Kernels and Smoothers

Muchang Bahng

Spring 2025

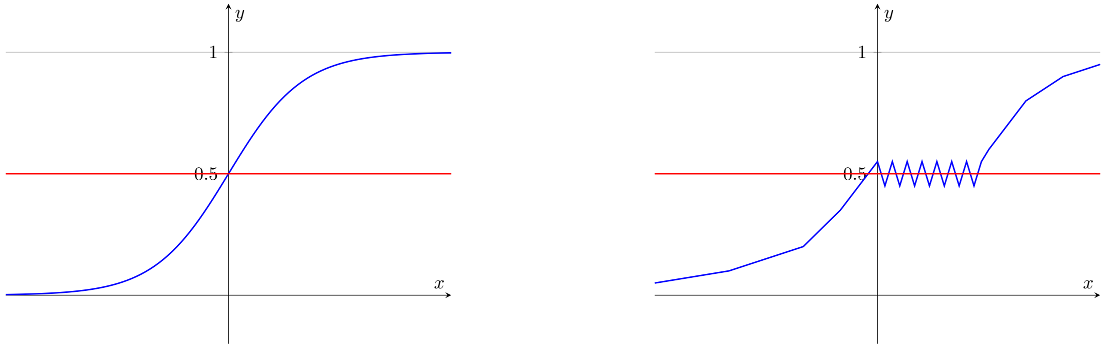
Contents

We have extensively studied parameteric supervised models like linear regression, SVMs, and softmax regression. Now we introduce analogues in the nonparameteric scheme, starting with kernel regression. If all data was intrinsically linear, then this would be an ideal world where we only need linear regression. However, this is not the case in reality, and we must resort to more flexible models to fit nonlinear data.

The basic motivation behind kernels is that samples with similar covariates $x_1, \ldots, x_d$ should be similar in their response y. Therefore, two data points $x^{(1)}, x^{(2)}$ near each other should have similar $y^{(1)}, y^{(2)}$ and so if we are given a new sample $x^{(n+1)}$, we should use similar samples to predict the corresponding $y^{(n+1)}$.

Like with a lot of things, we can in fact formalize this by constructing *reproducing kernel Hilbert spaces (RKHS)*. RKHS regression provides the theoretical foundation that explains why many kernel methods work. The representer theorem shows that solutions to regularized regression problems in an RKHS can be expressed as linear combinations of kernel functions evaluated at nearby training points.

One thing to keep in mind for nonparameteric classification is that when you are using plug-in classifiers (train a regressor and then run it through a threshold function), there may be problems near the boundary. For example, say that in a one-dimensional case, you would want the regressor to smoothly fit across the decision boundary. Classification could be difficult if we have bad boundaries, but the good news is that if the probability that the data lies in the margins is not high, then we can do pretty well in classification.



(a) Good case.

(b) Bad case.

Figure 1: We will generally have challenges differentiating points at the boundaries of a plug-in classifier. This may matter if $p(x, y)$ is concentrated around this jagged region, or it may not if most of the masses are concentrated far away from the boundary.

1 Smoothers

1.1 Linear Smoothers

2 K Nearest Neighbors

KNN was first introduced by Cover in 1967 [CH67].

2.1 Classification

One method to create a nonparameteric classifier is to take a nonparameteric regressor and put it through a threshold function to get a plug-in classifier. KNN is an example of such a classifier.

Given a bunch of points in a metric space $(\mathcal{X}, d)$ that have classification labels, we want to label new datapoints $\hat{\mathbf{x}}$ based on the labels of other points that already exist in our dataset. One way to look at it is to look for close points within the dataset and use their labels to predict the new ones.

Definition 2.1 (Closest Neighborhood)

Given a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}$ and a point $\hat{\mathbf{x}} \in (\mathcal{X}, d)$, let the **k closest neighborhood** of $\hat{\mathbf{x}}$ be $N_k(\hat{\mathbf{x}}) \subset [N]$ defined as the indices i of the k points in $\mathcal{D}$ that is closest to $\hat{\mathbf{x}}$ with respect to the distance metric $d_{\mathcal{X}}$.

Definition 2.2 (K Nearest Neighbors)

The **K Nearest Neighbors (KNN)** is a discriminative nonparametric supervised learning algorithm that doesn't have a training phase. Given a new point $\hat{\mathbf{x}}$, we look at all points in its k closest neighborhood, and $h(\hat{\mathbf{x}})$ will be equal to whatever the majority class will be in. Let us one-hot encode the labels $\mathbf{y}^{(i)}$ into $\mathbf{e}_i$'s, and the number of data point in the ith class can be stored in the variable

$$a_i = \sum_{i \in N_k(\hat{\mathbf{x}})} 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_i\}} \tag{1}$$

which results in the vector storing the counts of labels in the k closest neighborhood

$$\mathbf{a} = (a_1, a_2, \ldots, a_{\mathcal{K}}) = \left(\sum_{i \in N_k(\hat{\mathbf{x}})} 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_1\}}, \sum_{i \in N_k(\hat{\mathbf{x}})} 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_2\}}, \ldots, \sum_{i \in N_k(\hat{\mathbf{x}})} 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_{\mathcal{K}}\}} \right) \tag{2}$$

and take the class with the maximum element as our predicted label.

The best choice of K depends on the data:

1. Larger values of K reduces the effect of noise on the classification, but make boundaries between classes less distinct. The number of misclassified data points (error) increases.
2. Smaller values are more sensitive to noise, but boundaries are more distinct and the number of misclassified data points (error) decreases.

Too large of a K value may increase the error too much and lead to less distinction in classification, while too small of a k value may result in us overclassifying the data. Finally, in binary (two class) classification problems, it is helpful to choose K to be odd to avoid tied votes.

This is an extremely simple algorithm that may not be robust. For example, consider $K \geq 3$, and we are trying to label a point $\hat{\mathbf{x}}$ that happens to be exactly where one point is on our dataset $\mathbf{x}^{(i)}$. Then, we should do $h(\hat{\mathbf{x}}) = y^{(i)}$, but this may not be the case if there are no other points with class $y^{(i)}$ in the k closest neighborhood of $\mathbf{x}^{(i)}$. Therefore, we want to take into account the distance of our new points from the others.

Definition 2.3 (Weighted Nearest Neighbor Classifier)

Let us define a monotinically decreasing function $\omega : \mathbb{R}_0^+ \mapsto \mathbb{R}_0^+$. Given a point $i \in N_k(\hat{\mathbf{x}})$, we can construct the weight of our matching label as inversely proportional to the distance: $\omega_i[d(\hat{\mathbf{x}}, \mathbf{x}^{(i)})]$ and store them as

$$\mathbf{a} = (a_1, a_2, \ldots, a_{\mathcal{K}}) = \bigg(\sum_{i \in N_k(\hat{\mathbf{x}})} \omega_i 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_1\}}, \sum_{i \in N_k(\hat{\mathbf{x}})} \omega_i 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_2\}}, \ldots, \sum_{i \in N_k(\hat{\mathbf{x}})} \omega_i 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_{\mathcal{K}}\}} \bigg) \tag{3}$$

and again take the class with the maximum element.

One caveat of KNN is in high dimensional spaces, as its performance degrades quite badly due to the curse of dimensionality.

Example 2.1 (Curse of Dimensionality in KNN)

Consider a dataset of N samples uniformly distributed in a d-dimensional hypercube. Now given a point $x \in [0,1]^d$, we want to derive the expected radius r_k required to encompass its k nearest neighbors. Let us define this ball to be $B_{r_k} \coloneqq \{z \in \mathbb{R}^d \mid ||z - x||_2 \leq r_k\}$. Since thse N points are uniformly distributed, the expected number of points contained in $B_{r_k}(x)$ is simply the proportion of the volume that $B_{r_k}(x)$ encapsulates in the box, multiplied by N. Therefore, for some fixed x and r, let us denote $Y(x,y)$ as the random variable representing the number of points contained within $B_r(x)$. By linearity of expectation and summing over the expectation for whether each point will be in the ball, we have

$$\mathbb{E}[Y(x,r)] = N \cdot \frac{\mu(B_r(x) \cap [0,1]^d)}{\mu([0,1]^d)}$$

where μ is the Lebesgue measure of $\mathbb{R}^d$. Let us assume for not that we don't need to worry about cases where the ball is not fully contained within the cube, so we can just assume that Y is only dependent on r: $Y(r)$. Also, since the volume of the hypercube is 1, $\mu([0,1]^d) = 1$ and we get

$$\mathbb{E}[Y(r)] = N \cdot C_d \cdot r^d$$

which we set equal to k and evaluate for r. C_d is a constant such that the volume of the hypersphere of radius r can be derived as $V = C_d \cdot r^d$. We therefore get

$$N \cdot C_d \cdot r_k^d = k \implies r_k = \left(\frac{k}{NC_d} \right)^{1/d}$$

It turns out that C_d decreases exponentially, so the radius r_k explodes as d grows. Another way of looking at this is that in high dimensions, the ℓ_2 distance between all the pairwise points are close in every single dimension, so it becomes harder to distinguish points that are close vs those that are far.

2.2 Concentration Bounds

Theorem 2.1 (Devroye and Györfi 1985)

Suppose that the distribution of X has a density and that $k \to \infty$ and $k/n \to 0$. For every $\epsilon > 0$ the following is true. For all large n,

$$\mathbb{P}(R(\hat{h}) - R_* > \epsilon) \leq e^{-n\epsilon^2/(72\gamma_d^2)} \tag{4}$$

where $\hat{h}_n$ is the k-nearest neighbor classifier estimated on a sample of size n, and where γ_d depends on the dimension d of X.

2.3 Approximate K Nearest Neighbors

2.4 Regression

Question 2.1 (To Do)

Maybe similar like a kernel regression?

When we want to do nonparametric regression, i.e. when dealing with nonlinear functions, we can construct a function that uses local averaging of its nearby points.

Example 2.2 (Local Averaging)

Say that we want to fit some function through a series of datapoints in simple regression (one covariate). Then, what we can do is take some sliding window and our vale of the function at a point x is the average of all values in the window $[x - \delta, x + \delta]$.

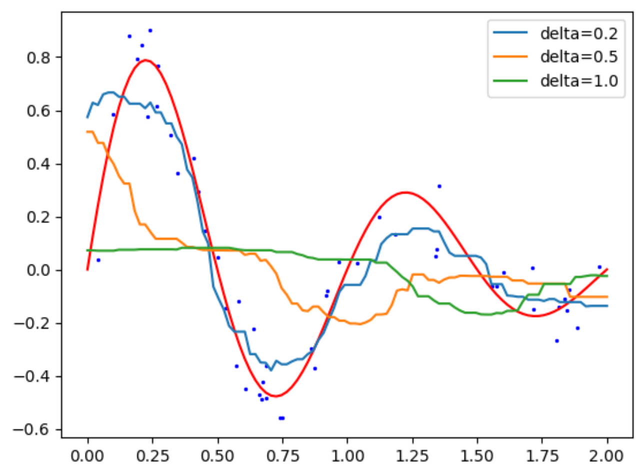


Figure 2: K means smoother

Code 2.1 (MWS of K Nearest Neighbor Regression in scikit-learn)

Local averaging is implemented as the K nearest neighbor regressor in scikit learn. It is slightly different in the way that it doesn't use the points within a certain δ away but rather the K nearest points. Either way, a minimal working example of this is

```python
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
from sklearn.neighbors import KNeighborsRegressor
neigh = KNeighborsRegressor(n_neighbors=2)
neigh.fit(X, y)
print(neigh.predict([[1.5]]))
```

Note that since $\hat{f}$ is a combination of step functions, this makes it discontinuous at points.

3 Kernel Regression

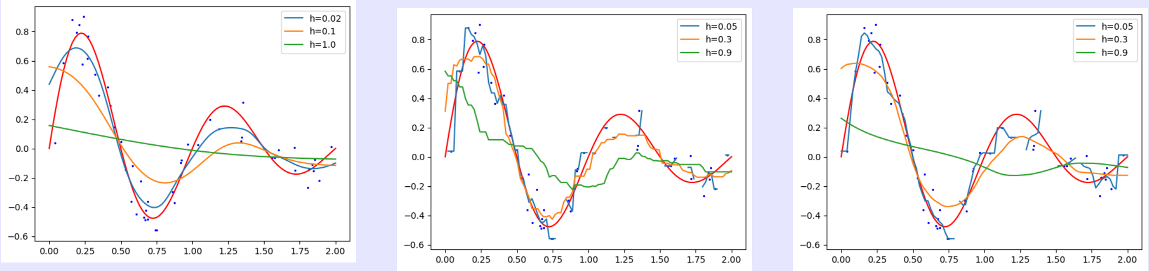
K nearest neighbor regression puts equal weights on both near and far points, as long as they are in the window. This may not be ideal, so a simple modification is to *weigh* these points according to their distance from the middle x. We can do this with a kernel, as the name suggests. Now this is not the same thing as a Mercer kernel in RKHS, so to distinguish that I will call it a *local averaging kernel*.

Definition 3.1 (Local Averaging Kernel)

A **kernel** is any smooth, symmetric, and non-negative function $K : \mathbb{R} \to \mathbb{R}$.

Example 3.1 (Some Simple Kernels)

Here are some popular kernels.



(a) $K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.

(b) $K(x) = \frac{1}{2}\mathbb{1}(|x| \leq 1)$.

(c) $K(x) = \frac{3}{4}(1 - x^2)\mathbb{1}(|x| \leq 1)$.

Figure 3: The Gaussian, boxcar, and Epanechnikov kernels.

The idea is to simply take the weighted average of the y_i's. $\hat{f}(x) = \sum_i y_i \ell(x)$ where $\sum_i \ell_i(x) = 1$. The reason we'd like to have the weights to sum to 1 is that if we had data that was constant (i.e. all y_i's are the same), then the fitted function should be constant at that value as well.

Definition 3.2 (Kernel Regression)

Given a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$, a **kernel regression** model[a], or **local smoothing regression**, is a model of the form

$$\hat{y} = f(x) = \frac{\sum_i y_i K\left(\frac{\|x - x_i\|}{h}\right)}{\sum_i K\left(\frac{\|x - x_i\|}{h}\right)} \tag{5}$$

where h is the **bandwidth** and the denominator is made sure so that the coefficients sum to 1. Note that this function can have kernels defined at all points in $\mathcal{X}$, but it is interesting to examine it at the training points.

[a] Not to be confused with RKHS regression, or kernel ridge regression!

Denoting $y = (y_1, \ldots, y_n) \in \mathbb{R}^n$ and the vector $f(x) = (f(x_1), \ldots, f(x_n))$, if we can write the kernel function as $\hat{y} = \hat{f}(x) = Sy$, which in matrix form, is

$$\begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} \hat{f}(x_1) \\ \vdots \\ \hat{f}(x_n) \end{bmatrix} = \begin{bmatrix} \ell_1(x_1) & \cdots & \ell_n(x_1) \\ \vdots & \ddots & \vdots \\ \ell_1(x_n) & \cdots & \ell_n(x_n) \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \tag{6}$$

then we say that we have a **linear smoother**, with stochastic matrix S being our **smoothing matrix**.

Furthermore, the theme of linearity is important and will be recurring. The kernel estimator is defined for all x, but it's important to see its behavior at the training points x_i. The estimator $\hat{y} = \hat{f}(x)$ is a linear combination of the y_i's, and the coefficients $\ell_i(x_j)$ depend on the values of x_j. Therefore, we have $\hat{y} = Sy$, which is very similar to the equation $\hat{y} = Hy$ in linear regression, where H is the hat matrix that projects y onto the column space of x. Nonparametric regression has the same form, but rather than being a projection, it is a linear smoothing matrix. Therefore, this theme unifies both linear regression and nonparametric regression. Linear smoothers, spline smoother, Gaussian processes, are all just different choices of the smoothing matrix S. However, not all nonparametric estimators are linear smoothers, as we will see later.

3.1 Bias Variance Tradeoff

Just like we do with OLS, we want to minimize the MSE loss. It turns out that from a theoretical point of view, the choice of the kernel doesn't really matter. What really matters is the bandwidth h since that is what determines the bias variance tradeoff. To see why, if $h = 0$, then it will simply interpolate the points and variance is extremely high, and if $h = \infty$, then the fitted function will be constant at $\bar{Y}$, leading to high bias. The following theorem formalizes this but for the simpler case of $d = 1$.

Theorem 3.1 (Bias Variance Tradeoff of Kernel Regression)

Suppose that $d = 1$ and that m'' is bounded. Also suppose that X has a nonzero, differentiable density p and that the support is unbounded. Then, the risk is

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}}\left[(\hat{m}(x) - m(x))^2\right] &= \frac{h^4}{4}\left(\int x^2 K(x)\right)^2 \int \left(m''(x) + 2m'(x)\frac{p'(x)}{p(x)}\right)^2 dx \qquad (7)\\
&\quad + \frac{\sigma^2 \int K^2(x)\,dx}{nh_n} \int \frac{dx}{p(x)} + o\left(\frac{1}{nh_n}\right) + o(h_n^4) \qquad (8)
\end{aligned}$$

The first term is the squared bias and the second term is the variance. p represents the density of x.

Proof.

We first denote

$$\hat{f}(X) = \frac{\frac{1}{nh}\sum_{i=1}^n K\left(\frac{X - X_i}{h}\right) Y_i}{\frac{1}{nh}\sum_{i=1}^n K\left(\frac{X - X_i}{h}\right)} \tag{9}$$

where the denominator is the kernel density estimator $\hat{p}(X)$. Therefore, we rewrite

$$\begin{aligned}
\hat{f}(x) - f(x) &= \frac{\hat{a}(x)}{\hat{p}(x)} - f(x) \qquad (10)\\
&= \left(\frac{\hat{a}(x)}{\hat{p}(x)} - f(x)\right)\left(\frac{\hat{p}(x)}{p(x) + 1 - \frac{\hat{p}(x)}{p(x)}}\right) \qquad (11)\\
&= \frac{\hat{a}(x) - f(x)\hat{p}(x)}{p(x)} + \frac{(\hat{f}(x) - f(x))(p(x) - \hat{p}(x))}{p(x)} \qquad (12)
\end{aligned}$$

as $n \to \infty$ both $\hat{f}(x) - f(x)$ and $p(x) - \hat{p}(x)$ going to 0, and since they're multiplied in the second term, it will go to 0 very fast. So the dominant term is the first term, and we can write the above as approximately

$$\hat{f}(x) - f(x) \approx \frac{\hat{a}(x) - f(x)\hat{p}(x)}{p(x)} \tag{13}$$

TBD continued. Wasserman lecture 6, 10:00.

From the theorem above, we can see that if the bandwidth is small, then h^4 is small and the bias decreases. However, there is a h term in the denominator of the variance term, which also trades it off. However, there are two problems.

1. We can furthermore see that the bias is sensitive to $p'/p(x)$. This means that if the density is steep, then the bias will be high. This is known as *design bias*, which refers to bias stemming from how the x's are distributed.

2. Another problem that is not contained in the theorem is the *boundary bias*, which states that if you're near the boundary of the distribution (near the boundary of its support), then the bias also explodes. This happens to be very nasty especially in high dimensions where most of the data tends to be near the boundary.

It turns out that this can be easily fixed with local polynomial regression, which gets rid of this term in the bias without any cost to variance. This means that this is unnecessary bias.

4 Local Linear/Polynomial Regression

Now another way to think about the kernel estimator is as such. Suppose that you're doing linear regression on a bunch of points and you want to choose a c that minimizes the loss.

$$\sum_i (Y_i - c)^2 \tag{14}$$

You would just pick $c = \hat{Y}$. But if you are given some sort of locality condition, that the value of c should depend more on the values closer to it, you're really now minimizing

$$\sum_i (Y_i - c(x))^2 K\left(\frac{X_i - x}{h}\right) \tag{15}$$

Minimizing this by setting the derivative equal to 0 and solving gives us the kernel estimator. Therefore you're fitting some sort of local constant at a point X. But why fit a local constant, when you can fit a local line or polynomial? This is the idea behind local polynomial regression.

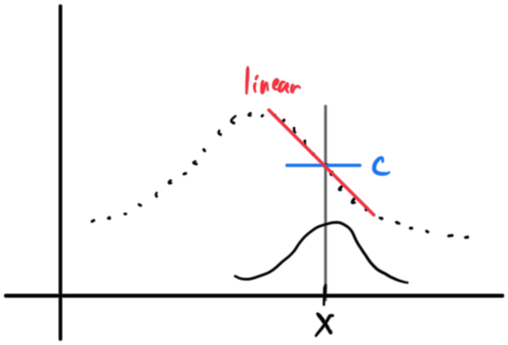


Figure 4: Rather than using a local constant, we can use a local linear estimator.

Therefore, we can minimize the modified loss.

Definition 4.1 (Local Polynomial Estimator)

The **local polynomial estimator** is a local linear kernel smoother that estimates the function $\hat{f}$ that minimizes the following loss.

$$\underset{\beta}{\text{argmin}} \sum_i K\left(\frac{X_i - x}{h}\right)\big(Y_i - \beta_0(x) - \beta_1(x)(x - X_i)\big) \tag{16}$$

So we can fit a line

$$f(\mu) \approx \hat{\beta}_0(x) + \hat{\beta}_1(x)(\mu - x) \tag{17}$$

and simply remove the intercept term to get the local linear estimator.

$$\hat{f}(x) = \hat{\beta}_0(x) \tag{18}$$

Note that this is not the same as taking the constant estimate. We are extracting the fitted intercept term and so $\hat{\beta}_0(x) \neq c(x)$.

4.1 Weighted Least Squares Solution

It turns out that this has an analytic solution. Looking the local polynomial loss should tell you that we're really doing OLS, but weighting the points differently. This sounds a lot like weighted least squares.

Theorem 4.1 (Weighted Least Squares)

The solution to the local linear estimator is similar to the weighted least squares solution.

$$\hat{\beta}(x) = \begin{pmatrix} \hat{\beta}_0(x) \\ \hat{\beta}_1(x) \end{pmatrix} = (X_x^T W_x X_x)^{-1} X_x^T W_x Y \tag{19}$$

where we have put the subscript x to emphasize that the matrices are dependent on x, and

$$X_x = \begin{pmatrix} 1 & X_1 - x \\ \vdots & \vdots \\ 1 & X_n - x \end{pmatrix} \qquad W_x = \begin{pmatrix} K\left(\frac{X_1 - x}{h}\right) & 0 & \cdots & 0 \\ 0 & K\left(\frac{X_2 - x}{h}\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K\left(\frac{X_n - x}{h}\right) \end{pmatrix} \tag{20}$$

Remember that at the end, once you compute $\hat{\beta}$, take the intercept term and your estimate is actually $\hat{\beta}_0$. Note that this of the form $(X_x^T W_x X_x)^{-1} X_x^T W_x Y = LY$, and so this is a linear smoother.

4.2 Bias Variance Decomposition

Computationally, it's similar to kernel regression and it gets rid of both the boundary and design bias. Let's see this mathematically.

Theorem 4.2 (Bias Variance Decomposition)

Under some regularity conditions, the risk of $\hat{m}$ is

$$\frac{h^4}{4} \int \left(\mathrm{tr}(m''(x) \int K(u) u u^T du) \right)^2 dP(x) + \frac{1}{n h_d} \int K^2(u) du \int \sigma^2(x) dP(x) + o(h_n^4 + (n h_n^d)^{-1}) \tag{21}$$

Proof.

For a proof, see Fan & Gijbels (1996).

For points near the boundary, the bias is $Ch^2 m''(x) + o(h^2)$ whereas, the bias is $Chm'(x) + o(h)$ for kernel estimators.

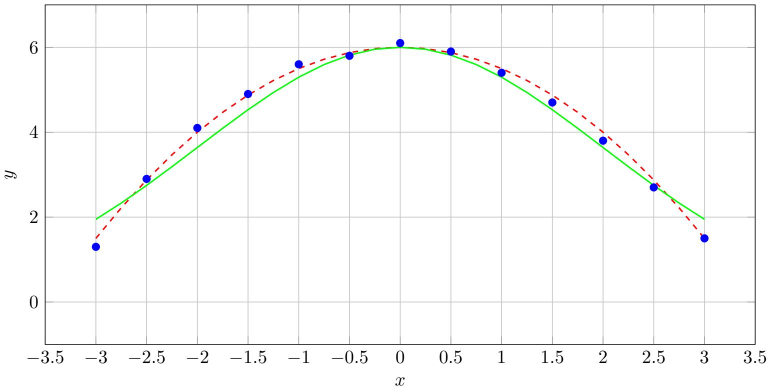


Figure 5: Diagram of the boundary bias. The green line represents what you would see for a fitted kernel regressor. At the boundaries (endpoints of the data), you can see that because the kernel is cut off, more and more bias accumulates. We avoid this in local polynomial regression.

4.3 Local Polynomial Regression

We can extend this and compute local polynomials rather than lines.

Definition 4.2 (Local Polynomial Estimator)

The **local polynomial estimator** is a local linear kernel smoother that estimates the function $\hat{f}$ that minimizes the following loss.

$$\operatorname*{argmin}_{\beta} \sum_i K\left(\frac{X_i - x}{h}\right)\big(Y_i - (\beta_0(x) - \beta_1(x)(x - X_i) + \ldots + \beta_k(x)(x - X_i)^k)\big) \tag{22}$$

People do this because there is also bias in the peaks and troughs of the data, and local quadratics can capture this much better. Beyond quadratics, there's not much more benefits and we have the accumulating cost of variance.

5 Splines

This is not local, but it's a linear smoother.

6 RKHS Regression

This is not local, but it's a linear smoother.

Code 6.1 (MWS of Kernel Ridge Regression in scikit learn)

```python
from sklearn.kernel_ridge import KernelRidge
import numpy as np
n_samples, n_features = 10, 5
rng = np.random.RandomState(0)
y = rng.randn(n_samples)
X = rng.randn(n_samples, n_features)
krr = KernelRidge(alpha=1.0)
krr.fit(X, y)
```

7 Additive Models and Naive Bayes

Additive models and naive bayes are both nonparameteric methods, but they are very similar.

7.1 Additive Models

In the most general case, we want to create nonparametric regression functions of the form

$$Y = f(x_1, \ldots, x_d) + \epsilon \tag{23}$$

We've done this for one dimensional case, but we can extend this to multiple dimensions through additive models of the form

$$Y = \sum_j f_j(x_j) + \epsilon \tag{24}$$

This gives us very interpretable models where we can clearly see the effect of each covariate on Y. Clearly, this is not as flexible as the previous model since they can't capture dependencies, but we can create sub-dependency functions and replace the form above to something like

$$Y = \sum_{i,j} f_{i,j}(x_i, x_j) + \epsilon \tag{25}$$

giving us more flexible models.

7.2 Naive Bayes

Say we are doing a binary classification problem. We treat the features as independent (which is very unrealistic) and model the probability distribution as

$$p(x \mid y = 0) = \prod_{j=1}^d p_j(x_j \mid y = 0), \qquad p(x \mid y = 1) = \prod_{j=1}^d p_j(x_j \mid y = 1) \tag{26}$$

When we take the log, we can see that it is like the additive model. This turns out to be surprisingly successful.

8 Nonlinear Smoothers, Trend Filtering

Tough example of the Dobbler function (like topologists sine curve). It's a pretty good fit but it's not too good since it's using a linear smoother (homogeneous). So we might need to fit it with nonlinear smoothers.

9 Nonparameteric Support Vector Machines

Rather than inner products and L2 regularization, we can take the kernel and L2 norm in the RKHS.

Recall that in linear SVMs, we have used a plug-in classifier on a linear model $F(x) = \beta^T x$[1] and trained it using the hinge loss regularized with the L^2 norm.

$$\operatorname*{argmin}_{\beta \in \mathbb{R}^{d+1}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y^{(i)} F(x^{(i)})\} + \lambda \|\beta\|^2 = \operatorname*{argmin}_{\beta \in \mathbb{R}^{d+1}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y^{(i)} \beta^T x^{(i)}\} + \lambda \|\beta\|^2 \tag{27}$$

Now in the nonparameteric case, all we do is replace F to not be a linear model, but some function in a RKHS $\mathcal{F}$. Therefore, we are minimizing

$$\operatorname*{argmin}_{F \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y^{(i)} F(x^{(i)})\} + \lambda \|\beta\|_{\mathcal{F}}^2 \tag{28}$$

where the L^2 norm of the vector β becomes the norm of the function. So you are just minimizing the hinge loss over an arbitrary RKHS.

This numerically corresponds to replacing every inner product of your data $\langle x^{(i)}, x^{(j)} \rangle$ with the Mercer kernel $K(x^{(i)}, x^{(j)})$.

9.1 Concentration Bounds

The following theorem gives a bit of insight into the bias-variance tradeoff.

Theorem 9.1 (Blanchard, Bosquet, Massert, 2008)

For all F in a RKHS with $\|F\|_K \leq R$, the following is true with probability $1 - \delta$.

$$P(Y \neq h(X)) \leq \left(\frac{1}{n} \sum_i \max\{0, 1 - y^{(i)} F(x^{(i)})\} \right) + \frac{2R}{n} \sqrt{\sum_i K(x^{(i)}, x^{(i)})} + \sqrt{\frac{8 \log(2/\delta)}{n}} \tag{29}$$

It's a little weird that it states that it holds for *all* F in the RKHS, but since it holds for all functions, it also holds for the minimizer $\hat{F}$ as well.

But we also see that there is a tradeoff between bias and variance. If we set R to be large, i.e. we do not regularize as much, then the second term will get large and we get a worse bound (higher complexity and so higher variance), but at the same time we may get a better fit in the first term (lower bias).

If we apply this with a Gaussian kernel $K(x, y) = \exp(-\frac{\|x-y\|}{2\sigma^2})$, then we see that

$$P(Y \neq h(X)) \leq \left(\frac{1}{n} \sum_i \max\{0, 1 - y^{(i)} F(x^{(i)})\} \right) + \frac{R}{\sqrt{\pi n} \sigma} + \sqrt{\frac{8 \log(2/\delta)}{n}} \tag{30}$$

and so we have another parameter σ to tune. We could try to make σ large to make the second term small. This would certainly decrease the variance, but then the first term (the loss) might increase due to high bias.[2]

[1] This means that $f(x) = \text{sign}(F(x))$.

[2] Think again that if we had a nearly uniform kernel over the entire space, then we would have a constant function, which has extremely high bias and low variance.

Bibliography

[CH67] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.