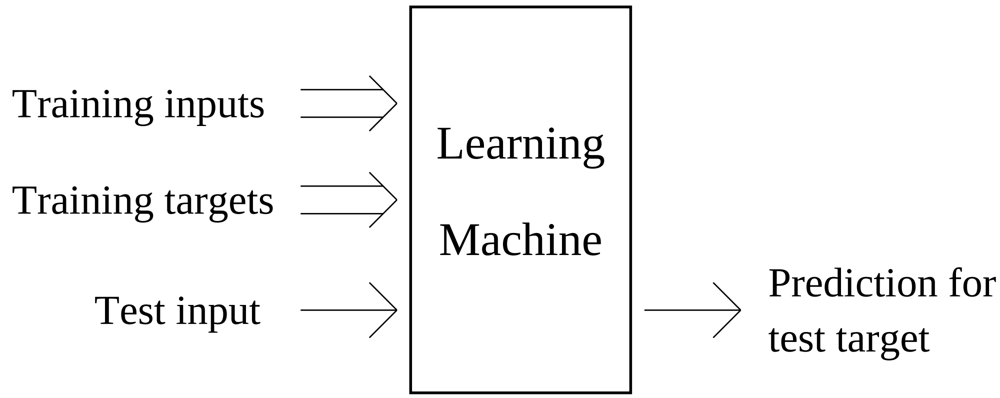
A “Supervised Learning Machine”

Here’s the most general view of how a “learning machine” operates for a supervised learning problem:



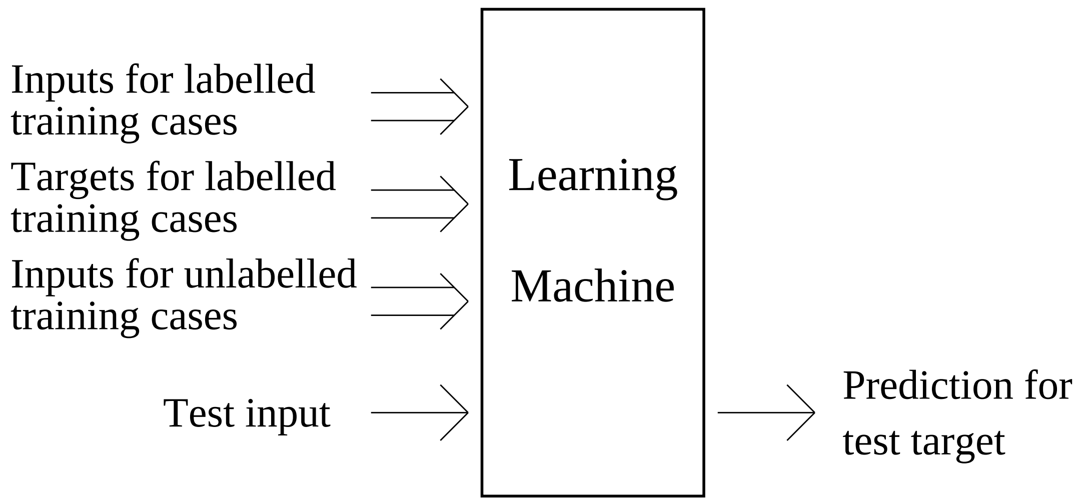
Any sort of statistical procedure for this problem *can* be viewed in this “mechanical” way, but is this a useful view? It does at least help clarify the problem. . .

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).

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.

Data Notation for Supervised Learning

We want to predict a *response* variable (also called the *output* or *target*), 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 seen as a column vector.

We have a *training set* of n cases in which we know the values of both Y and $X = (X_1, \ldots, X_p)$. The text uses lower-case letters with subscripts for the values of Y and X in these training cases. So,

> $y_1, \ldots, y_n$ are the values of the response variable in the n training cases.
>
> $x_1, \ldots, x_n$ are the vectors of inputs for the n training cases.
>
> x_{ij} is the value of the j'th input for the i'th training case.

Note: This doesn't follow the usual random variable notation for upper/lower case, in which x_j would be a specific value for X_j. Here X_j is a single number, but x_j is a vector of p numbers!

Notation for Predictions for Supervised Learning

Given a *test case*, with inputs x, we would like to predict the value of the response, y.

Ideally, we would produce a probability distribution, $P(y \,|\, x)$, as our prediction. (I'll be using $P(\cdot)$ for either probabilities or probability densities.)

But suppose we need to make a single guess, called $\hat{y}$.

For a real-valued response, we might set $\hat{y}$ to the *mean* of this predictive distribution (which minimizes expected squared error) or to the *median* (which minimizes expected absolute error).

For a categorical response, we might set $\hat{y}$ to the *mode* of the predictive distribution (which minimizes the probability of our making an error).

Sometimes, errors in one direction are worse than in the other — eg, failure to diagnose cancer may be worse than mistakenly diagnosing it (leading to further tests). Then we should choose $\hat{y}$ to minimize the expected value of an appropriate *loss function*.

Nearest-Neighbor Methods

A direct approach to making predictions is to approximate the mean, median, or mode of $P(y\,|\,x)$ by the sample mean, median, or mode for a subset of the training cases whose inputs are "near" the test inputs.

We need to decide how big a subset to use — one possibility is to always use the k nearest training points. We also need to decide how to measure "nearness". If the inputs are numeric, we might just use Euclidean distance.

If Y is real-valued, and we want to make the mean prediction, this is done as follows:

$$\hat{Y}(x) \;=\; \frac{1}{k} \sum_{i \in N_k(x)} y_i$$

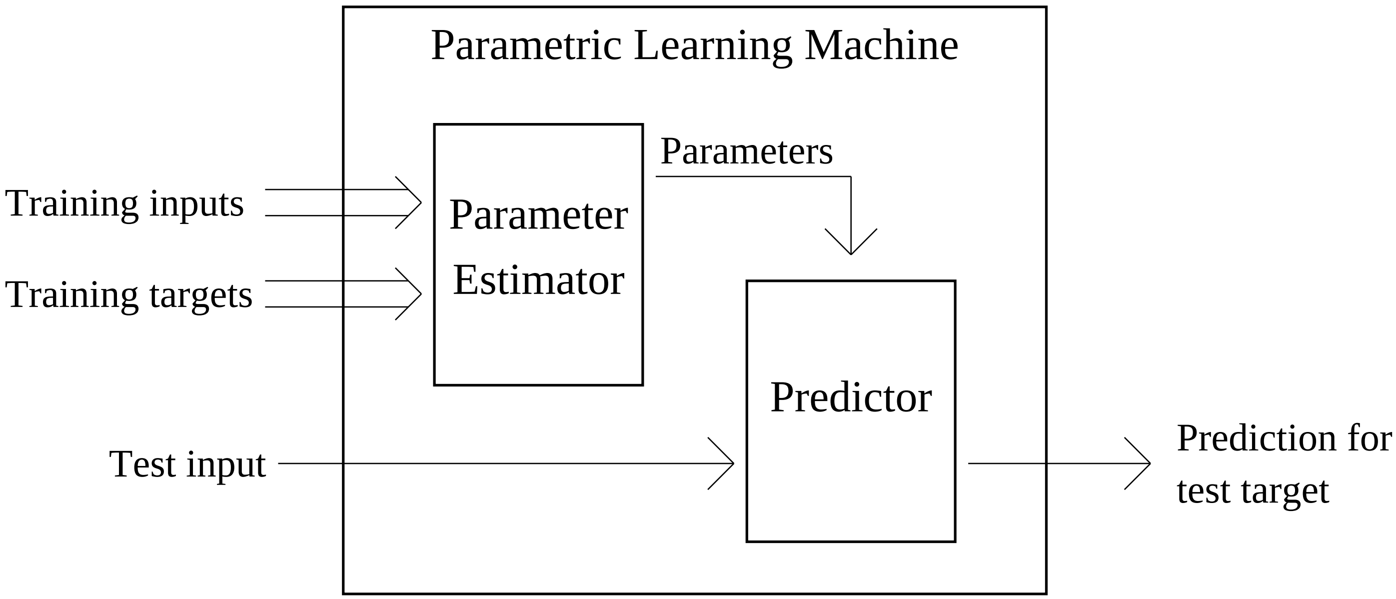
where $N_k(x)$ is the set of k training cases whose inputs are closest to the test inputs, x. (We'll ignore the possibility of ties.)

Big question: How should we choose k?

If k is too small, we may "overfit" (see Figure 2.3), but if k is too big, we will average over training cases that aren't relevant to the test case.

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 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 $\hat{Y}$ 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 the square error on the training cases, defined as

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

The $\hat{\beta}$ that minimizes this is easily found using matrix operations (see the text for details).

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*.

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