**Locally Weighted Linear Regression in Python**

Machine Learning from Scratch: Part 2

[Suraj Verma](https://medium.com/@rssp036?source=post_page-----3d324108efbf--------------------------------)

[Suraj Verma](https://medium.com/@rssp036?source=post_page-----3d324108efbf--------------------------------)

[Mar 18·7 min read](https://towardsdatascience.com/locally-weighted-linear-regression-in-python-3d324108efbf?source=post_page-----3d324108efbf--------------------------------)

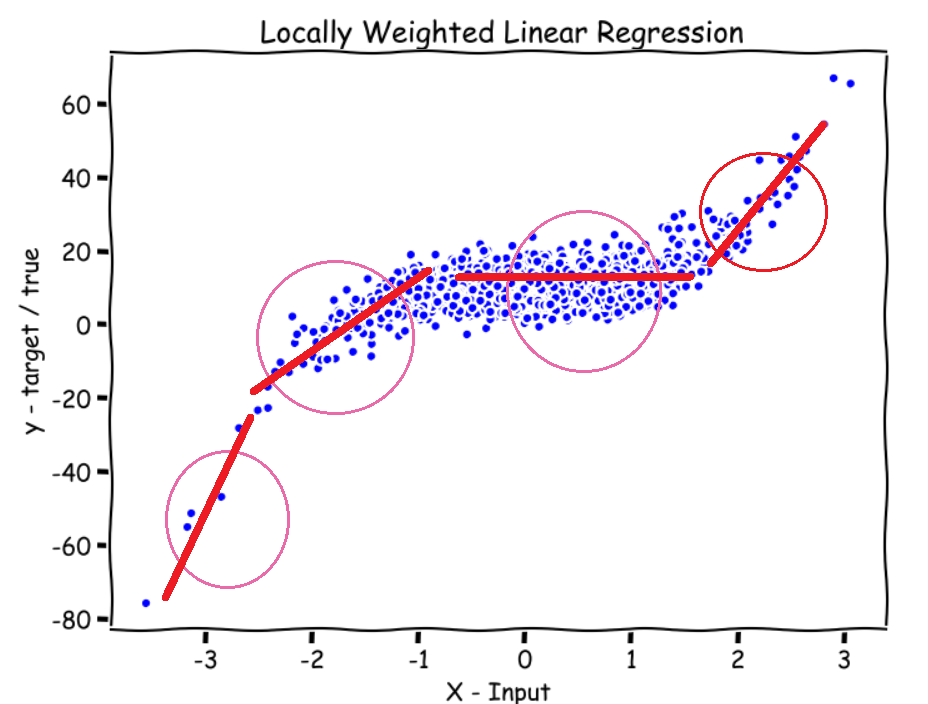


Image by Author

In this article, we will implement a Non-Parametric Learning Algorithm called the *Locally Weighted Linear Regression*. First, we will look at the difference between the *parametric and non-parametric* learning algorithms, followed by understanding the *weighting Function*, *predict function*, and finally *plotting* the predictions using Python NumPy and Matplotlib.

**[Linear Regression from scratch in Python](https://medium.com/analytics-vidhya/linear-regression-from-scratch-in-python-b6501f91c82d" \t "_blank)**

[Machine Learning from Scratch: Part 1](https://medium.com/analytics-vidhya/linear-regression-from-scratch-in-python-b6501f91c82d" \t "_blank)

[medium.com](https://medium.com/analytics-vidhya/linear-regression-from-scratch-in-python-b6501f91c82d" \t "_blank)

**[Normal Equation Using Python: The Closed-Form Solution for Linear Regression](https://towardsdatascience.com/normal-equation-in-python-the-closed-form-solution-for-linear-regression-13df33f9ad71" \t "_blank)**

[Machine Learning from scratch: Part 3](https://towardsdatascience.com/normal-equation-in-python-the-closed-form-solution-for-linear-regression-13df33f9ad71" \t "_blank)

[towardsdatascience.com](https://towardsdatascience.com/normal-equation-in-python-the-closed-form-solution-for-linear-regression-13df33f9ad71" \t "_blank)

**Machine Learning from scratch series —**

* Part **1:**[**Linear Regression from scratch in Python**](https://medium.com/analytics-vidhya/linear-regression-from-scratch-in-python-b6501f91c82d?source=your_stories_page-------------------------------------)
* Part **2:**[**Locally Weighted Linear Regression in Python**](https://towardsdatascience.com/locally-weighted-linear-regression-in-python-3d324108efbf?source=your_stories_page-------------------------------------)
* Part **3:**[**Normal Equation Using Python: The Closed-Form Solution for Linear Regression**](https://towardsdatascience.com/normal-equation-in-python-the-closed-form-solution-for-linear-regression-13df33f9ad71?source=your_stories_page-------------------------------------)
* Part **4:**[**Polynomial Regression From Scratch in Python**](https://towardsdatascience.com/polynomial-regression-in-python-b69ab7df6105)
* Part **5**: [**Logistic Regression From Scratch in Python**](https://medium.com/p/ec66603592e2?source=post_stats_page-------------------------------------)

Parametric vs Non-Parametric Learning Algorithms

**Parametric** — In a Parametric Algorithm, we have a fixed set of parameters such as theta that we try to find(the optimal value) while training the data. After we have found the optimal values for these parameters, we can put the data aside or erase it from the computer and just use the model with parameters to make predictions. Remember, the model is just a function.

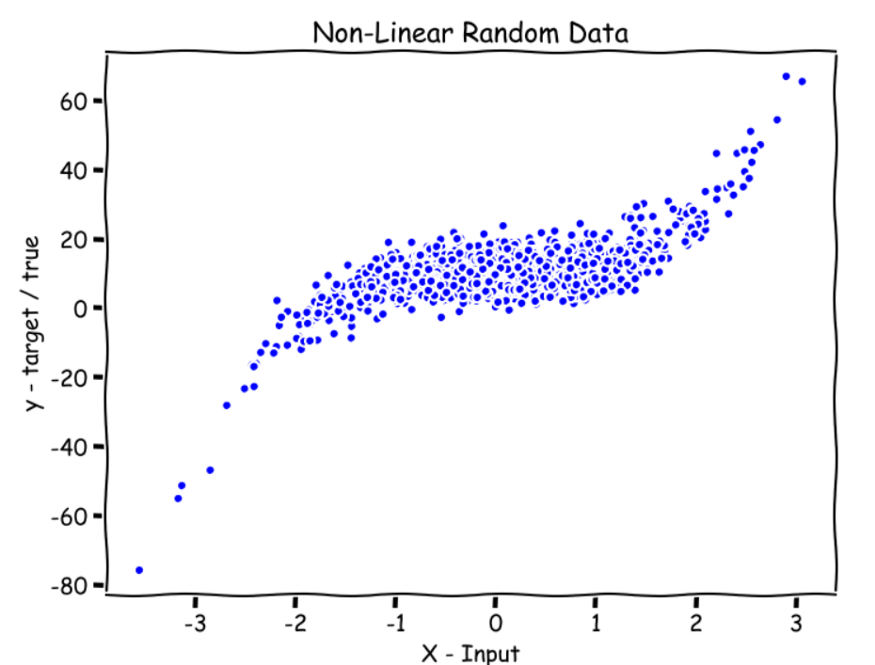
**Non-Parametric** — In a Non-Parametric Algorithm, you always have to keep the data and the parameters in your computer memory to make predictions. And that’s why this type of algorithm may not be great if you have a really really massive dataset.

Locally Weighted Linear Regression

Let us use the following randomly generated data as a motivational example to understand the Locally weighted linear regression.

**import numpy as npnp.random.seed(8)X = np.random.randn(1000,1)  
y = 2\*(X\*\*3) + 10 + 4.6\*np.random.randn(1000,1)**

https://miro.medium.com/max/60/1*DHYvJg55uN-Kj8jHaxDKvQ.png?q=20



Random data; Image by Author

In *Linear Regression* we would fit a straight line to this data but that won’t work here because the data is non-linear and our predictions would end up having large errors. We need to fit a curved line so that our error is minimized.

Notations —

n → number of features (1 in our example)

m → number of training examples (1000 in our example)

X(Uppercase) → Features

y → output sequence

x (lowercase)→ The point at which we want to make the prediction. Referred as point in the code.

x(i) →ith training example

In Locally weighted linear regression, we give the model the x where we want to make the prediction, then the model gives all the x(i)’s around that x a higher weight close to one, and the rest of x(i)’s get a lower weight close to zero and then tries to fit a straight line to that weighted x(i)’s data.

This means that if want to make a prediction for the green point on the x-axis **(see Figure-1 below)**, the model gives higher weight to the input data i.e. x(i)’s near or around the circle above the green point and all else x(i) get a weight close to zero, which results in the model fitting a straight line only to the data which is near or close to the circle. The same goes for the purple, yellow, and grey points on the x-axis.

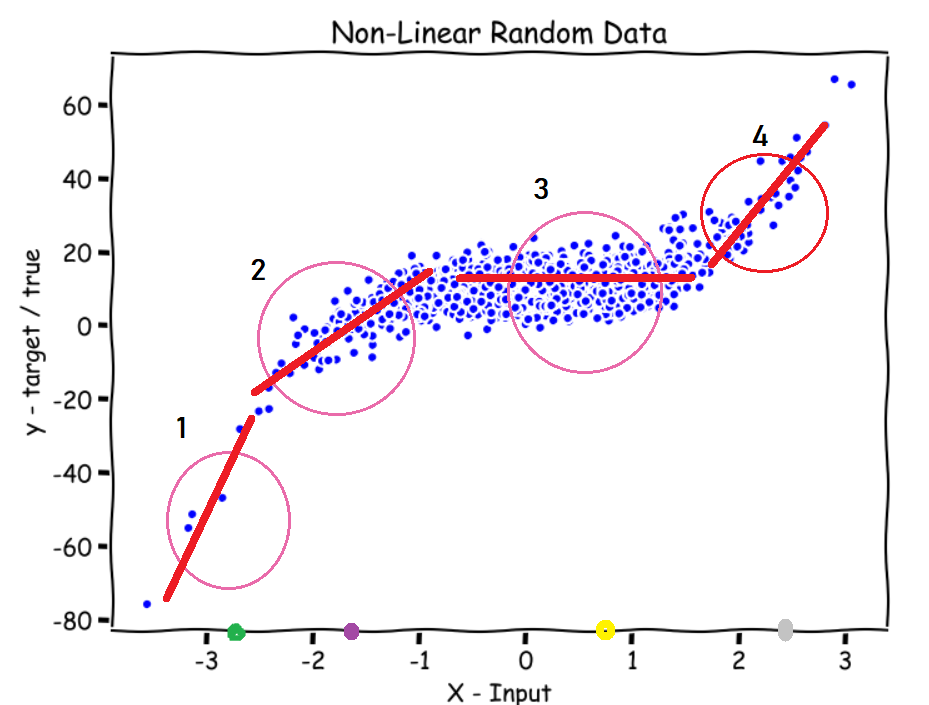


Figure - 1; Image by Author

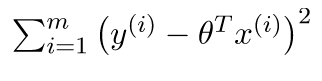
Two questions come to mind after reading that —

1. **How to assign the weights?**
2. **How big should the circle be?**

Weighting function (w(i) →weight for the ith training example)

In Linear regression, we had the following loss function —

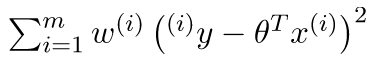
https://miro.medium.com/max/60/1*-7tUX3vd89cyBZp93rgiUA.png?q=20



Loss Function for Linear Regression; source: geeksforgeeks.org

The modified loss for locally weighted regression —

https://miro.medium.com/max/60/1*mTBHc2-V51S4gC4qwyZpjg.png?q=20

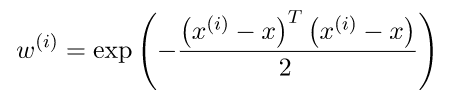


Loss Function for Locally weighted Linear Regression; source: geeksforgeeks.org

w(i) (the weight for the ith training example) is the only modification.

where,

https://miro.medium.com/max/60/1*McabEu7nOc2KjsyP3x6nVw.png?q=20



The weighting Function; source: geeksforgeeks.org

x is the point where we want to make the prediction. x(i) is the ith training example.

***The value of this function is always between 0 and 1.***

So, if we look at the function, we see that

* If |x(i)-x| is small, w(i) is close to 1.
* If |x(i)-x| is large, w(i) is close to 0.

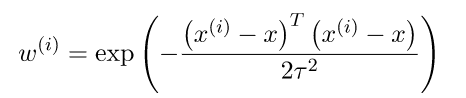
The x(i)’s which are far from x get w(i) close to zero and the ones which are close to x, get w(i) close to 1.

In the loss function, it translates to error terms for the x(i)’s which are far from x being multiplied by almost zero and for the x(i)’s which are close to x get multiplied by almost 1. In short, it only sums over the error terms for the x(i)’s which are close to x.

**How big should the circle be?**

We introduce a hyperparameter tau in the weighting function which decided how big the circle should be.

https://miro.medium.com/max/60/1*RajKXugYR612lZCqDfhulA.png?q=20



Weighting Function with tau; source: geeksforgeeks.org

By changing the value of tau we can choose a fatter or a thinner width for circles.

***For the math people here, tau is the bandwidth of the Gaussian bell-shaped curve of the weighing function.***

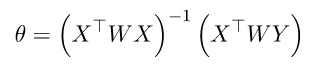
Let’s code the weighting matrix. See comments (#).

# Weight Matrix in code. It is a diagonal matrix.**def wm(point, X, tau):**   
   
 # tau --> bandwidth  
 # X --> Training data.  
 # point --> the x where we want to make the prediction.  
   
 # m is the No of training examples .  
  **m = X.shape[0]**   
   
 # Initialising W as an identity matrix.  
 **w = np.mat(np.eye(m))**   
   
 # Calculating weights for all training examples [x(i)'s].  
 **for i in range(m):   
 xi = X[i]   
 d = (-2 \* tau \* tau)   
 w[i, i] = np.exp(np.dot((xi-point), (xi-point).T)/d)   
   
 return w**

Finally, The Algorithm

Actually, there exists a closed-form solution for this algorithm which means that we do not have to train the model, we can directly calculate the parameter theta using the following formula.

https://miro.medium.com/max/60/1*9aM27k8gO0jkfa8u5G5V8Q.png?q=20



Closed-form solution for theta; source: geeksforgeeks.org

**How to calculate theta Do you ask?**

Just take the partial derivative of the modified loss function with respect to theta and set it equal to zero. Then, do a little bit of linear algebra to get the value of theta.

For reference — [Closed-form solution for locally weighted linear regression](http://www.dsplog.com/2012/02/05/weighted-least-squares-and-locally-weighted-linear-regression/)

And after calculating theta, we can just use the following formula to predict.

https://miro.medium.com/max/60/1*Nb14EtCwnAYB7rJ2eXJAAw.png?q=20

https://miro.medium.com/max/78/1*Nb14EtCwnAYB7rJ2eXJAAw.png

The formula for prediction; source: geeksforgeeks.org

Let’s code a predict function. See comments (#)

**def predict(X, y, point, tau):**   
   
 # m = number of training examples.   
  **m = X.shape[0]**   
   
 # Appending a cloumn of ones in X to add the bias term.  
## # Just one parameter: theta, that's why adding a column of ones #### to X and also adding a 1 for the point where we want to #### predict.   
 **X\_ = np.append(X, np.ones(m).reshape(m,1), axis=1)**   
 # point is the x where we want to make the prediction.   
 **point\_ = np.array([point, 1])**   
 # Calculating the weight matrix using the wm function we wrote # # earlier.   
 **w = wm(point\_, X\_, tau)**   
 # Calculating parameter theta using the formula.  
 **theta = np.linalg.pinv(X\_.T\*(w \* X\_))\*(X\_.T\*(w \* y))**   
 # Calculating predictions.   
 **pred = np.dot(point\_, theta)**   
 # Returning the theta and predictions   
 **return theta, pred**

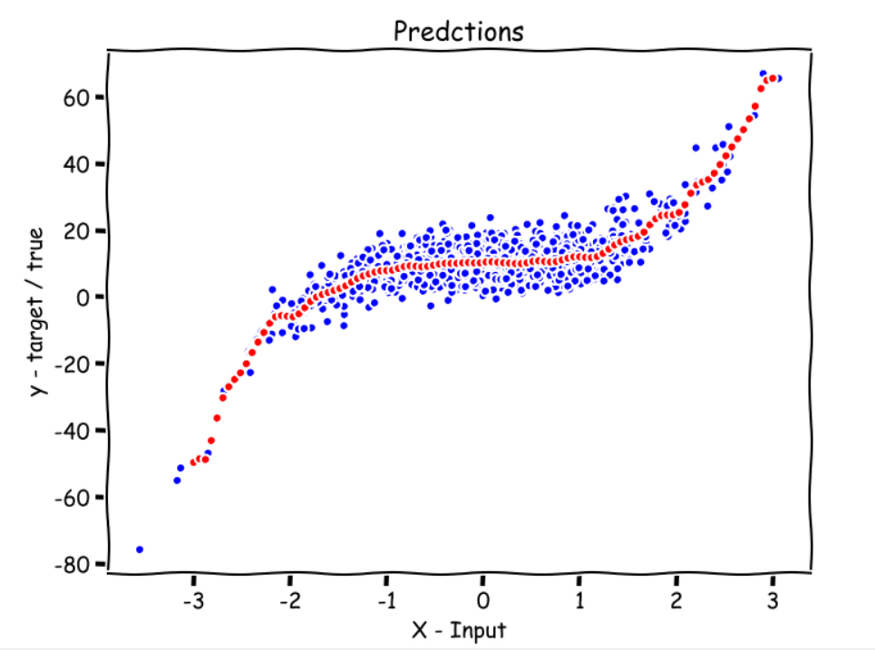
Plotting Predictions

Now, let’s plot our predictions for about 100 points(x) which are in the domain of x.

See comments (#)

**def plot\_predictions(X, y, tau, nval):** # X --> Training data.   
 # y --> Output sequence.  
 # nval --> number of values/points for which we are going to  
 # predict. # tau --> the bandwidth.   
 # The values for which we are going to predict.  
 # X\_test includes nval evenly spaced values in the domain of X.  
 **X\_test = np.linspace(-3, 3, nval)**   
 # Empty list for storing predictions.   
 **preds = []**   
 # Predicting for all nval values and storing them in preds.   
 **for point in X\_test:   
 theta, pred = predict(X, y, point, tau)   
 preds.append(pred)**  
   
 # Reshaping X\_test and preds  
  **X\_test = np.array(X\_test).reshape(nval,1)  
 preds = np.array(preds).reshape(nval,1)**  
   
 # Plotting   
  **plt.plot(X, y, 'b.')  
 plt.plot(X\_test, preds, 'r.')** # Predictions in red color. **plt.show()plot\_predictions(X, y, 0.08, 100)**

https://miro.medium.com/max/60/1*nydrc60uVwiecsI3yPxHag.png?q=20



Predictions for 100 values between -3 and 3 are shown in red; Image by Author

**Red points are the predictions for 100 evenly spaced numbers between -3 and 3.**

Looks pretty good. Play around with the value of tau .

When to use Locally Weighted Linear Regression?

* When n (number of features) is small.
* If you don’t want to think about what features to use.

For questions, comments, concerns, talk to be in the response section. More ML from scratch is coming soon.

Check out the Machine Learning from scratch series —

* Part 1: [Linear Regression from scratch in Python](https://medium.com/analytics-vidhya/linear-regression-from-scratch-in-python-b6501f91c82d?source=your_stories_page-------------------------------------)
* Part 2: [Locally Weighted Linear Regression in Python](https://towardsdatascience.com/locally-weighted-linear-regression-in-python-3d324108efbf?source=your_stories_page-------------------------------------)
* Part 3: [Normal Equation Using Python: The Closed-Form Solution for Linear Regression](https://towardsdatascience.com/normal-equation-in-python-the-closed-form-solution-for-linear-regression-13df33f9ad71?source=your_stories_page-------------------------------------)
* Part 4: [Polynomial Regression From Scratch in Python](https://towardsdatascience.com/polynomial-regression-in-python-b69ab7df6105)