Multiple Methods to Approximate Empirical Density Functions

Prof. Mjolsness
TA: Corey Scott Due Date: 12/13/18

Problem Statement

The objective of this project is to approximate uni-variate probability density functions of a given data set. Usually data has unknown probability density functions and it is very important to approximate these functions in order to generate data from these distributions. Although this project focuses on 1-dimensional data, some of these methods would work in higher dimensions. We used 4 different methods of approximating density functions: Interpolation, Polynomial Least Squares Approximation, Kernel Density Estimation, and Generative Adversarial Network. The idea is to generate data from known density functions, train our methods with a the four methods, and then compare which method(s) did the best, how well it approximated the density function, using K-L divergence. We worked with the following three distributions:

Normal($\mu = 3$, $\sigma^2 = 4$):

$$f(x; 3, 4) = \frac{1}{\sqrt{8\pi}} e^{-\frac{(x-3)^2}{8}}$$

Bimodal: $\frac{1}{2}$(Normal($\mu = -2$, $\sigma^2 = 1$) + Normal($\mu = 2$, $\sigma^2 = 2$):

$$f(x) = \frac{1}{2}\left(\frac{1}{\sqrt{2\pi}} e^{-\frac{(x+2)^2}{1}} + \frac{1}{\sqrt{8\pi}} e^{-\frac{(x-2)^2}{8}}\right)$$

Exponentially Modified Normal with $\lambda = 0.1$, $\mu = 0$ and $\sigma = 2$:

$$f(x; \mu, \sigma, \lambda) = \frac{\lambda}{2} e^{\frac{\lambda}{2}(2\mu + \lambda\sigma^2 - 2x)} erfc\Big(\frac{\mu + \lambda\sigma^2 - x}{\sqrt{2}\sigma}\Big) = 0.05 e^{0.02 - 0.1x} erfc\left(\frac{0.4 - x}{2\sqrt{2}}\right)$$

where *erfc* is the complementary error function defined as

$$erfc(x) = 1 - erf(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$$

Previous Work

We each did a different method and there has been previous work done for each method. The previous work is therefore explain under each method. Overall, there has been a lot of research in this area. Barron and Sheu used maximum likelihood in sequences of exponential families [2] and there has been work done using orthogonal polynomials [1]. These are just a few examples of methods to approximate density functions.

Methods

1 Interpolation/Histosplines and Polynomial Least Squares Approximation

By: Corey Katz

Setup of Points and Function Values:

Our simplest (or most naive) method for approximating probability density functions, was the use of interpolation, more specifically the use of Cubic Splines, or the use of Least Squares Polynomial Approximation. Before we discuss the two methods of line-fitting, we must first set up the points needed for both methods. To determine the points at which to interpolate or approximate with a polynomial, the use of histogram information is a key aspect of this method.

An important aspect of the histosplines or fitting a polynomial to the data is that they preserve the area in the histogram [4]. In this case we would like to have a function that has area beneath it equal to 1, due to the fact that this is a probability density function. The following is the set up for the points at which we will interpolate and there corresponding function evaluations:

$$\vec{x} = [x_0, ..., x_{n+1}] \text{ where } x_i \text{ are midpoints of the histogram bins,}$$
$$x_0 \, and \, x_{n+1} \text{ are the midpoints of "bins" past the min and max of the data.}$$
$$\vec{y} = [y_0, ..., y_{n+1}] \text{ where } y_i \text{ is the proportion of the data in each bin } (x_i, x_{i+1})$$
$$\text{Note: } y_0 = 0 \text{ and } y_{n+1} = 0$$

The number of bins determines the number of points we will fit a function to and therefore is a hyper-parameter. Another parameter that shows up in Least Squares fitting is the degree of polynomial. In our code we try out different numbers of bins and polynomial degrees to find the best fitting curve. The bins are otherwise arbitrary and and only depend on the data [4].

Constraints of Our Functions:

There are a few conditions that need to be met given we are trying to approximate a probability density function:

1. f(x) is non-negative for all x
2. the area under the curve must be 1
3. We want a smooth function (not necessary but we want this property)

With this in mind a challenge, I had was making sure that the function was non-negative. This problem occurs at the tails of the data because the frequencies are very close to zero and the splines are forced to go through every point. This is a big problem because density functions can not be negative, otherwise it would not be an issue. In a paper called, "A Local Algorithm for Constructing Non-negative Cubic Splines," the

authors propose running the cubic splines and if there are any intervals where the function is negative, we should run cubic splines in that region with a finer grid [5]. As for the polynomial approximation, this problem does arise as well.

The cubic spline method discussed above was very difficult to implement and therefore I found my own creative way of interpolating(and polynomial fitting) to get a non-negative function. I took the square-roots of all of the $y_i's$, interpolated or polynomial approximated, and then squared my outputted function. This seemed to do the trick in fixing the non-negative issue and it was also much easier to code.

Cubic Splines:

The idea of cubic splines, and interpolation in general, is to find a piece-wise or smooth function that goes through every point. Note that no points can have the same x-value, otherwise we can not get a function. Cubic Splines are the most common piece-wise-polynomial approximation method[Burden]. The cubic splines are continuously differentiable on the intervals(there is a cubic function between each point) and they have continuous second derivatives [Burden]. This process uses a $(n+1) \times (n+1)$ system of equations to find the cubic spline coefficients. Within each interval, we need to have four unknowns because each spline should take the form $a_i x^3 + b_i x^2 + c_i x + d$[3]. This method does not use any optimization because we are solving a system of equations, but cubic spline is a method that has very high accuracy. Results of the cubic splines can be found in the "Results" section.

This is the system of equations will solve for $c_i's$: $A\vec{x} = \vec{b}$:

$$A = \begin{bmatrix} 1 & 0 & 0 & \dots & \dots & \dots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & & \ddots & \ddots & \vdots \\ \vdots & \ddots & & & & & 0 \\ \vdots & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \dots & \dots & & 0 & 0 & 1 \end{bmatrix}$$

$$\vec{b} = \begin{bmatrix} 0 \\ 3/h_1{*}(\mathrm{a}_2 - a_1) - 3/h_0{*}(\mathrm{a}_1 - a_0) \\ \vdots \\ 3/h_{n-1}{*}(\mathrm{a}_n - a_{n-1}) - 3/h_{n-2}{*}(\mathrm{a}_{n-1} - a_{n-2}) \\ 0 \end{bmatrix}$$

$$\vec{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

Note: h_i is the width of the interval. Once we know $a_i's$ and $c_i's$ we can get $b_i's$ and $d_i's$

Least Squares Polynomial Approximation:

The Least Squares Polynomial Approximation is very similar to the interpolation as it tries to fit the best polynomial (in this case it is not piece-wise and it can be of any degree polynomial). The difference between the two methods is that the Least Squares methods does not have to go through every point and it is generated by minimizing the distance between each point and the point of the function(residuals). The set up is as follows:

First note that the setup of the points to which we would like to fit a polynomial is the same as interpolation. We use the midpoints of the bins, so we are basically fitting the polynomial to the histogram of data.

We would like to get a polynomial of the form : $y = a_0 + a_1 x + ... + a_k x^k$. This is not a regular system of equations due to the fact that we need to solve for k+1 unknowns, but we (usually) have more data points to fit. So this is an over-determined system. So we find the $a_i's$ by minimizing the following function:

$$residual^2 = \sum_{i=1}^{n} (y_i - (a_0 + a_1 x_i + ... + a_k x_i^k))^2$$

This can be solved analytically using partial derivatives and setting them equal to zero. Now we have a system of equations that ban be solved using usual methods.

This can be written in matrix form: We want to solve $\vec{y} = X\vec{a}$. If we premultiply by X^t we are left with

$$\vec{a} = (X^t X)^{-1} X^t \vec{y}$$

where

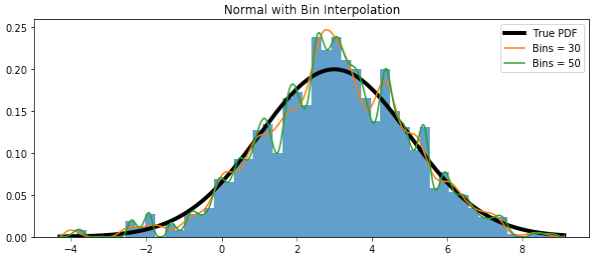
$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

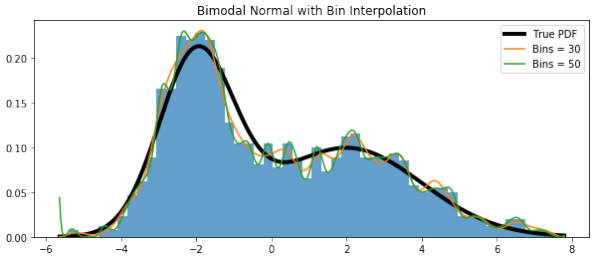
$$\vec{a} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix}$$

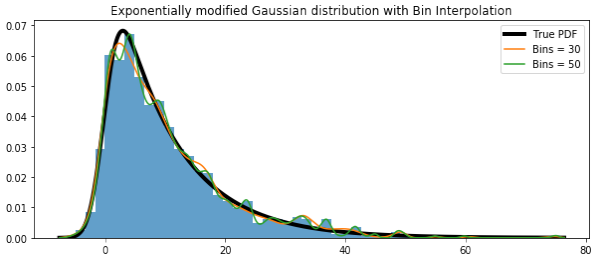
$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^k \\ 1 & x_2 & x_2^2 & \dots & x_2^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^k \end{bmatrix}$$

I found that lower degree polynomials are not well suited for a lot of data points that could potentially take a weird shape. I also had a hard time finding a way to make sure that the functions were non-negative. Ultimately, I used the same trick as in interpolation. I took the square roots of the y's, fit the polynomial, and then squared the polynomial. The results of how well this method did will be in the "Results" section.

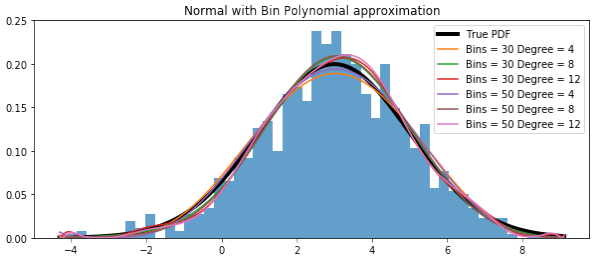
Plots for Cubic Spline Interpolation:

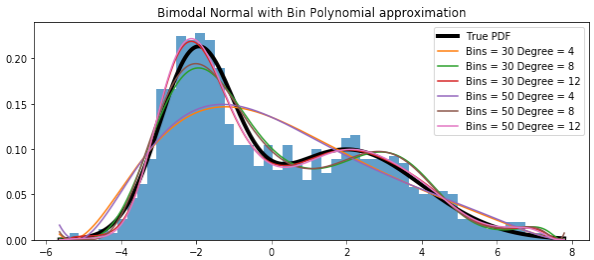


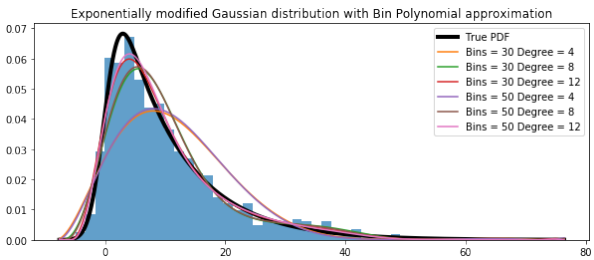




Plots for Least Squares Polynomial Approximation







Coding these Method and Difficulties(Coding and in General):

These methods were fairly straightforward to code; the numpy and scipy libraries have built-in functions to do cubic spline interpolation and polynomial regression. As mentioned above, there was some issue coding the functions in a way to achieve non-negative output functions. I was unable to code the suggested solution to this problem, but came up with a trick to fix the problem. There are some issues also with the function after the endpoints, but K-L divergence only looks at the data in the interval(or the function on the interval), so this did not pose a issue with coding. One problem with interpolation is that it may over-fit the data (or in this case fit to well to the histogram). This problem arises because interpolation is piece-wise and is forced through every point, so basically it picks up every grove in the histogram, this makes polynomial approximation somewhat better.

References

[1] Ralph Badinelli. Approximating probability density functions and their convolutions using orthogonal polynomials. *European Journal of Operational Research*, 95:211–230, 02 1996.

[2] Andrew R. Barron and Chyong-Hwa Sheu. Approximation of density functions by sequences of exponential families. *Ann. Statist.*, 19(3):1347–1369, 09 1991.

[3] Faires J. Douglas Burden, Richard L. and Annette L. Burden.

[4] Paolo Costantini and Francesca Pelosi. Shape preserving histogram approximation. *Advances in Computational Mathematics*, 26(1):205–230, Jan 2007.

[5] Bernd Fischer, Gerhard Opfer, and Madan Puri. A local algorithm for constructing non-negative cubic splines. *Journal of Approximation Theory*, 64:1–16, 01 1991.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[8] S. J. Sheather and M. C. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, 53(3):683–690, 1991.

[9] Jake VanderPlas. *Python Data Science Handbook*. 2016.