May 2, 2018

CS 361: Probability & Statistics

Regression

Regression in practice

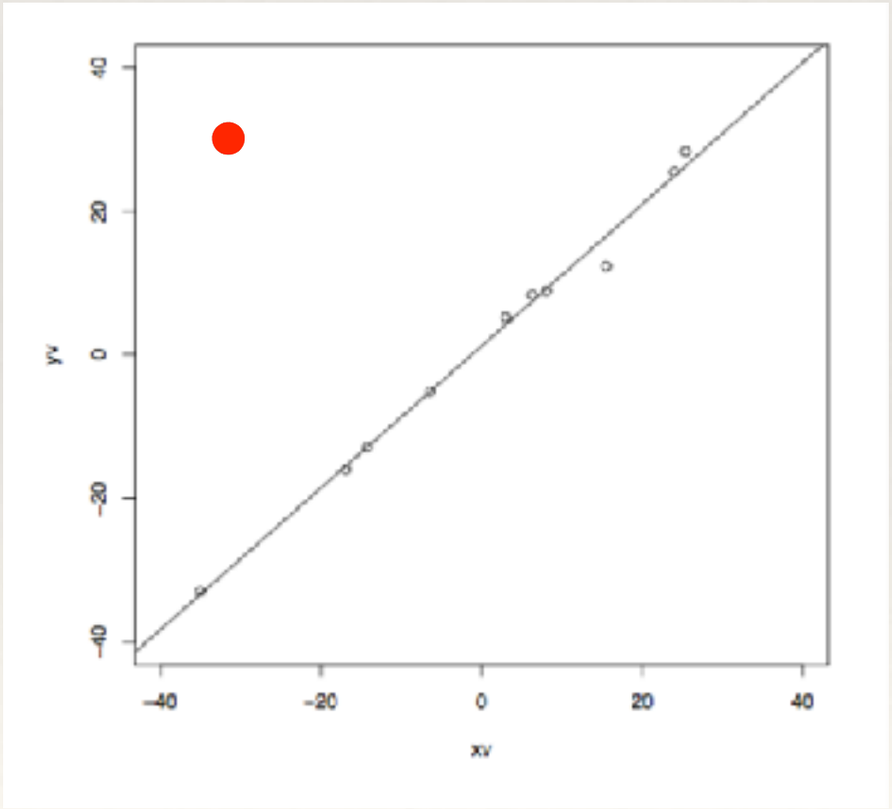
We've learned a procedure for getting a good Beta by considering our training data, let's explore some things that could go wrong

The two big sources of error we will consider are outliers and insufficient explanatory variables

Outliers

Why do outliers produce problems for linear regression?

Suppose we had a nice regression and someone plopped down an outlier point, what would happen?

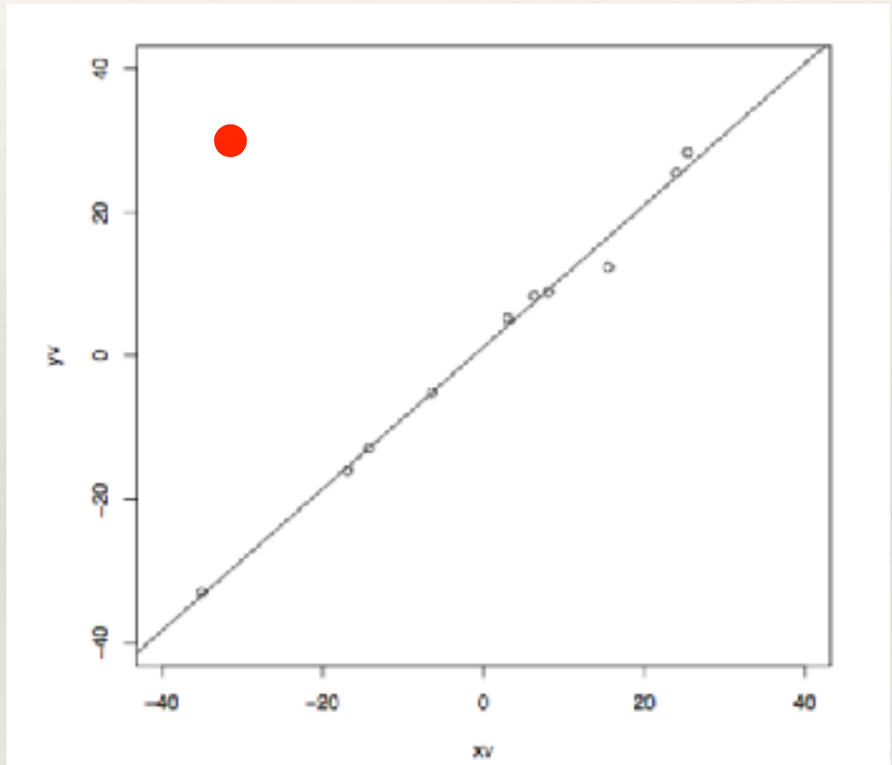


Outliers

Recall the minimization form of our objective function

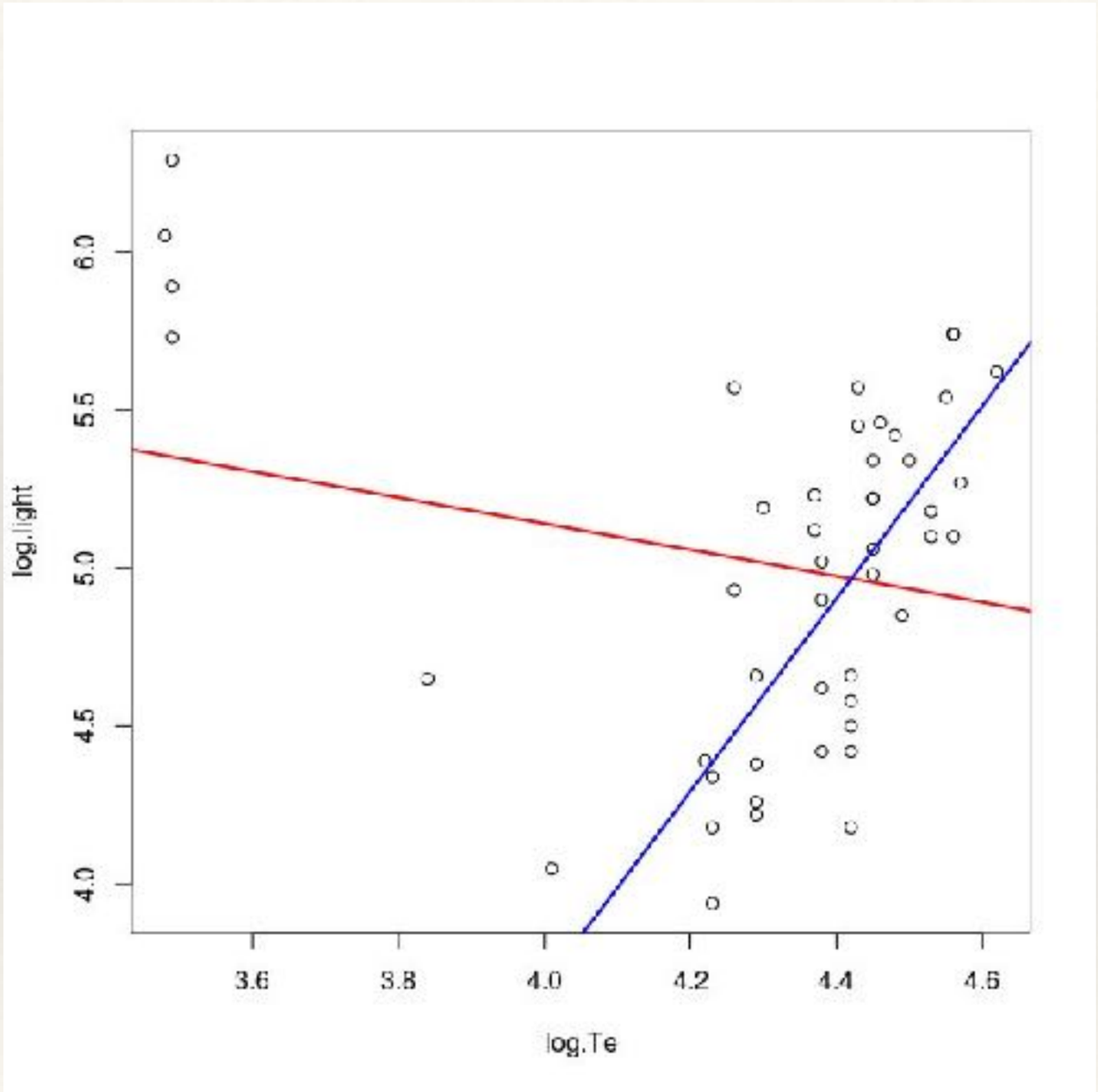
$$\sum_i (y_i - \mathbf{x_i}^T \beta)^2$$

Since we are trying to minimize the squared error over the whole dataset a point that is very far from the line contributes a large amount—squared—to the overall result



In general, many residuals of medium size will have a smaller cost than one large residual with the rest being tiny (e.g., the picture above)

Outliers



Source: user603@stackoverflow

While we would probably desire the blue line, the red line would be the best fit line for this dataset

Insufficient explanatory variables

Even if we have no outliers, we could still fail to get a good regression simply because our data isn't actually linear

It's easy enough to use what we have learned so far to deal with data that follows some other kind of curve by simply transforming our data

In particular, if we are trying to model a dataset where we have only one explanatory variable and upon plotting the data we see that the data appears to be better fit by a cubic than by a line, we can create a new dataset where each data point is now given by

$$x_i' = \begin{bmatrix} x_i & x_i^2 & x_i^3 \end{bmatrix}$$

Insufficient explanatory variables

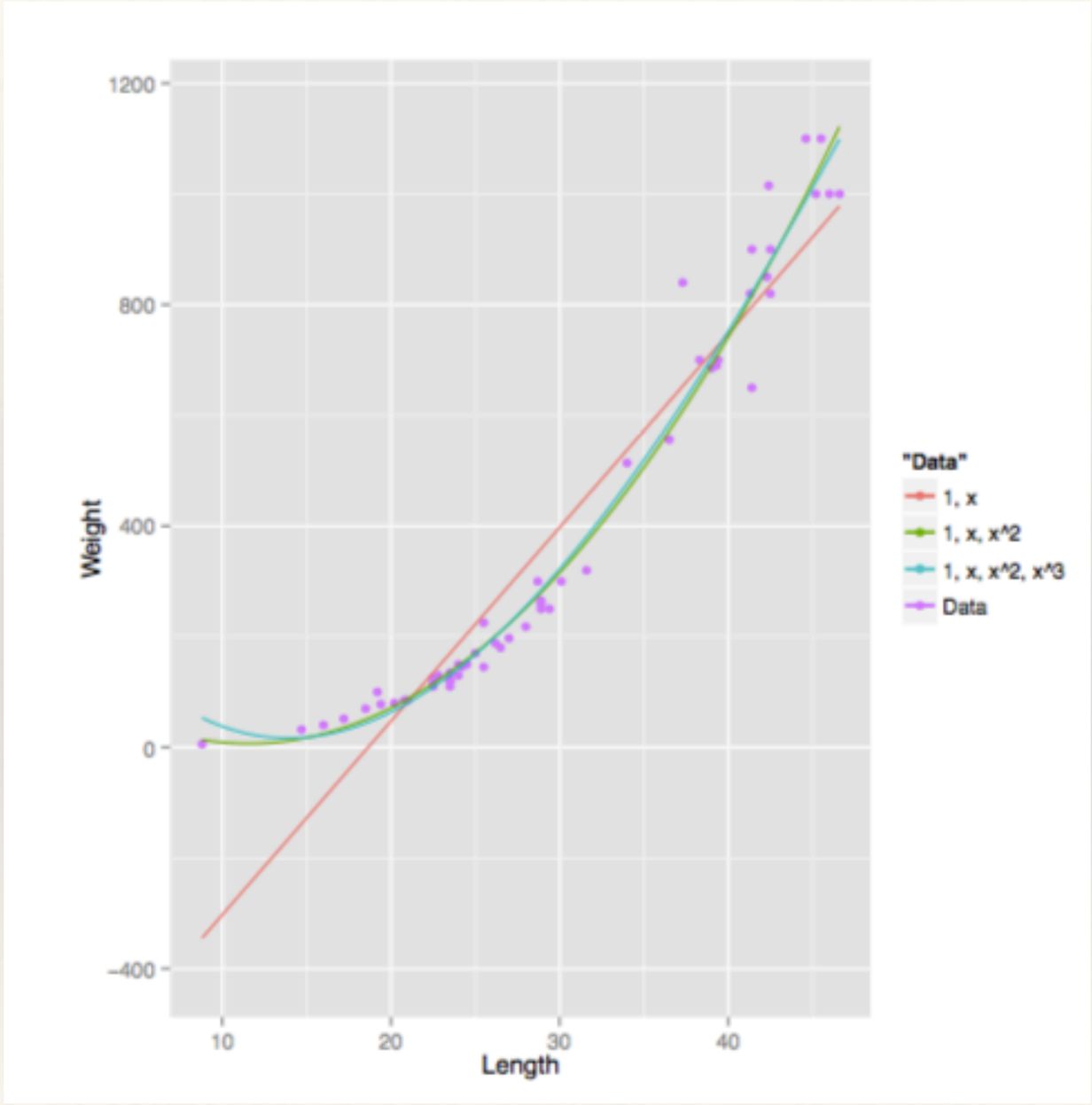
Prior to the transformation, regression would produce a function of the form

$$y = \beta_1 x + \beta_0$$

Applying the transformation and doing regression gives us a function of the form

$$y = \beta_3 x^3 + \beta_2 x^2 + \beta_1 x + \beta_0$$

Adding explanatory variables



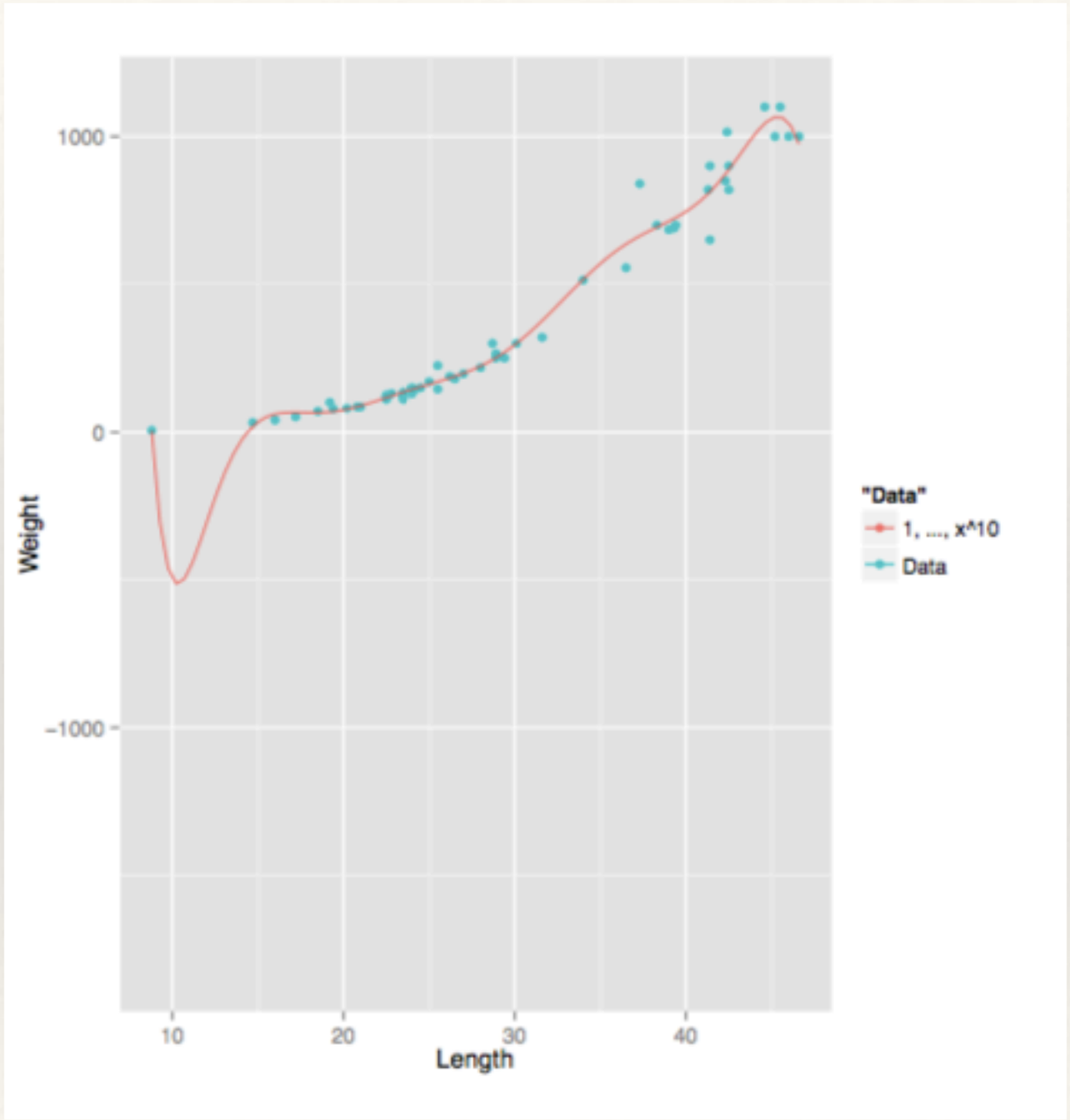
Insufficient explanatory variables

Adding explanatory variables in this way can never cause our error to increase, so why don't we just keep adding them until we perfectly fit our data? (which would take at most N-1 of these kinds of features)

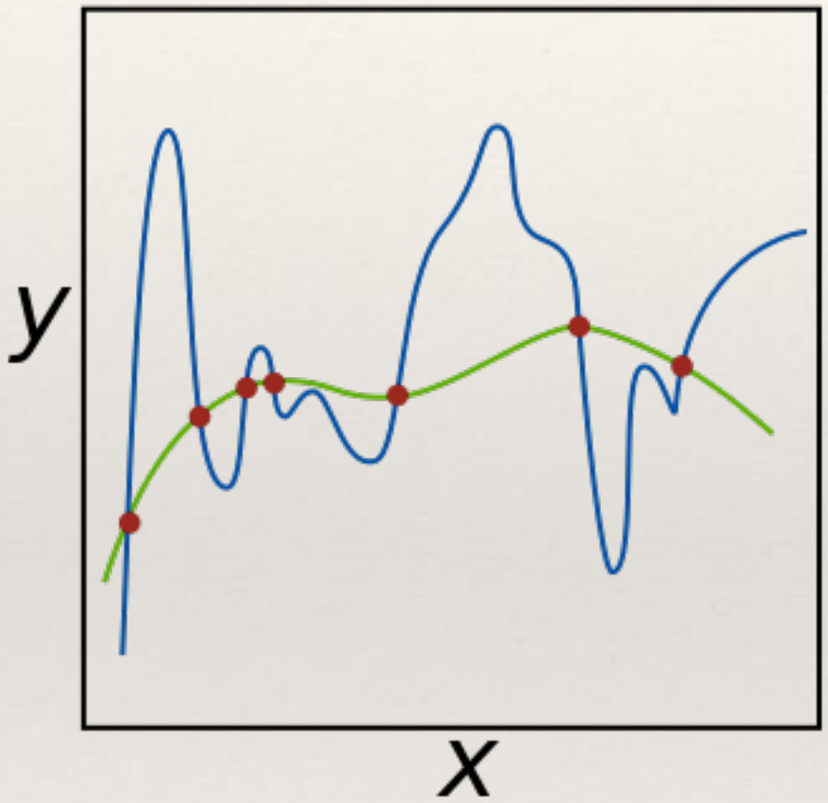
The answer is that it begins to hurt generalizability: doing this can make us do arbitrarily well on our training data while degrading our performance on data we have not yet seen

The coefficients that we learn for this polynomial will usually have some large positive and negative coefficients so that the curve fits the data in just the right way, but if the data were changed by a small amount, the corresponding set of coefficients needs to change a good deal to fit the data. This instability is a bad sign

Adding too many variables



Adding too many variables



Source: wikipedia

Both have zero loss, how do we come up with a procedure that prefers the green curve?

Regularization

Perhaps we could keep a validation set and only use the number of polynomials that still performs well on held-out data

What we will do instead, though, is similar to what we did for classification. We will penalize complexity—we penalize large coefficients—with regularization

You can think of small coefficients as penalizing complexity by looking at the limiting case. beta=0 would be a straight line with no slope, which would probably do poorly making predictions but would be quite a simple function

Regularization

Instead of minimizing

$$\sum_i (y_i - \mathbf{x_i}^T \beta)^2$$

We minimize

$$\sum_i (y_i - \mathbf{x_i}^T \beta)^2 + \lambda \sum_j \beta_j^2$$

Or in the matrix formulation, we minimize

$$(\mathbf{y} - X\beta)^T (\mathbf{y} - X\beta) + \lambda \beta^T \beta$$

Regularization

Solving the gradient with respect to beta for

$$(\mathbf{y} - X\beta)^T(\mathbf{y} - X\beta) + \lambda\beta^T\beta$$

Yields a solution of

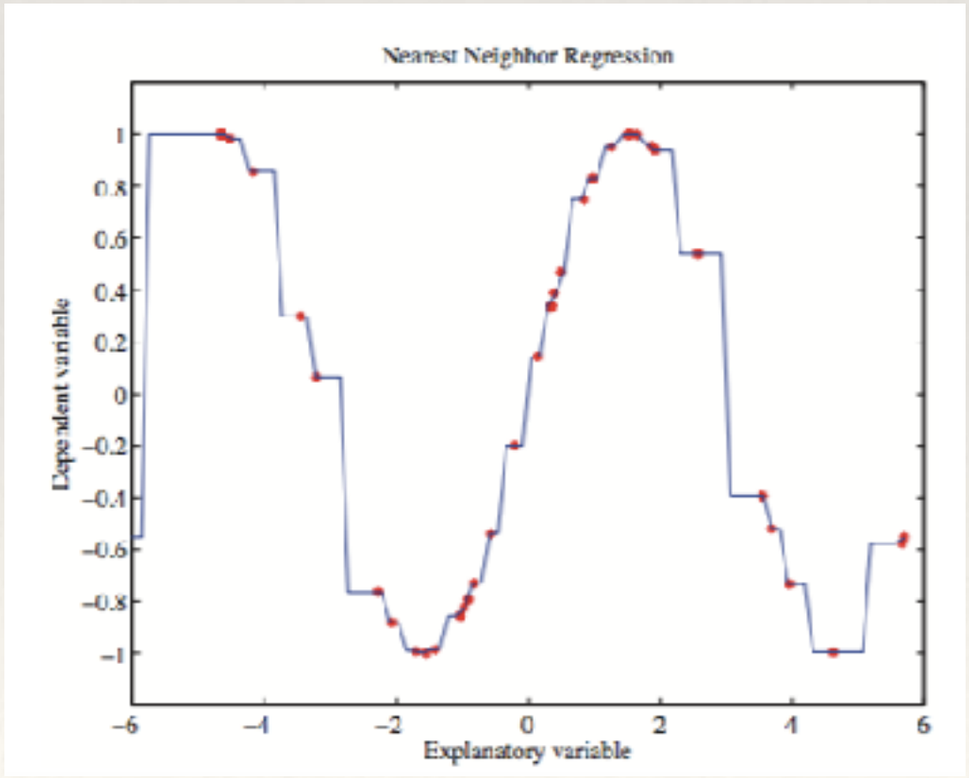
$$\beta = (X^T X + \lambda\mathcal{I})^{-1} X^t y$$

Nearest neighbors for regression

Nearest neighbors

Just like with classification, there is nothing keeping us from storing our training data and using it directly to make predictions on held-out or unseen data

One option would be to just figure out which point from the training set is nearest to a query point and return its value. This would produce a piecewise-constant function



Nearest neighbors

Another option is to consider the k nearest neighbors to a given query point. The question is how to combine the k values of y into a prediction

The query point is probably closer to some of the points than the others, so it makes sense to combine the k nearest neighbors in a weighted sum, perhaps weighted by distance

$$y_{pred} = \frac{\sum_i w_i y_i}{w_i} \qquad \text{with} \qquad w_i = \| x - x_i \|$$

Nearest neighbors

Another option for the weighting is to have some kind of exponential decay for distance points, as follows

$$y_{pred} = \frac{\sum_i w_i y_i}{w_i} \qquad \text{with} \qquad w_i = \exp\left(\frac{-\|x - x_i\|^2}{2\sigma^2}\right)$$

Where the sigma would be a parameter that we could set ourselves by using a validation set to find a good value

Nearest neighbors