Nadaraya-Watson Regression

Linkang Dong

April 26, 2025

1 Introduction

This is a simple implementation of the Nadaraya-Watson regression [1, 2] algorithm in Python.

2 Data Generation

We generate some random data from the model:

$$y = 2\sin(x) + x^{0.8} + \epsilon$$

where x is uniformly distributed in the range $[0, 5]$ and ϵ is a Gaussian noise with mean 0 and standard deviation 0.5.

3 Kernel

First, we define some kernel functions here, which are used to compute the weights for the Nadaraya-Watson regression. The kernel function $K(\mathbf{d})$ is defined as a function of the distance $\mathbf{d}$ between the query point $\mathbf{q}$ and the training point $\mathbf{k}$. The distance is computed as follows:

$$\mathbf{d} = \|\mathbf{q} - \mathbf{k}\|$$

The kernel function is then defined as a function of the distance $\mathbf{d}$, and it can take different forms. The most common ones are:

$$\text{Gaussian:} \quad K(\mathbf{d}) = \exp\left(-\frac{\mathbf{d}^2}{2\sigma^2}\right) \qquad \text{Boxcar:} \quad K(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{d} \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Constant:} \quad K(\mathbf{d}) = 1 \qquad \text{Epanechikov:} \quad K(\mathbf{d}) = \max(0, 1 - \mathbf{d})$$

We plot different kernel functions as shown in Figure 1:

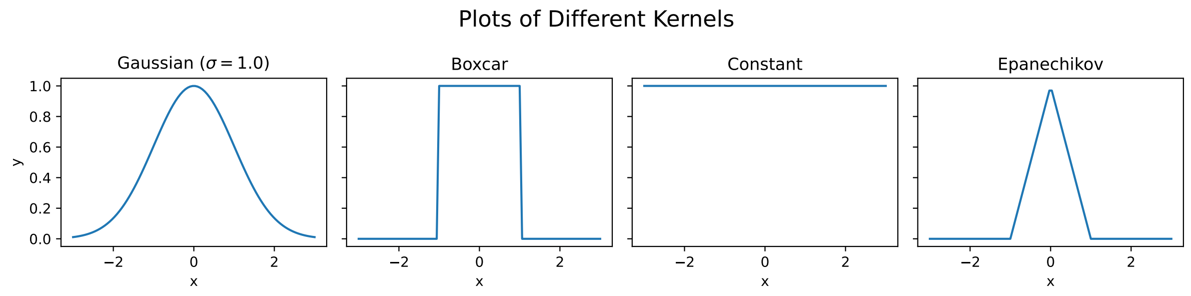

Figure 1: Plot of different kernels.

Our weights essentially determine the influence of neighboring points on a given point. For the Gaussian kernel, closer points exert a greater influence, while farther points have less impact. With the boxcar kernel, all points within a distance of 1 have equal influence (this distance can be adjusted as a bandwidth parameter), and points beyond this distance have zero influence. The constant kernel assigns equal influence to all points, effectively averaging them. The Epanechnikov kernel resembles a 'hard' Gaussian kernel: influence increases linearly with proximity, but abruptly drops to zero beyond a certain distance.

The Gaussian kernel is the most frequently used. It can be considered a 'soft' version of the boxcar kernel. From a signal processing standpoint, the boxcar kernel acts like a discrete signal, whereas the Gaussian kernel behaves like a continuous signal. In terms of filtering, the boxcar kernel is similar to a band-pass filter (or a low-pass filter if only the positive region is considered), while the Gaussian filter passes all frequencies but attenuates higher frequencies more significantly.

For the Gaussian kernel, the bandwidth can be adjusted by tuning the standard deviation σ. A larger σ results in a wider range of influence from neighboring points, while a smaller σ narrows this range. Figure 2 illustrates Gaussian kernels with varying bandwidths:

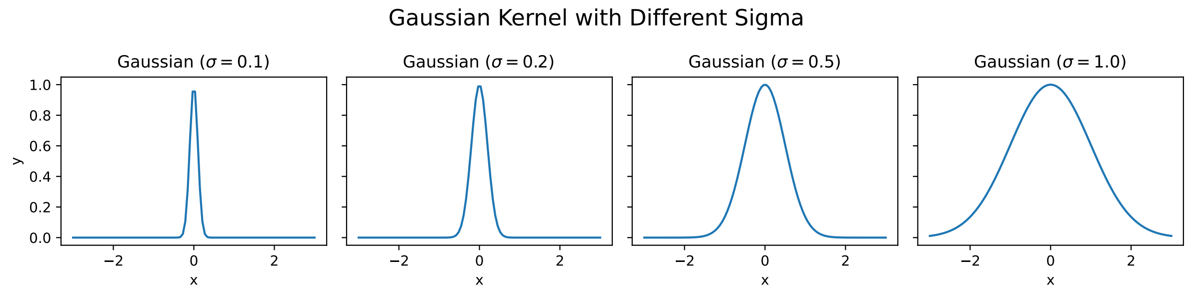

Figure 2: Gaussian kernel using different bandwidths.

4 Nadaraya-Watson Regression

Our implementation of the Nadaraya-Watson regression algorithm is as follows:

```python
def nadaraya_watson_regression(x_train, y_train, x_val, kernel):
    diff = x_val.view(-1,1) - x_train.view(1,-1)
    weights = kernel(diff).type(torch.float32)
    weights /= torch.sum(weights, dim=1, keepdim=True)
    y_pred = (weights @ y_train.view(-1,1)).squeeze(1)
    return y_pred
```

The few lines of code above essentially encapsulate the core idea of Nadaraya-Watson regression: **learning by example**. Suppose we have a collected dataset (`x_train`, `y_train`). We need to use this data to estimate the corresponding values for new data points (`x_val`) to predict `y_pred`. This utilizes an interpolation-like approach. First, we compute the similarity (or distance, used by the kernel) between the new points `x_val` and the training points `x_train`. Then, using these similarities as weights (after normalization), we compute a weighted average of the training outputs `y_train` to obtain the predicted values `y_pred`.

The regression results using different kernel functions are shown in Figure 3. We observe that the Gaussian kernel produces the smoothest result, though it differs somewhat from the true function. The boxcar kernel yields the least smooth result. The constant kernel simply produces a horizontal line (equivalent to averaging all samples). The Epanechnikov kernel gives a result similar to the Gaussian kernel but appears to perform best here.

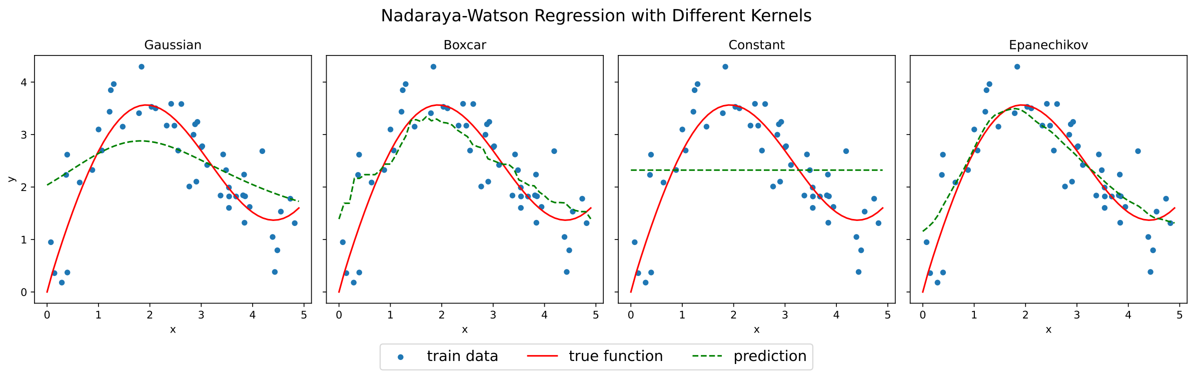

Figure 3: Estimated regression function using different kernels.

We can also perform regression using different bandwidths. Using the Gaussian kernel, we perform regression with bandwidths of 0.1, 0.2, 0.5, and 1.0, as shown in Figure 4. We observe that a larger bandwidth results in a smoother estimated regression function. At a bandwidth of 0.5, the estimated function is closest to the true function. However, with bandwidths of 0.1 and 0.2, the predicted curve exhibits significant ”jittering”. This occurs because a very small bandwidth considers only a limited local neighborhood, leading to high variance (similar to overfitting).

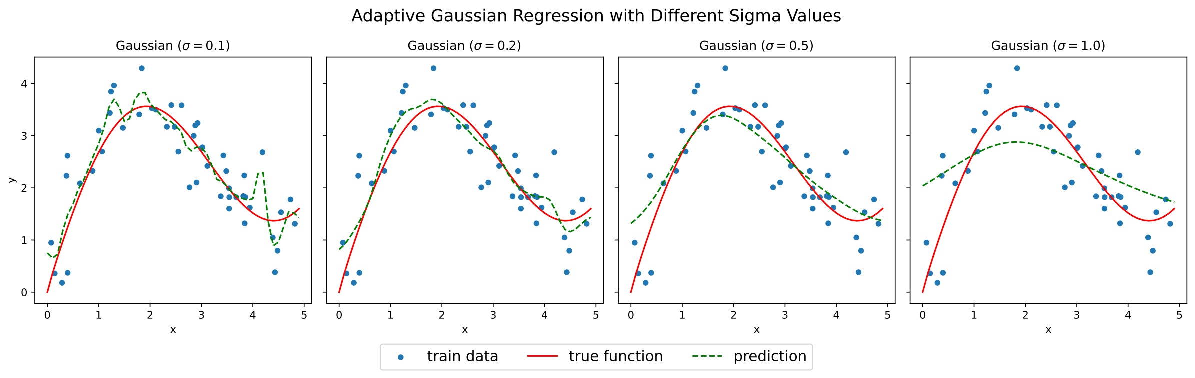

Figure 4: Estimated regression function using different bandwidths for the Gaussian kernel.

Additionally, we visualize the data for all `x_val` and `x_train`. The weight matrix, representing the attention weights between pairs of samples after processing by the kernel function, is plotted as a heatmap, as shown in Figure 5. From this figure, the effect of the bandwidth is clearly visible, indicating how many surrounding `x_train` points are considered by each `x_val`.

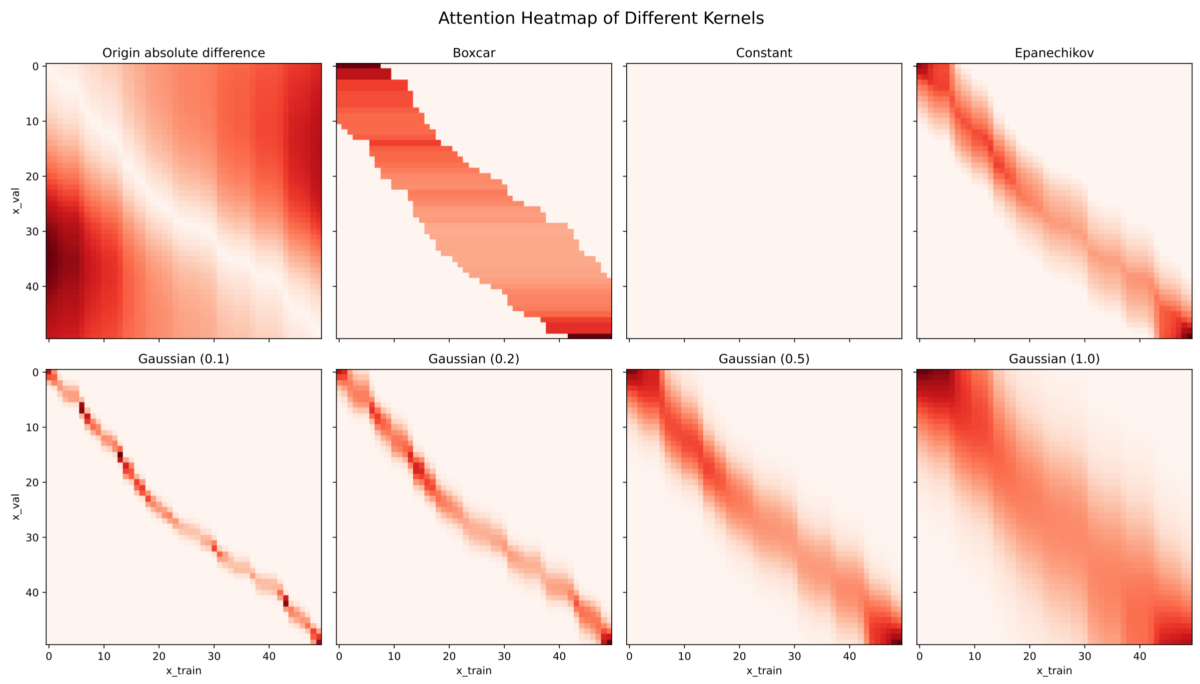

Figure 5: Heatmap of the weight matrix.

References

[1] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Attention pooling: Nadaraya-watson kernel regression. `https://d2l.ai/chapter_attention-mechanisms-and-transformers/attention-pooling.html`, 2023.

[2] Wikipedia contributors. Kernel regression. `https://en.wikipedia.org/wiki/Kernel_regression`, 2025.