LECTURE 1/25

CS 181

Contents

1 Housekeeping

We have a few policy changes:

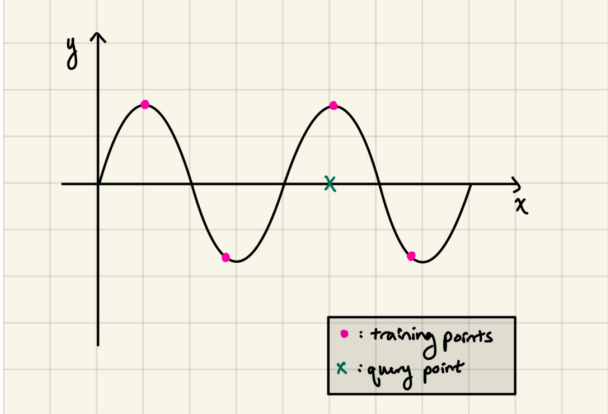
- Lectures are now recorded.
- But we will record attendance through an attendance code for every class: bit.ly/cs181-attendance.

2 Non-Parametric Regression: Epilogue

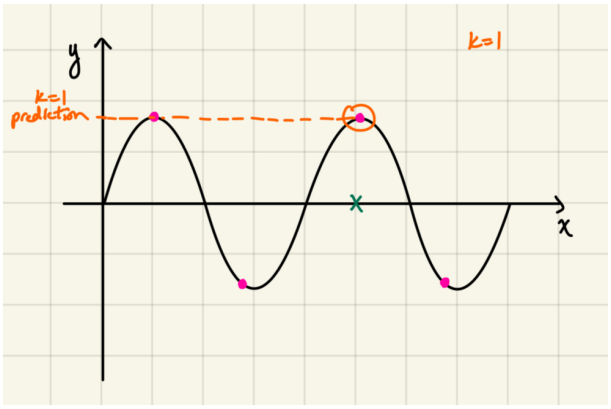
Before moving onto new content, we will wrap up some loose ends from the previous lecture.

kNN and KR are locality adaptive (via k or scale) and "geometry" adaptive. By locality adaptive, we mean that we can adjust how local or global a kNN or KR model is in terms of which points the model is using for its predictions. We control the locality by adjusting k for kNN and adjusting the bandwidth of the kernel for KR. By "geometry" adaptive, we mean that we can alter the geometry of the space. Geometry is used here in a loose sense to refer to how we compare points. In kNN, we have the notion of distance, and we can choose different notions of distance to emphasize certain aspects of the space. For KR, we have the shape of the kernel.

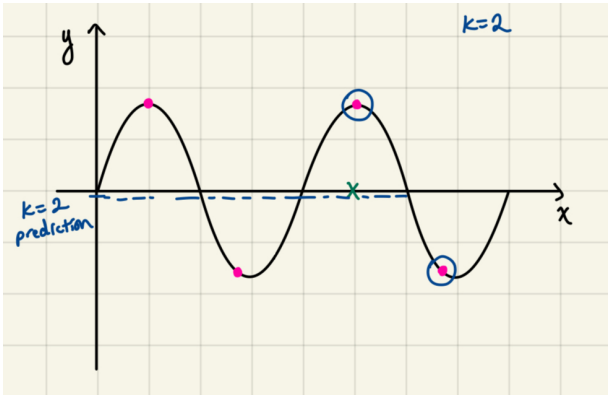
To get at this notion of the difference between a discrete choice of neighbors and a weighted average, let us examine the following example.



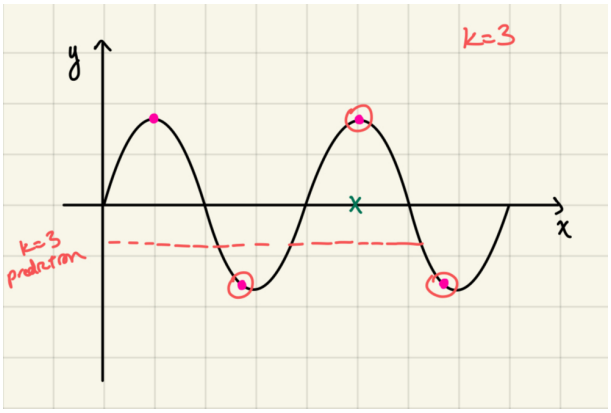
The pink points are in our training dataset and the green marked X is the point we wish to predict. We see that on our sin wave function, the correct value for our query point should be 1 (assuming the maximum of the sin wave goes to 1).



If we do kNN where k = 1, then the prediction would be 1 because the nearest neighbor would be the point circled in orange.

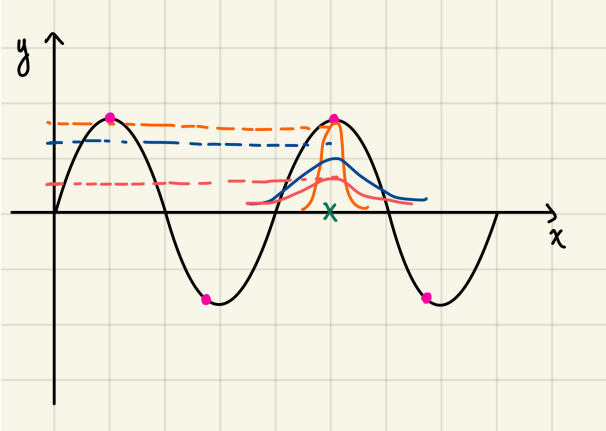


If we do KNN where k = 2, the prediction would be 0 because the two nearest neighbors would be points circled in blue and the average of their y-values is $\frac{1}{2}(1+(-1)) = 0$. This is a pretty bad prediction for the query point.



If we do KNN where k = 3, the prediction would be around -0.33 because the three nearest neighbors would be the circled points and the average of their y-values is $\frac{1}{3}(1+(-1)+(-1)) = -0.33$.

We see that as our k increased, our predictions became worse and worse. Now let us investigate how kernelized regression performs.



The different colors correspond to different kernel functions and the predictions that correlate. We see that as the kernel function becomes flatter, the prediction becomes worse. But at least we stay on the right side of the axisv (as in the prediction will never become negative). Thus, the predictions with kernelized regression seem better than the kNN predictions. So, this is an example of a situation where kNN is unstable.

To generalize this example, we have a highly changing function that is sparsely sampled. The sparse samplings cause the neighbors to be a course measure of prediction because we are taking points from all over the place.

3 Parametric Regression

To motivate both types of regression, let's ask: why do we want to do regression? Regression in general is the problem of predicting a continuous value from a set of predictive variables. So, whenever there is a continuous quantity that we want to predict as a function of other quantities, we use a regression.

Regression is all around us, even in the most unassuming places. For example, in air traffic control, we want to predict arrival times of planes to the gate. We can use information about departure time, the type of airplane, the arrival times of past planes to the gate, etc. We have all sorts of historical information spanning decades that can help us predict. Therefore we can pose a regression problem to predict arrival time. Beyond saving millions of dollars for airline companies, it also saves time for everybody if customers know when they are going to arrive.

Another example is predicting which movies a user would enjoy watching. We can use what they have watched in the past, what scores they gave to past movies, and what their friends have watched and enjoyed. All of these variables can help us predict the score of how much the user likes a new movie. So, these two examples illustrate how regression is useful.

Today, we are going to talk about how to do regression with parametric models. In non-parametric models, there were no weights, no parameters of the models. When the prediction is going to be done with explicit parameters, we call it parametric.

In parametric regression, we are given data $(x^{(n)}, y^{(n)})$ and our goal is to find some function $f(x; w)$ to make our predictions $\hat{y} = f(x; w)$.

Recipe For Parametric Regression Model:

1. Loss Function: We need something to evaluate whether we are doing a good job or not. That is the job of the loss function. Typically we denote the loss function as

$$\mathcal{L}(w)$$

   and we'll write the loss function as a function of weights.

2. Class of Predictive Functions: We have the freedom in choosing the function class. This could be, for example, linear functions, quadratic function, or neural networks.

3. Pick the best parameters, which are the parameters that minimize the loss function. We use a star to denote the optimal parameters.

$$w^* = \min_w \mathcal{L}(w)$$

4 Linear Regression

4.1 Linear Function

Now, let us apply the general recipe to linear regression. First, we satisfy Step 2 by choosing the function to be linear.

$$\hat{y} = w_0 + w_1 x_1 + ... + w_D w_D$$

Notice w_0 in the expression above, which doesn't have a corresponding x_0 value. This is known as the bias term, and it accounts for data that has a non-zero mean. The last term has subscripts D, which represents the number of dimensions. We can write the function in a more compact way using the summation operator.

$$\hat{y} = w_0 + \sum_{d=1}^{D} w_d x_d$$

Then, we can write in an even more compact way by using linear product. We're going to introduce a common notational trick here for making the bias term, w_0, easier to handle. At the moment w_0 is unwieldy because it is not being multiplied by an x_i value. A simple way to make our bias term easier to handle is to simply introduce another variable, x_0, that is always 1 for every data point. Let $\tilde{\mathbf{X}}$ be the matrix after the bias trick.

This bias trick lets us write:

$$\hat{y} = \begin{bmatrix} 1 & x_1 & ... & x_D \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ ... \\ w_D \end{bmatrix} = \tilde{\mathbf{X}}^\top \mathbf{w}$$

This is more compact, easier to reason about, and makes properties of linear algebra nicer for the calculations we will be performing.

So, we have expressed a relationship between one sample and a predicted value. Remember we have n samples, so we will have n equations.

$$\hat{y}^{(1)} = w_0 + \sum_{d=1}^{D} w_d x_d^{(1)}$$
$$\hat{y}^{(2)} = w_0 + \sum_{d=1}^{D} w_d x_d^{(2)}$$
$$...$$
$$\hat{y}^{(n)} = w_0 + \sum_{d=1}^{D} w_d x_d^{(n)}$$

Note that we have one set of weights we are using to predict all the samples so $\mathbf{w}$ is not changing. To write this in compact matrix form, we stack all the y equations to create one matrix.

$$\mathbf{Y} = \begin{bmatrix} 1 & x_1 & ... & x_D \\ 1 & x_1 & ... & x_D \\ ... & & & \\ 1 & x_1 & ... & x_D \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ ... \\ w_D \end{bmatrix} = \mathbf{X}\mathbf{w}$$

4.2 Loss Function

Now, let's go back to our recipe. Step 1 is to define the loss function. Loss function is a notion of how well we are doing. In this case, we can think of it as how well is our model fitting the data that we are trying to fit. In other words, how close is $\hat{y}$ to y where y are the actual values and $\hat{y}$ are our predictions.

One of the most commonly used measurements is known **as least squares or L2 loss**. Least squares, as it is often abbreviated, says that the loss for a given data point is the square of the difference between the target and predicted values.

$$\mathcal{L}_n(\mathbf{w}) = (y_n - \mathbf{w}^\top \mathbf{x}_n)^2$$

Therefore, the loss incurred by parameters $\mathbf{w}$ over our entire data set $\mathbf{X}$ is

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (y_n - \mathbf{w}^\top \mathbf{x}_n)^2.$$

Note that we added a constant $\frac{1}{2}$ to the beginning of our loss expression. This scales the loss, which will not change our final result for the optimal parameters. It has the benefit of making our calculations cleaner once we've taken the gradient of the loss.

We now want to solve for the values of $\mathbf{w}$ that minimize this expression.

5 Method 1: Gradient Descent

We find the optimal weights $\mathbf{w}^*$ as follows:

Start by taking the gradient of the loss with respect to our parameter $\mathbf{w}$:

$$\nabla \mathcal{L}(\mathbf{w}) = \frac{1}{2} \nabla \sum (\mathbf{y}^{(n)} - \tilde{\mathbf{X}}^{\top(n)} \mathbf{w})^2 = \sum (\mathbf{y}^{(n)} - \tilde{\mathbf{X}}^{\top(n)} \mathbf{w})(-\tilde{\mathbf{X}}^{\top(n)})$$

We update $\mathbf{w}$ using the following update function

$$\mathbf{w} = \mathbf{w} - \eta \nabla \mathcal{L}(w)$$

Since we are trying to minimize our weights, we subtract. The symbol η represents our step-size.

6 Method 2: Analytical Solution

Rather than using a gradient update and a step-size, we can solve the optimization analytically.

We will rewrite the loss function.

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^2 = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$= \frac{1}{2}[\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\mathbf{w} + \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w}]$$

Thus, when we apply the gradient, we get

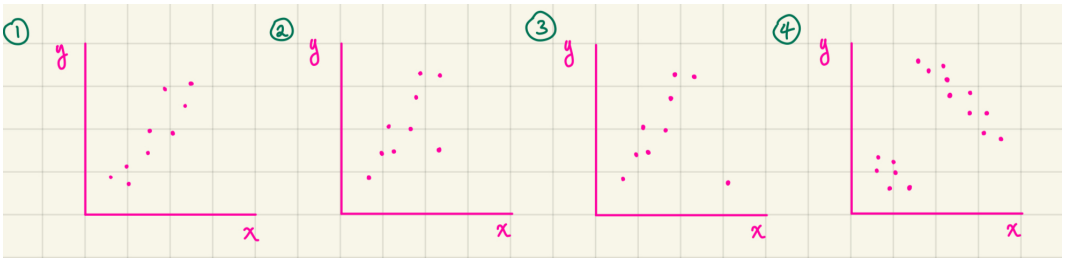
$$\nabla \mathcal{L}(\mathbf{w}) = \frac{1}{2}[-2\mathbf{y}^\top \mathbf{X} + (\mathbf{X}^\top \mathbf{X} + (\mathbf{X}^\top \mathbf{X})^\top)\mathbf{w}].$$

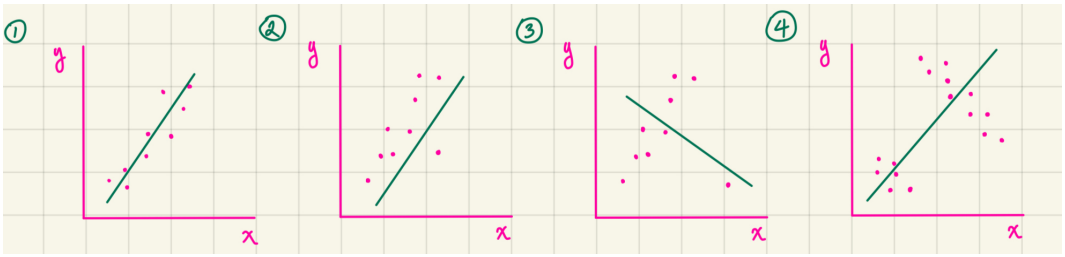
Set the gradient and solve for $\mathbf{w}$. We then get

$$\mathbf{w}^* = \boxed{(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}}.$$

7 Concept Check

What would the linear regression model for look like for each of the following datasets?





1) The line of best fit has a positive slope.

2) The line of best fit still has a positive slope but the outlier pulls the line to the right.

3) When the outlier is very far out, it can mess with the line of best fit, causing the slope to flip.

4) We can see that the linear regression model does not do a good job expressing the dataset. But the best that a linear regression model can do is capture the two clusters by having a positive slope.