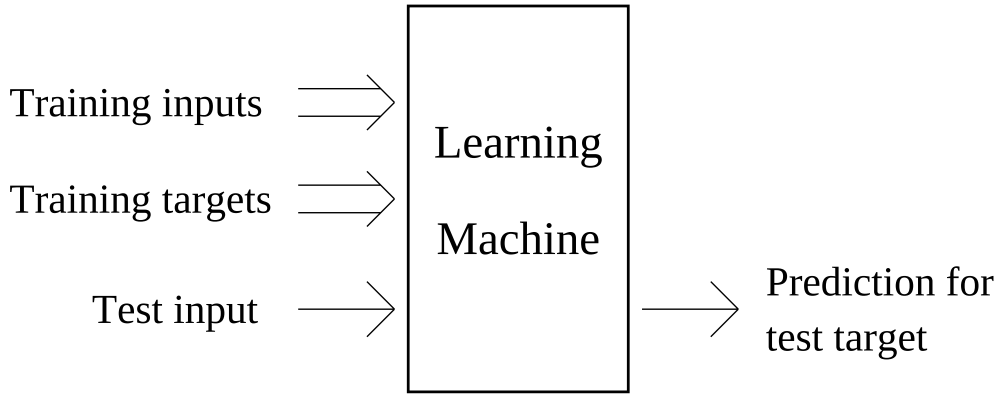
A "Supervised Learning Machine"

Here's the most general view of how a "learning machine" operates for a supervised learning problem:



Note that, conceptually, our goal is to make a prediction for just one test case. In practice, we usually make predictions for many test cases, but in this formulation, these predictions are separate (though often we'd compute some things just once and then use them for many test cases).

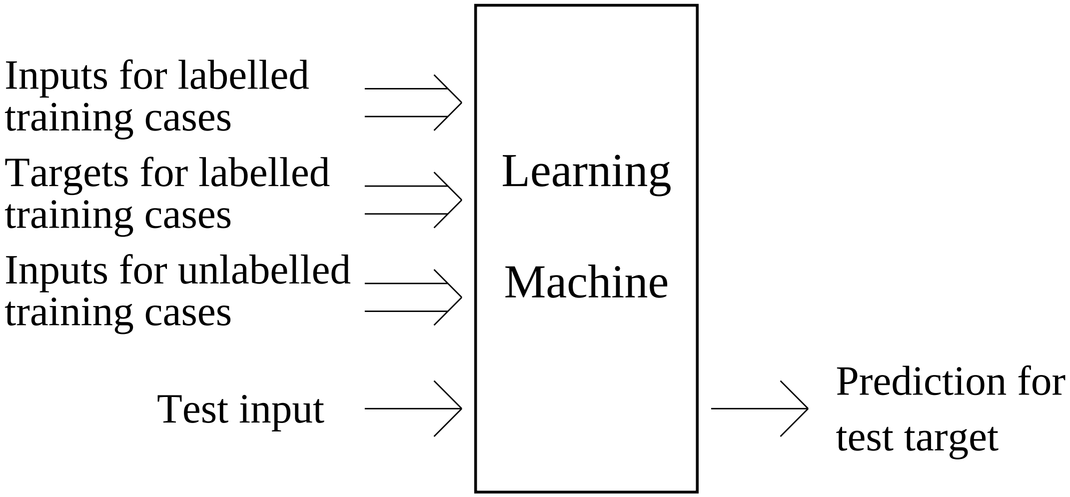
Any sort of supervised learning procedure *can* be viewed in this "mechanical" way, but is this a useful view?

A more statistical approach is to think in terms of a *model* that we choose based on our knowledge of the situation.

A "Semi-Supervised Learning Machine"

Recently, a variation on supervised learning has attracted lots of interest.

For *semi-supervised* learning we have both some *labelled* training data (where we know both the inputs and targets) and also some *unlabelled* training data (where we know only the inputs). Here's a picture:



This can be very useful for applications like document classification, where it's easy to get lots of unlabelled documents (eg, off the web), but more costly to have someone manually label them.

Notation for Supervised Learning

We want to learn how to predict a *target* variable (also called the *output* or *response*), which we call y.

To help us predict y, we have measurements of p *input* variables (also called *features* or *predictors*), denoted by $x_1, \ldots, x_p$. The collection of all p inputs will be denoted by x, which is sometimes treated as a column vector.

We have a *training set* of n cases, for which we know both inputs and targets. We refer to the inputs for these cases as $x^{(1)}$, $x^{(2)}$, $\ldots$, $x^{(n)}$ and to the targets for these cases as $y^{(1)}$, $y^{(2)}$, $\ldots$, $y^{(n)}$.

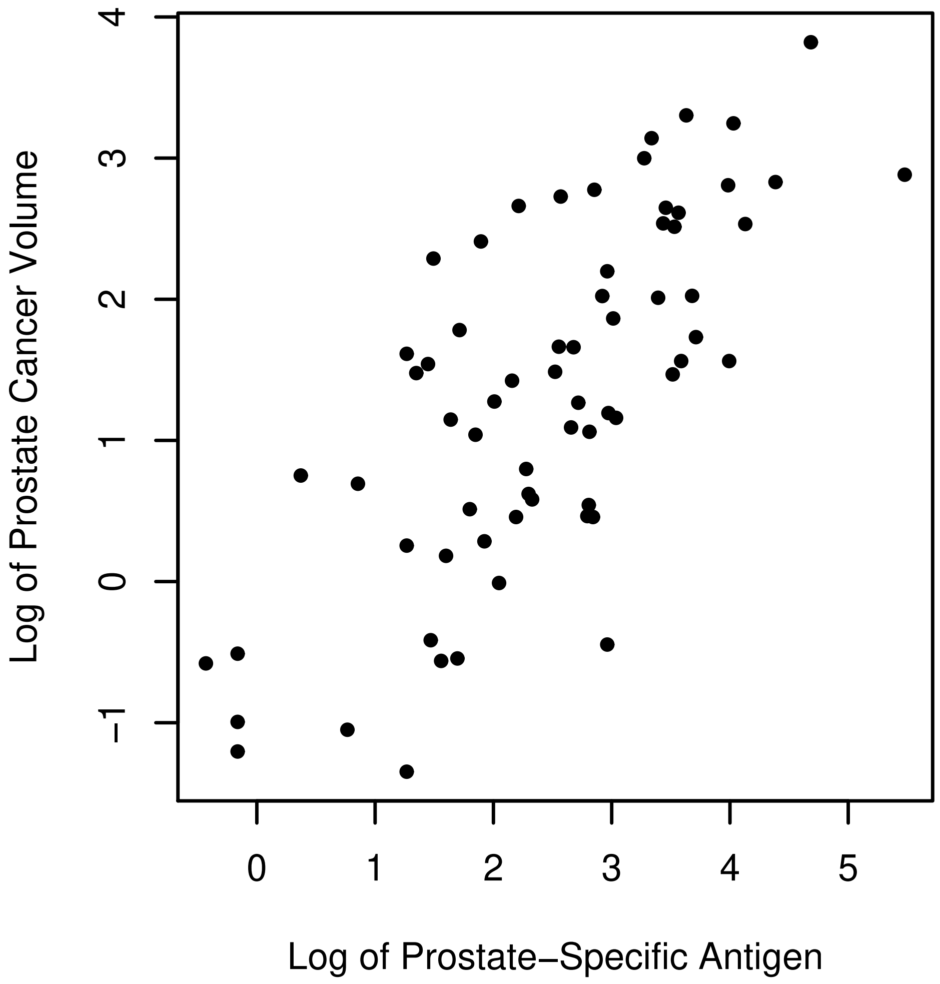
So, for example, $x_4^{(3)}$ is the fourth input for the third training case.

Using this training data, for a *test case* with inputs x^*, we would like to predict the value of the response, y^*. We'll sometime write our prediction as $\hat{y}^*$, or as $\hat{y}(x^*)$ to emphasis its dependence on the test inputs.

A Simple Example — Prostate Cancer

Here is some data from a study of prostate cancer. Nine variables were measured for each patient, but we'll look at only two. We'll try to predict the log of cancer volume (y) from the log of the PSA level (x). The training data available is plotted on the right.

Example and data taken from Hastie, Tibshirani, and Friedman, *The Elements of Statistical Learning*, p. 47.



The k-Nearest-Neighbors (k-NN) Method

A simple way to make predictions is to just look at the training cases whose inputs are "near" the inputs for the test case. We might then average the target values for these nearby training cases to get our prediction for the target in the test case.

How many training cases should we look at? Suppose we look at the nearest k.

We also need to decide how to measure "nearness" of input vectors (x). We might use Euclidean distance, or we might use "Manhattan" distance (sum of absolute values of coordinate differences).

We can express the k-NN method as follows:

$$\hat{y}(x^*) \;=\; \frac{1}{k} \sum_{i \in N_k(x^*)} y^{(i)}$$

where $N_k(x^*)$ is the set of k training cases with inputs closest to x^*.

Big question: How should we choose k?

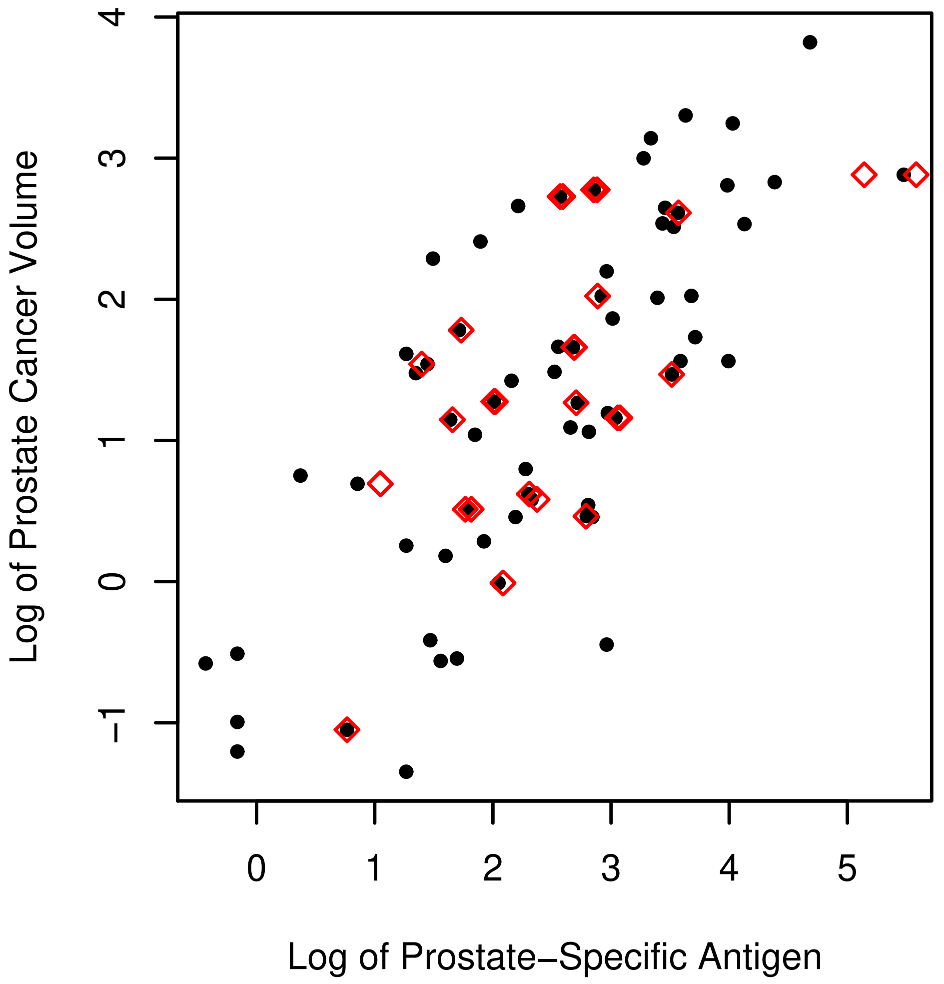
If k is too small, we may "overfit" — pay too much attention to chance variation. But if k is too big, we will average over training cases that aren't relevant to the test case.

1-NN for the Prostate Cancer Example

Here is the result of using 1-NN for the prostate cancer example.

The plot on the right shows the training cases (black dots) and the predictions for test cases (red diamonds).

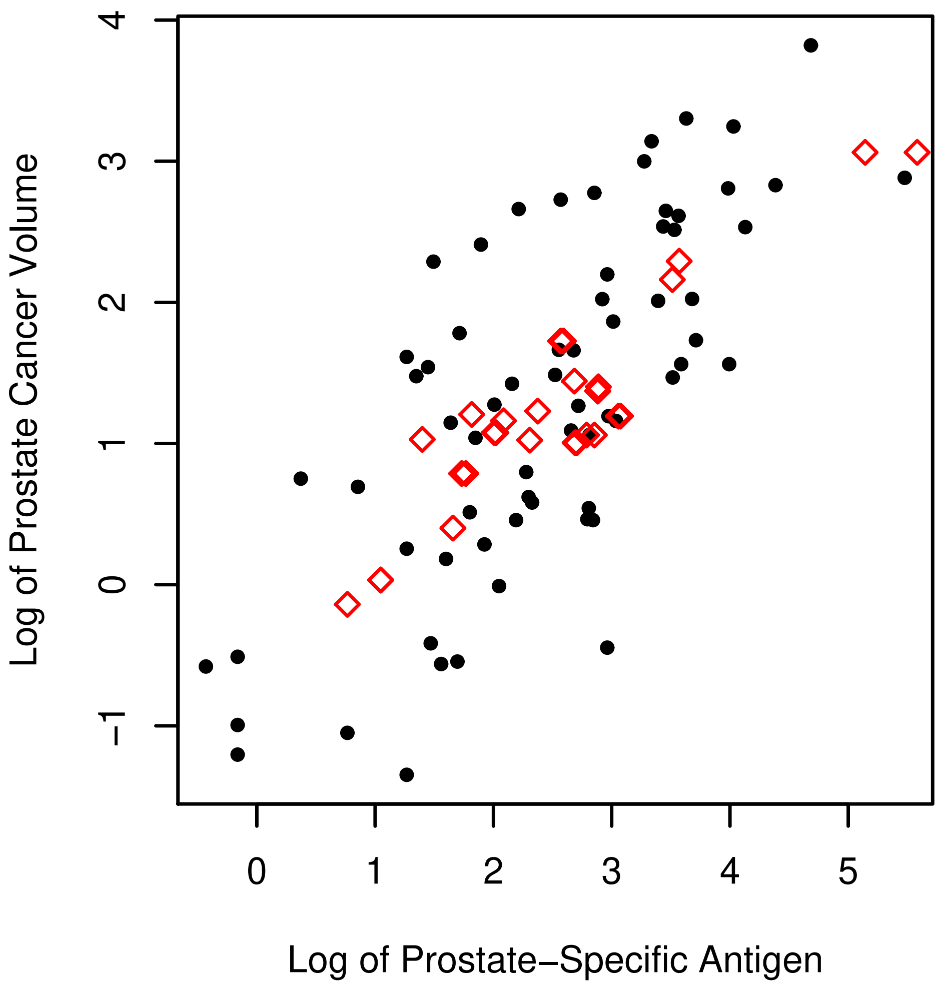
Note how variable the predictions are. We probably don't believe that these variations are real.



5-NN for the Prostate Cancer Example

Here is the result of using 5-NN for the prostate cancer example.

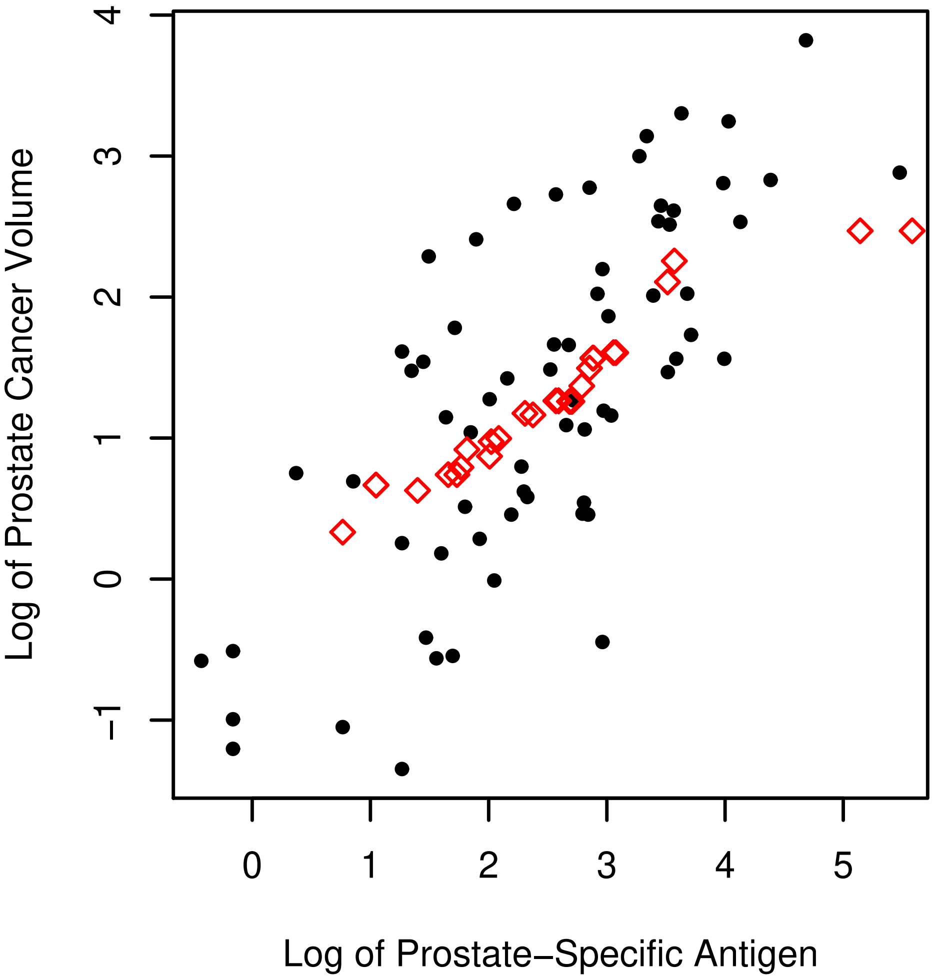
The predictions are less variable than we saw with 1-NN, due to the effect of averaging.



20-NN for the Prostate Cancer Example

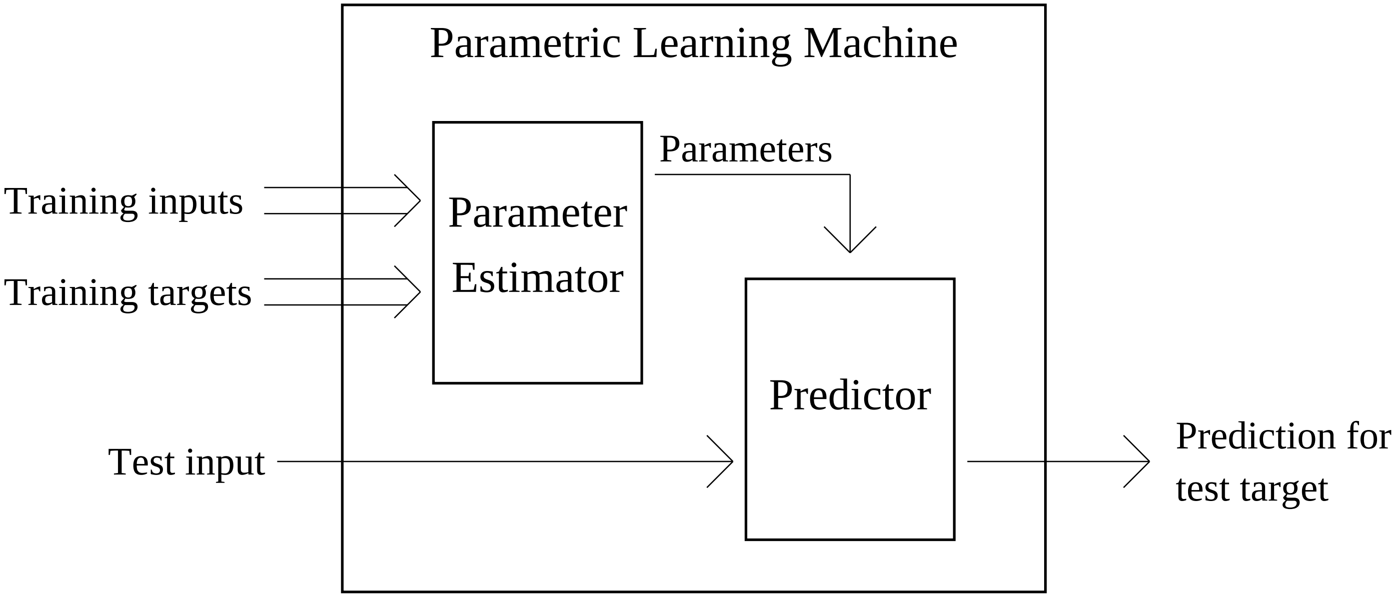
Here is the result of using 20-NN for the prostate cancer example.

Predictions are now even less variable than for 5-NN, but they seem to be systematically wrong at the two ends.



Parametric Learning Machines

One way a learning machine might work is by using the training data to estimate *parameters*, and then using these parameters to make predictions for the test case. Here's a picture:



This approach saves computation if we make predictions for many test cases. We can estimate the parameters just once, then use them many times.

A mixed strategy: Estimate some parameters (eg, k for a nearest-neighbor method), but have the predictor look at the training inputs and targets as well.

Linear Regression

One of the simplest parametric approaches is *linear regression*.

The predictor for this method takes parameter estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$, and produces a prediction by the formula

$$\hat{y}(x^*) \;=\; \hat{\beta}_0 \,+\, \sum_{j=1}^{p} x_j^* \hat{\beta}_j$$

For this to make sense, the inputs and response need to be numeric, but binary variables can be coded as 0 and 1 and treated as numeric. (If our predictions must be either 0 or 1, we threshold $\hat{y}(x^*)$ at 1/2.)

The traditional parameter estimator for linear regression is *least squares* — use $\hat{\beta}_0$, $\hat{\beta}_1$, $\ldots$, $\hat{\beta}_p$ that minimize squared error on the n training cases, defined as

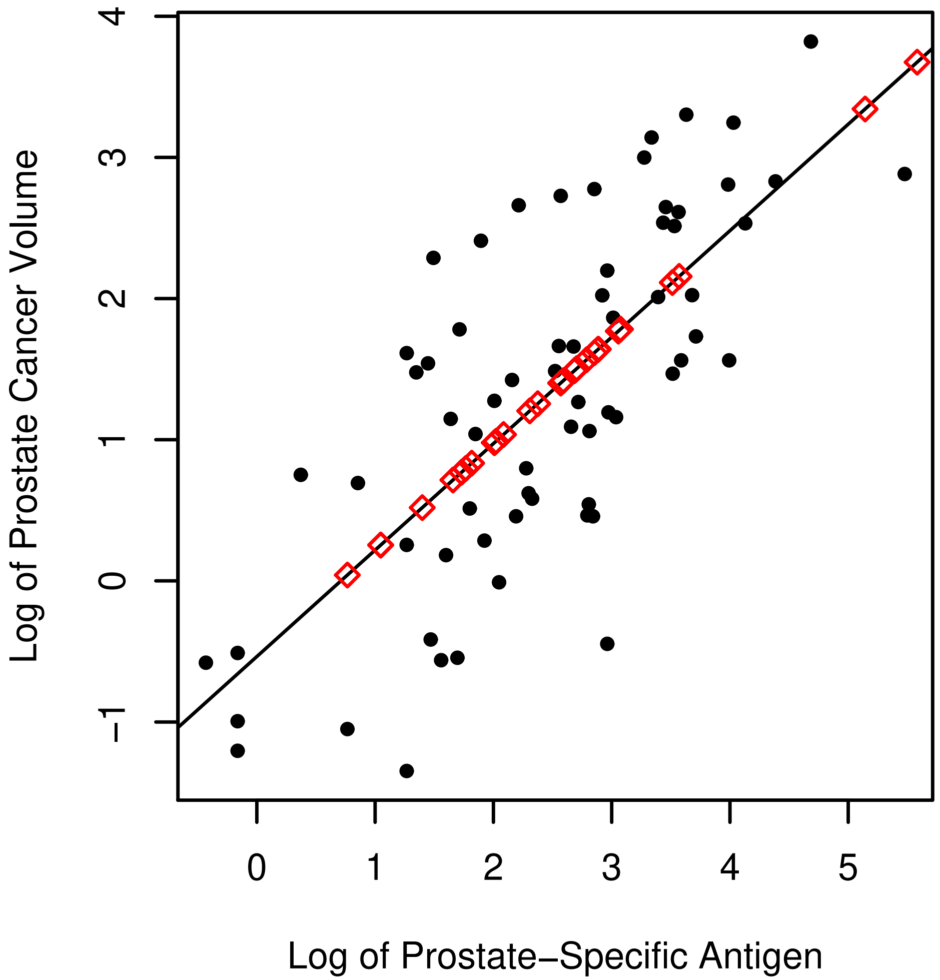
$$\text{RSS}(\beta) \;=\; \sum_{i=1}^{n} \left(y^{(i)} - \left(\beta_0 \,+\, \sum_{j=1}^{p} x_j^{(i)} \beta_j \right) \right)^2$$

The $\hat{\beta}$ that minimizes this is easily found using matrix operations.

Linear Regression for the Prostate Cancer Example

Here is the result of using least-squares linear regression for the prostate cancer example.

The parameter values found were $\hat{\beta}_0 = -0.54$ and $\hat{\beta}_1 = 0.75$. The line defined by $y = \hat{\beta}_0 + x\hat{\beta}_1$ is shown on the plot to the right, along with with the training cases (black dots) and the predictions for test cases (red diamonds).



Linear Regression Versus Nearest Neighbor

These two methods are opposities with respect to computation:

Nearest neighbor is a *memory-based* method — we need to remember the whole training set.

Linear regression is *parametric* — after finding $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ we can forget the training set and use just these parameters.

They are also opposites with respect to statistical properties:

Nearest neighbor makes *few assumptions* about the data, but consequently has a high potential for overfitting, if k is small. If k is big, predictions may be biased.

Linear regression make *strong assumptions* about the data, and consequently has a high potential for bias, if these assumptions are wrong.