}(x)$ be the estimate of the density function 'f'.

  Then, $\hat{f}_{kernel}(x) = \frac{1}{\text{Nh}} \sum_{i=1}^{N} K(\frac{x - x_i}{h})$

  'K' is the chosen kernel function which assigns weight to each x_i based on its distance from x.

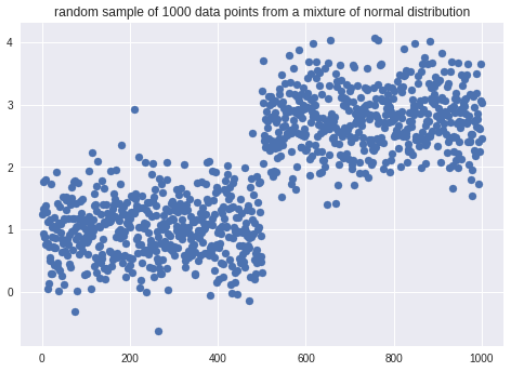
  'h' = bandwidth (measure of how far away from 'x' are we going to consider our local neighborhood)

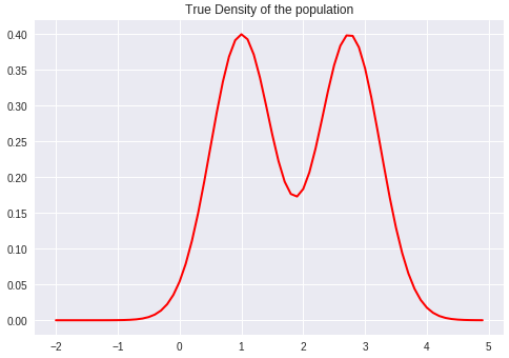
  - Higher the value of 'h', the more we think that data way from 'x' matters leading to underfitting (high bias, smoother fit).
  - Lower the value of 'h', the more we think that only local data matters leading to overfitting (high variance, noisier fit).

  So, there is a bias-variance trade-off to choose the optimal value of bandwidth.

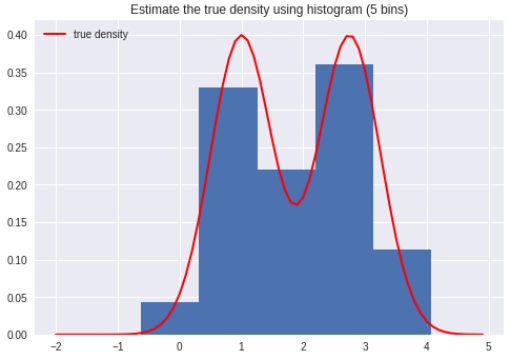
  Let us consider some examples to show the effect of variation of bandwidth on the non-parametric estimates.[5]

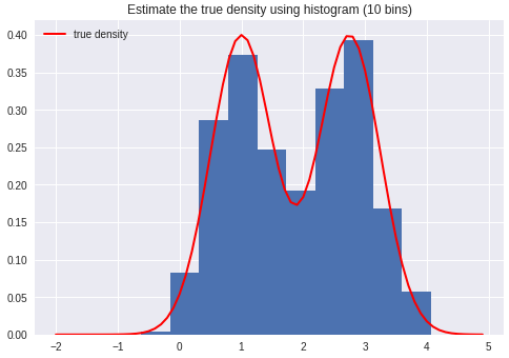
  We take a random sample of 1000 data points from a mixture of normal distribution and try to estimate it using two popular non-parametric techniques (histogram and kernel density estimators).

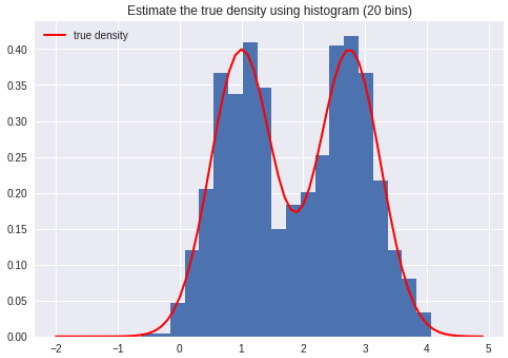

random sample of 1000 data points from a mixture of normal distribution

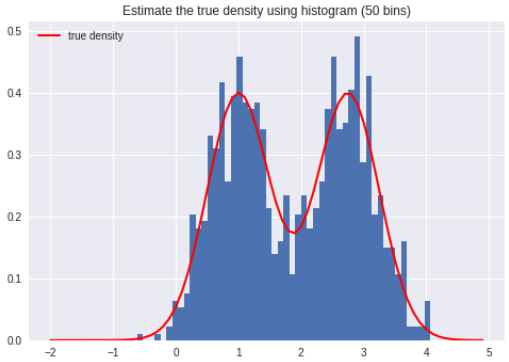

True Density of the population

– Histogram

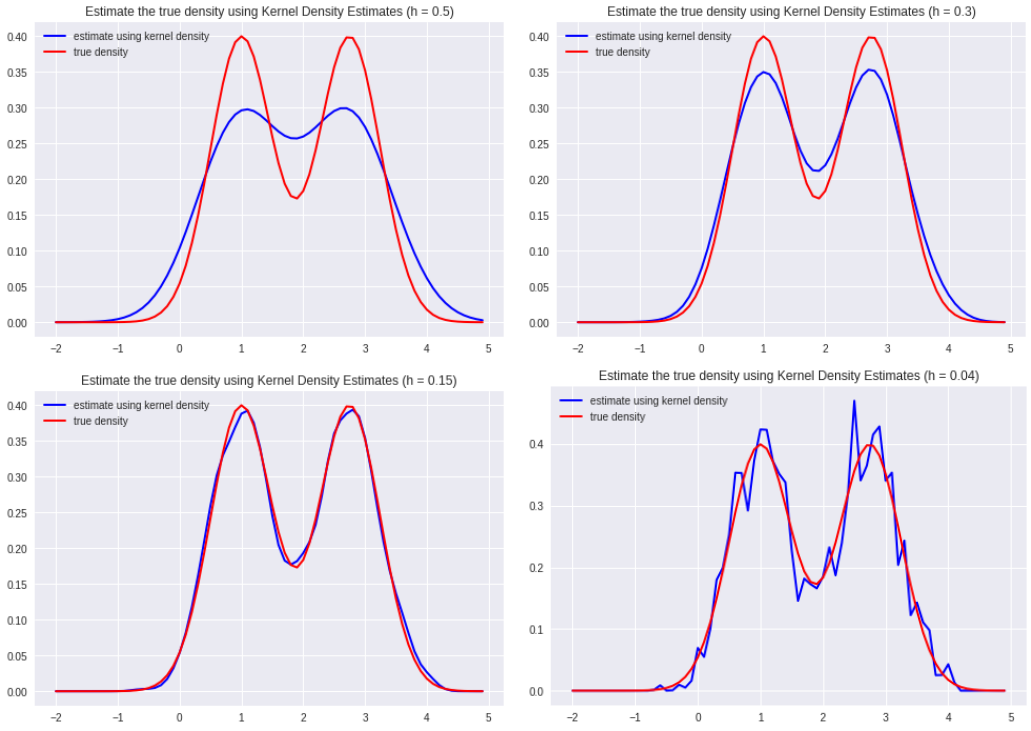

Estimate the true density using histogram (5 bins)
true density


Estimate the true density using histogram (10 bins)
true density


Estimate the true density using histogram (20 bins)
true density


Estimate the true density using histogram (50 bins)
true density

– Kernel Density Estimate (Gaussian kernel)



3 Main Body

Statistical Framework for smoothing problem [1]:

Let $(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)$ be n data points assumed to be iid observations (unless stated otherwise).

We wish to estimate the regression function $m(X)$ or its derivatives.

The basic idea of smoothing is : if $m(X)$ is smooth, it should be possible to use a *local average of the data near* x to construct an estimator of $m(X)$.

The generalized version of local modelling principle can be written as :

$$\hat{\theta}(X) = argmin_\theta \ \frac{1}{n} \sum_{i=1}^{n} l_i(X_i, Y_i, \theta) \ K(\frac{Xi - X}{h_n})$$

where θ is the set of parameters that need to be estimated by minimizing a loss function given in the form of the summation.

$l_i(X_i, Y_i, \theta)$ determines the type of loss function.

In this project, we will restrict ourselves to **Local Constant Estimators** and explore two types of them using computational approach.

- Nadaraya Watson Estimator
- Local Constant Quantile estimator

Nadaraya Watson Estimator

$$\hat{\theta}_{const}(X) = argmin_\theta \ \frac{1}{n} \sum_{i=1}^{n} (Y_i - \theta)^2 \ K(\frac{Xi - X}{h_n})$$

Here, $l_i(X_i, Y_i, \theta)$ is of the form $\rho(Y_i - \theta)$ where $\rho(z) = z^2$

Local Constant Quantile estimator

$$\hat{\theta}_{\alpha,const}(X) = argmin_\theta \ \frac{1}{n} \sum_{i=1}^{n} \left(|Y_i - \theta| + (2\alpha - 1)(Y_i - \theta)\right) \ K(\frac{Xi - X}{h_n}) \ ; \alpha \in (0, 1)$$

Here, $l_i(X_i, Y_i, \theta)$ is of the form $\rho(Y_i - \theta)$ where $\rho(z) = |z| + (2\alpha - 1)z$

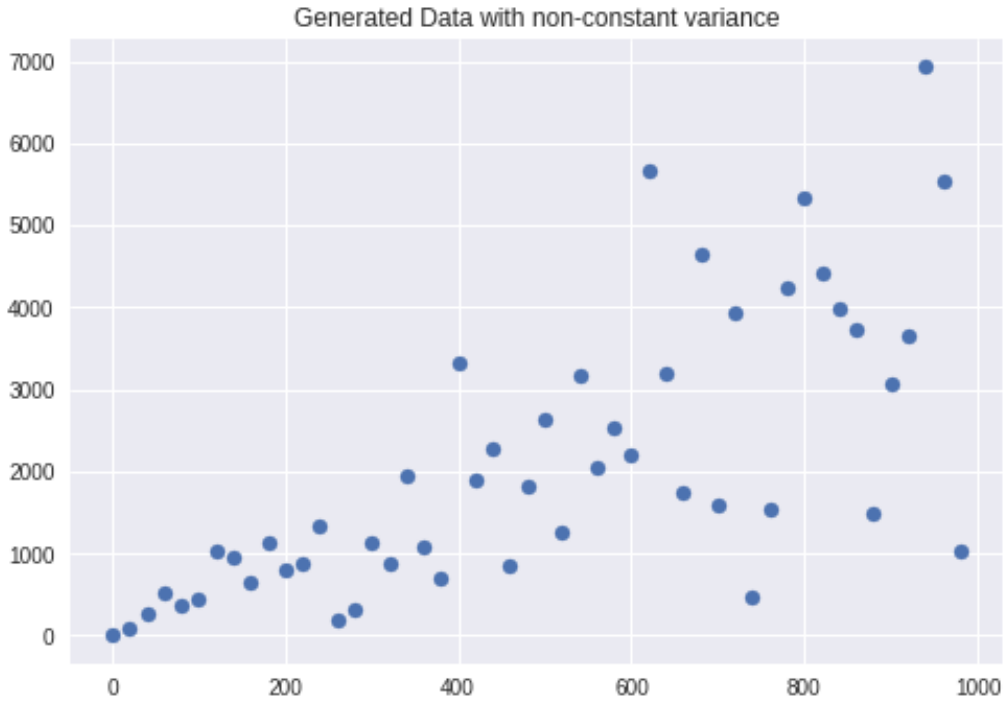
Note that the object function $m_l(x)$ depends on the loss function l(.).[1]

In Nadaraya Watson estimator, the loss function is $\rho(z) = z^2$ reducing it to the weighted least squares problem, estimating the conditional mean function.

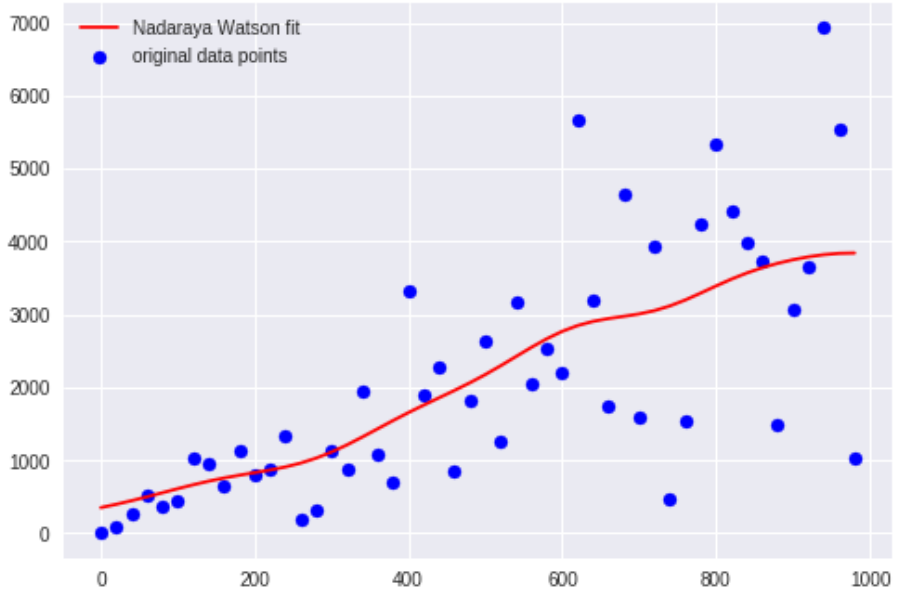
In local constant quantile estimator, the loss function is $\rho(z) = |z| + (2\alpha - 1)z$ estimating the conditional quantile function.

This is an example of a *Robust Regression Technique*[1] which is particularly useful when the variance of the error distribution is quite large or there is presence of outliers. We replace the least squares criterion by a robust method (*Quantile Regression* here).

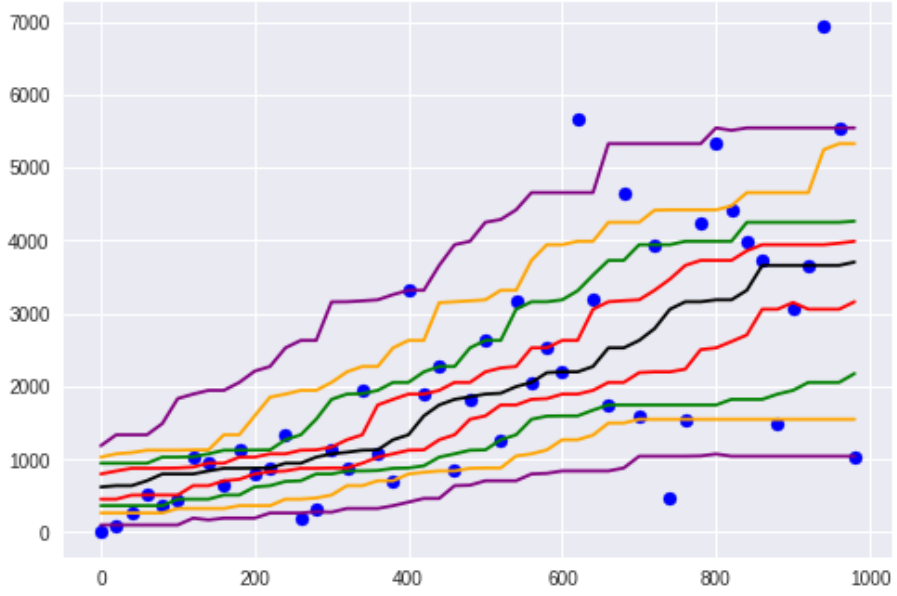
Let us compare the two local constant estimators using data with non-constant variance.



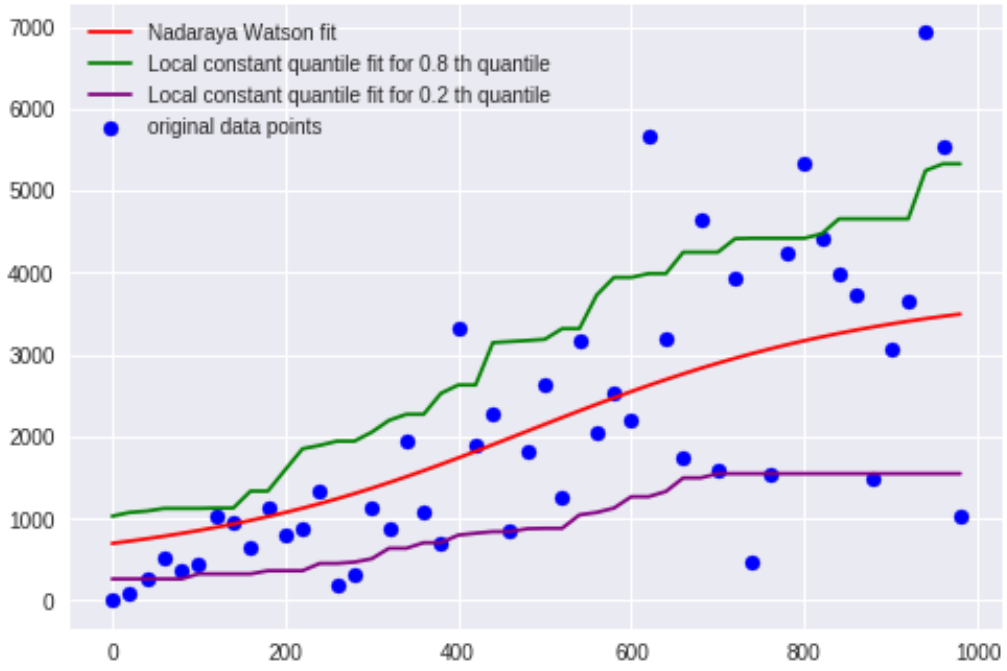
- Nadaraya Watson Fit



- Local Constant Quantile Estimator's Fit ($\alpha = 0.1, 0.2, \ldots, 0.9$)



The figure below compares the two methods.



4 Conclusion and Future Work

Conclusion:

Note that the Nadaraya Watson fit estimates the conditional mean function which doesn't tell us much about the relationship between x and y, especially as x gets larger. The mean of the response variable given 'x' becomes lesser meaningful as 'x' increases due to the large variance of 'x'.

However, the local constant quantile estimators corresponding to $\alpha = 0.2$ and $\alpha = 0.8$ are able to capture the association of 'y' with 'x' despite increasing variability. The distance between the two fits ('green' and 'purple') keeps on increasing as 'x' increases which reveals about increasing variance of the data. It can also help us to model different areas of the data appropriately.

Future Work :

We can extend the local constant idea to local linear estimators or in general local polynomial fitting for quantile regression and make an empirical comparison between them.

Python Codes

- **parametric_techniques_examples.ipynb**

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, logistic, uniform

# let's take a random sample of 1000 data points from N(0, 1) and plot them
np.random.seed(42) # seed is used to make the result reproducible

sz = 1000

x = norm.rvs(loc = 0, scale = 1, size=sz)
index_x = np.linspace(1, sz, sz)
plt.scatter(index_x, x)
plt.title('random sample of 1000 data points from Normal (0, 1)')

mean_x = np.mean(x)
sd_x = np.std(x)

print("mean of sample: ", mean_x)
print("standard deviation of sample: ", sd_x)

# let us try to recover the true density using parametric technique
# This is an example of a case where parametric technique works very well

fig, ax = plt.subplots(1,1)

x_test = np.linspace(norm.ppf(0.01), norm.ppf(0.99), 100)

# calculate the estimate of density of x over x_test
ax.plot(x_test, norm.pdf(x_test, loc=mean_x, scale=sd_x), 'k-', lw=2, color = 'blue', label = 'estimate using normal
distribution')

# Then, compare this estimate with the true density
ax.plot(x_test, norm.pdf(x_test, loc = 0, scale = 1), 'k-', lw=2, color = 'red', label = 'true density')

ax.legend(loc=2, frameon=False)

plt.show()

# let's take a random sample of 1000 data points from logistic (0, 1) and plot them
```

```
np.random.seed(42) # seed is used to make the result reproducible

sz2 = 1000
pi = 3.1416

x2 = logistic.rvs(loc = 0, scale = 1, size=sz2) # Note: for logisitic distribution, variance  = (scale**2)*(pi**2)/3
index_x2 = np.linspace(1, sz2, sz2)
plt.scatter(index_x2, x2)
plt.title('random sample of 1000 data points from logistic (0, 1)')

mean_x2 = np.mean(x2)
sd_x2 = np.std(x2)

print("mean of sample: ", mean_x2)
print("standard deviation of sample: ", sd_x2)

# let us try to recover the true density using parametric technique
# This is an example of a case where parametric technique works satisfactorily well

fig, ax = plt.subplots(1,1)

x_test2 = np.linspace(logistic.ppf(0.01), logistic.ppf(0.99), 100)

# calculate the estimate of density of x2 over x_test2
ax.plot(x_test2, norm.pdf(x_test2, loc=mean_x2, scale=sd_x2), 'k-', lw=2, color = 'blue', label = 'estimate using
normal distribution')

# Then, compare this estimate with the true density
ax.plot(x_test2, logistic.pdf(x_test2, loc = 0, scale = 1), 'k-', lw=2, color = 'red', label = 'true density')

ax.legend(loc=2, frameon=False)

plt.show()

# let's take a random sample of 1000 data points from uniform (0, 1) and plot them
np.random.seed(42) # seed is used to make the result reproducible

sz3 = 1000

x3 = uniform.rvs(loc = 0, scale = 1, size=sz3)
index_x3 = np.linspace(1, sz3, sz3)
plt.scatter(index_x3, x3)
plt.title('random sample of 1000 data points from uniform (0, 1)')

mean_x3 = np.mean(x3)
```

```
sd_x3 = np.std(x3)

print("mean of sample: ", mean_x3)
print("standard deviation of sample: ", sd_x3)

# let us try to recover the true density using parametric technique
# This is an example of a case where parametric technique works very bad

fig, ax = plt.subplots(1,1)

x_test3 = np.linspace(uniform.ppf(0.01), uniform.ppf(0.99), 100)

# calculate the estimate of density of x2 over x_test2
ax.plot(x_test3, norm.pdf(x_test3, loc=mean_x3, scale=sd_x3), 'k-', lw=2, color = 'blue', label = 'estimate using
normal distribution')

# Then, compare this estimate with the true density
ax.plot(x_test3, uniform.pdf(x_test3, loc = 0, scale = 1), 'k-', lw=2, color = 'red', label = 'true density')

ax.legend(loc=2, frameon=False)

plt.show()
```

- **choice_of_bandwidth_value.ipynb**

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, gaussian_kde

np.random.seed(42) # seed is used to make the result reproducible
sz = 500

x1 = norm.rvs(loc = 1, scale = 0.5, size=sz)
x2 = norm.rvs(loc = 2.75, scale = 0.5, size=sz)

x = np.concatenate((x1, x2)) # mixture of normal distributions
index_x = np.linspace(1, 2*sz, 2*sz)
plt.scatter(index_x, x)
plt.title('random sample of 1000 data points from a mixture of normal distribution')

mean_x = np.mean(x)
sd_x = np.std(x)

print("mean of sample: ", mean_x)
print("standar deviation of sample: ", sd_x)
```

```
fig, ax = plt.subplots(1,1)

x_test = np.arange(-2, 5, 0.1)

# the true density
ax.plot(x_test, 0.5*norm.pdf(x_test, loc = 1, scale = 0.5) + 0.5*norm.pdf(x_test, loc = 2.75, scale = 0.5), 'k-', lw=2,
    color = 'red')
plt.title('True Density of the population')
plt.show()

# Let's try to estimate the true density using non-parametric techniques and try to visualize the double hump
    pattern

# 1. Histograms

fig, ax = plt.subplots(1,1)
ax.hist(x, bins = 5, density = True) # can't see the pattern easily
ax.plot(x_test, 0.5*norm.pdf(x_test, loc = 1, scale = 0.5) + 0.5*norm.pdf(x_test, loc = 2.75, scale = 0.5), 'k-', lw=2,
    color = 'red', label = 'true density')
ax.legend(loc=2, frameon=False)
plt.title('Estimate the true density using histogram (5 bins)')
plt.show()

fig, ax = plt.subplots(1,1)
ax.hist(x, bins = 10, density = True) # can see the pattern now
ax.plot(x_test, 0.5*norm.pdf(x_test, loc = 1, scale = 0.5) + 0.5*norm.pdf(x_test, loc = 2.75, scale = 0.5), 'k-', lw=2,
    color = 'red', label = 'true density')
ax.legend(loc=2, frameon=False)
plt.title('Estimate the true density using histogram (10 bins)')
plt.show()

fig, ax = plt.subplots(1,1)
ax.hist(x, bins = 20, density = True) # The pattern is becoming clear!!
ax.plot(x_test, 0.5*norm.pdf(x_test, loc = 1, scale = 0.5) + 0.5*norm.pdf(x_test, loc = 2.75, scale = 0.5), 'k-', lw=2,
    color = 'red', label = 'true density')
ax.legend(loc=2, frameon=False)
plt.title('Estimate the true density using histogram (20 bins)')
plt.show()

fig, ax = plt.subplots(1,1)
ax.hist(x, bins = 50, density = True) # The pattern is very clear. But, it also starts to get noisier
ax.plot(x_test, 0.5*norm.pdf(x_test, loc = 1, scale = 0.5) + 0.5*norm.pdf(x_test, loc = 2.75, scale = 0.5), 'k-', lw=2,
    color = 'red', label = 'true density')
ax.legend(loc=2, frameon=False)
plt.title('Estimate the true density using histogram (50 bins)')
plt.show()

# 2. Kernel Density Estimates
fig, ax = plt.subplots(1,1)
kde = gaussian_kde(dataset = x, bw_method = 0.5) # too high bandwidth
```

```
ax.plot(x_test, kde.pdf(x_test), 'k-', lw=2, color = 'blue', label = 'estimate using kernel density')
ax.plot(x_test, 0.5*norm.pdf(x_test, loc = 1, scale = 0.5) + 0.5*norm.pdf(x_test, loc = 2.75, scale = 0.5), 'k-', lw=2,
    color = 'red', label = 'true density')
ax.legend(loc=2, frameon=False)
plt.title('Estimate the true density using Kernel Density Estimates (h = 0.5)')
plt.show()

fig, ax = plt.subplots(1,1)
kde = gaussian_kde(dataset = x, bw_method = 0.3) # slightly better

ax.plot(x_test, kde.pdf(x_test), 'k-', lw=2, color = 'blue', label = 'estimate using kernel density')
ax.plot(x_test, 0.5*norm.pdf(x_test, loc = 1, scale = 0.5) + 0.5*norm.pdf(x_test, loc = 2.75, scale = 0.5), 'k-', lw=2,
    color = 'red', label = 'true density')
ax.legend(loc=2, frameon=False)
plt.title('Estimate the true density using Kernel Density Estimates (h = 0.3)')
plt.show()

fig, ax = plt.subplots(1,1)
kde = gaussian_kde(dataset = x, bw_method = 0.15) # looks like optimal value !!

ax.plot(x_test, kde.pdf(x_test), 'k-', lw=2, color = 'blue', label = 'estimate using kernel density')
ax.plot(x_test, 0.5*norm.pdf(x_test, loc = 1, scale = 0.5) + 0.5*norm.pdf(x_test, loc = 2.75, scale = 0.5), 'k-', lw=2,
    scolor = 'red', label = 'true density')
ax.legend(loc=2, frameon=False)
plt.title('Estimate the true density using Kernel Density Estimates (h = 0.15)')
plt.show()

fig, ax = plt.subplots(1,1)
kde = gaussian_kde(dataset = x, bw_method = 0.04) # too low bandwidth (got noisier)

ax.plot(x_test, kde.pdf(x_test), 'k-', lw=2, color = 'blue', label = 'estimate using kernel density')
ax.plot(x_test, 0.5*norm.pdf(x_test, loc = 1, scale = 0.5) + 0.5*norm.pdf(x_test, loc = 2.75, scale = 0.5), 'k-', lw=2,
    color = 'red', label = 'true density')
ax.legend(loc=2, frameon=False)
plt.title('Estimate the true density using Kernel Density Estimates (h = 0.04)')
    plt.show()
```

- **local_constant_estimators.ipynb**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import norm, logistic, uniform
from statsmodels.nonparametric.kernel_regression import KernelReg

np.random.seed(42) # seed is used to make the result reproducible

rng_min = 0
rng_max = 1000
```

```
step = 20

x = np.arange(rng_min, rng_max, step) # independent variable
var = 0.1 + 5*x**2    # non-constant variance
b = 6                 # intercept
m = 5                 # slope

# normal random errors with non-constant variance
sz = (rng_max - rng_min)//step
err = norm.rvs(loc = 0, scale = np.sqrt(var), size=sz)

y = m*x + b + err

x_sub = x[0:len(x):5]
y_sub = y[0:len(y):5]

plt.scatter(x, y)
plt.title('Generated Data with non-constant variance')
print(np.mean(x), np.mean(y))

"""But as x increases, the mean of y given x becomes less meaningful. As a predictor of y it's pretty useless.

But what about a quantile, like the 0.90 quantile?
"""

# 1) NW estimator

NW_model = KernelReg(endog = y, exog = x, var_type = 'c', reg_type = 'lc')
#NW_model = KernelReg(endog = y_sub, exog = x_sub, var_type = 'c', reg_type = 'lc')

fig, ax = plt.subplots(1,1)

ax.scatter(x, y, color = 'blue', label = 'original data points')
ax.plot(x, NW_model.fit()[0], color = 'red', label = 'Nadaraya Watson fit (bw using cross-validation)')
#ax.plot(x_sub, NW_model.fit()[0], color = 'red', label = 'Nadaraya Watson fit (bw using cross-validation)')

ax.legend(loc=2, frameon=False)

# 2) NW estimator (choose bw manually)

import math

def gaus_ker(x):
  return (1/np.sqrt(2*math.pi))*np.exp(-0.5*(x**2))
```

```
def NW_pred(x, y, data_xpred, bw):

  ker_x = np.zeros(len(x))

  for i in range(len(x)):
    ker_x[i] = gaus_ker((x[i]-data_xpred)/bw)

  G_numer = (ker_x * y).sum(axis=0)
  G_denom = ker_x.sum(axis=0)
  return G_numer / G_denom

def my_NW_model(x, y, bw):
  my_pred = np.zeros(len(x))
  for i in range(len(x)):
    my_pred[i] = NW_pred(x, y, x[i], bw)

  return my_pred

bw = 200

fig, ax = plt.subplots(1,1)

ax.scatter(x, y, color = 'blue', label = 'original data points')
ax.plot(x, my_NW_model(x, y, bw=80), color = 'red', label = 'Nadaraya Watson fit')
#ax.plot(x_sub, my_NW_model(x_sub, y_sub, bw=bw), color = 'red', label = 'Nadaraya Watson fit (choose bw manually)')
ax.legend(loc=2, frameon=False)

"""Ideally, the fit should have gone upwards gradually, but we observe that towards the end, it bent slightly downwards because of the presence of outliers. Thus, it is NOT able to capture the trend of the data appropriately."""

from scipy.optimize import minimize

def sum_q_est(x, y, data_xpred, theta, alpha, bw):
  sum = 0
  for i in range(len(x)):
    sum+=(abs(y[i]-theta) + (2*alpha-1)*(y[i]-theta))*gaus_ker((x[i]-data_xpred)/bw)

  return sum

alpha = np.arange(0.1, 1, 0.1)
y_pred = np.zeros((len(alpha), len(x)))
#y_pred = np.zeros((len(alpha), len(x_sub)))
```

```
bw = 200

for j in range(len(alpha)):
  for i in range(len(x)):
  #for i in range(len(x_sub)):
    fun = lambda theta: sum_q_est(x, y, x[i], theta, alpha[j], bw)
    #fun = lambda theta: sum_q_est(x_sub, y_sub, x_sub[i], theta, alpha[j], bw)
    y_pred[j][i] = minimize(fun, 10).x

fig, ax = plt.subplots(1,1)

ax.scatter(x, y, color = 'blue', label = 'original data points')

ax.plot(x, y_pred[0], color = 'purple', label = 'Local constant quantile fit for 0.1 th quantile')
ax.plot(x, y_pred[1], color = 'orange', label = 'Local constant quantile fit for 0.2 th quantile')
ax.plot(x, y_pred[2], color = 'green', label = 'Local constant quantile fit for 0.3 th quantile')
ax.plot(x, y_pred[3], color = 'red', label = 'Local constant quantile fit for 0.4 th quantile')
ax.plot(x, y_pred[4], color = 'black', label = 'Local constant quantile fit for 0.5 th quantile')
ax.plot(x, y_pred[5], color = 'red', label = 'Local constant quantile fit for 0.6 th quantile')
ax.plot(x, y_pred[6], color = 'green', label = 'Local constant quantile fit for 0.7 th quantile')
ax.plot(x, y_pred[7], color = 'orange', label = 'Local constant quantile fit for 0.8 th quantile')
ax.plot(x, y_pred[8], color = 'purple', label = 'Local constant quantile fit for 0.9 th quantile')

"""
ax.plot(x_sub, y_pred[0], color = 'purple', label = 'Local constant quantile fit for 0.1 th quantile')
ax.plot(x_sub, y_pred[1], color = 'orange', label = 'Local constant quantile fit for 0.2 th quantile')
ax.plot(x_sub, y_pred[2], color = 'green', label = 'Local constant quantile fit for 0.3 th quantile')
ax.plot(x_sub, y_pred[3], color = 'red', label = 'Local constant quantile fit for 0.4 th quantile')
ax.plot(x_sub, y_pred[4], color = 'black', label = 'Local constant quantile fit for 0.5 th quantile')
ax.plot(x_sub, y_pred[5], color = 'red', label = 'Local constant quantile fit for 0.6 th quantile')
ax.plot(x_sub, y_pred[6], color = 'green', label = 'Local constant quantile fit for 0.7 th quantile')
ax.plot(x_sub, y_pred[7], color = 'orange', label = 'Local constant quantile fit for 0.8 th quantile')
ax.plot(x_sub, y_pred[8], color = 'purple', label = 'Local constant quantile fit for 0.9 th quantile')
"""

fig, ax = plt.subplots(1,1)

ax.scatter(x, y, color = 'blue', label = 'original data points')

ax.plot(x, my_NW_model(x, y, bw=200), color = 'red', label = 'Nadaraya Watson fit')
ax.plot(x, y_pred[7], color = 'green', label = 'Local constant quantile fit for 0.8 th quantile')
ax.plot(x, y_pred[1], color = 'purple', label = 'Local constant quantile fit for 0.2 th quantile')

ax.legend(loc=2, frameon=False)
```

Bibliography

[1] Jianqing Fan, Irene Gijbels, *Local Polynomial Modelling and Its Applications (Monographs on Statistics and Applied Probability 66)*

[2] Wolfgang Härdle, *Applied Nonparametric Regression*

[3] https://en.wikipedia.org/wiki/Quantile_regression

[4] https://data.library.virginia.edu/getting-started-with-quantile-regression/

[5] https://www.youtube.com/watch?v=pAKwoz05lK4