STA 414/2104

Statistical Methods for Machine Learning and Data Mining

Radford M. Neal, University of Toronto, 2012

Week 1

What are Machine Learning and Data Mining?

Typical Machine Learning and Data Mining Problems

Document search:

Given counts of words in a document, determine what its topic is.

Group documents by topic without a pre-specified list of topics.

Many words in a document, many, many documents available on the web.

Cancer diagnosis:

Given data on expression levels of genes, classify the type of a tumor.

Discover categories of tumors having different characteristics.

Expression levels of many genes measured, usually for only a few patients.

Marketing:

Given data on age, income, etc., predict how much each customer spends.

Discover how the spending behaviours of different customers are related.

Fair amount of data on each customer, but messy (eg, missing values).

May have data on a very large number of customers (millions).

Supervised Learning Problems

In the ML literature, a *supervised learning* problem has these characteristics:

> We are primarily interested in prediction.
> We are interested in predicting only one thing.
> The possible values of what we want to predict are specified, and we have some *training* cases for which its value is known.

The thing we want to predict is called the *target* or the *response variable*.

For a *classification* problem, we want to predict the class of an item — the topic of a document, the type of a tumor, whether a customer will purchase a product.

For a *regression* problem, we want to predict a numerical quantity — the amount a customer spends, the blood pressure of a patient, the melting point of an alloy.

To help us make our predictions, we have various *inputs* (also called *predictors* or *covariates*) — eg, gene expression levels for predicting tumor type, age and income for predicting amount spent. We use these inputs, but don't try to predict them.

Unsupervised Learning Problems

For an *unsupervised learning* problem, we do not focus on prediction of any particular thing, but rather try to find interesting aspects of the data.

One non-statistical formulation: We try to find *clusters* of similar items, or to *reduce the dimensionality* of the data.

Examples: We may find clusters of patients with similar symptoms, which we call "diseases". We may find that an overall "inflation rate" captures most of the information present in the price increases for many commodities.

One statistical formulation: We try to learn the *probability distribution* of all the quantities, often using *latent* (also called *hidden*) variables.

These formulations are related, since the latent variables may identify clusters or correspond to low-dimensional representations of the data.

Machine Learning and Data Mining Problems Versus Problems in Traditional Statistics

Motivations:

Prediction · Understanding · Causality

Much traditional statistics is motivated primarily by showing that one factor causes another (eg, clinical trials). Understanding comes next, prediction last.

In machine learning and data mining, the order is usually reversed — prediction is most important.

Amount of Data:

Many machine learning problems have a large number of variables — maybe 10,000, or 100,000, or more (eg, genes, pixels). Data mining applications often involve very large numbers of cases — sometimes millions.

Complex, non-linear relationships:

Traditional statistical methods often assume linear relationships (perhaps after simple transformations), or simple distributions (eg, normal).

Attitudes in Machine Learning and Data Mining Versus Attitudes in Traditional Statistics

Despite these differences, there's a big overlap in problems addressed by machine learning and data mining and by traditional statistics. But attitudes differ...

Machine learning	**Traditional statistics**
No settled philosophy or widely accepted theoretical framework.	Classical (frequentist) and Bayesian philosophies compete.
Willing to use *ad hoc* methods if they seem to work well (though appearances may be misleading).	Reluctant to use methods without some theoretical justification (even if the justification is actually meaningless).
Emphasis on automatic methods with little or no human intervention.	Emphasis on use of human judgement assisted by plots and diagnostics.
Methods suitable for many problems.	Models based on scientific knowledge.
Heavy use of computing.	Originally designed for hand-calculation, but computing is now very important.

How Do "Machine Learning" and "Data Mining" Differ?

These terms are often used interchangeably, but...

"Data mining" is more often used for problems with very large amounts of data, where computational efficiency is more important than statistical sophistication — often business applications.

"Machine learning" is more often used for problems with a flavour of artificial intelligence — such as recognition of objects in visual scenes, or robot navigation.

The term "data mining" was previously used in a negative sense — to describe the misguided statistical procedure of looking for many, many relationships in the data until you finally find one, but one which is probably just due to chance.

One challenge of data mining is to avoid doing "data mining" in this sense!

Some Challenges for Machine Learning

Handling complexity: Machine learning applications usually involve many variables, often related in complex ways. How can we handle this complexity without getting into trouble?

Optimization and integration: Most machine learning methods either involve finding the "best" values for some parameters (an optimization problem), or averaging over many plausible values (an integration problem). How can we do this efficiently when there a great many parameters?

Visualization: Understanding what's happening is hard when there are many variables and parameters. 2D plots are easy, 3D not too bad, but 1000D?

All these challenges are greater when there are many variables or parameters — the so-called "curse of dimensionality". But more variables also provide more information — a blessing, not a curse.

Ways of Handling Complexity

Complex Problems

Machine learning applications often involve complex, unknown relationships:

Many features may be available to help predict the response (but some may be useless or redundant).

Relationships may be highly non-linear.

We don't have a theoretical understanding of the problem, which might have helped limit the forms of possible relationships.

Example: Recognition of handwritten digits.

Consider the problem of recognizing a handwritten digit from a 16×16 image. There are 256 features, each the intensity of one pixel.

The relationships are very complex and non-linear. Knowing that a particular pixel is black tells you something about what the digit is, but much of the information comes only from looking at more than one pixel simultaneously.

People do very well on this task, but how do they do it? We have some understanding of this, but not enough to easily mimic it.

How Should We Handle Complexity?

Properly dealing with complexity is a crucial issue for machine learning.

Limiting complexity is one approach — use a model that is complex enough to represent the essential aspects of the problem, but that is not so complex that *overfitting* occurs.

Overfitting happens when we choose parameters of a model that fit the data we have very well, but do poorly on new data.

Reducing dimensionality is another possibility. Perhaps the complexity is only apparent — really things are much simpler if we can find out how to reduce the large number of variables to a small number.

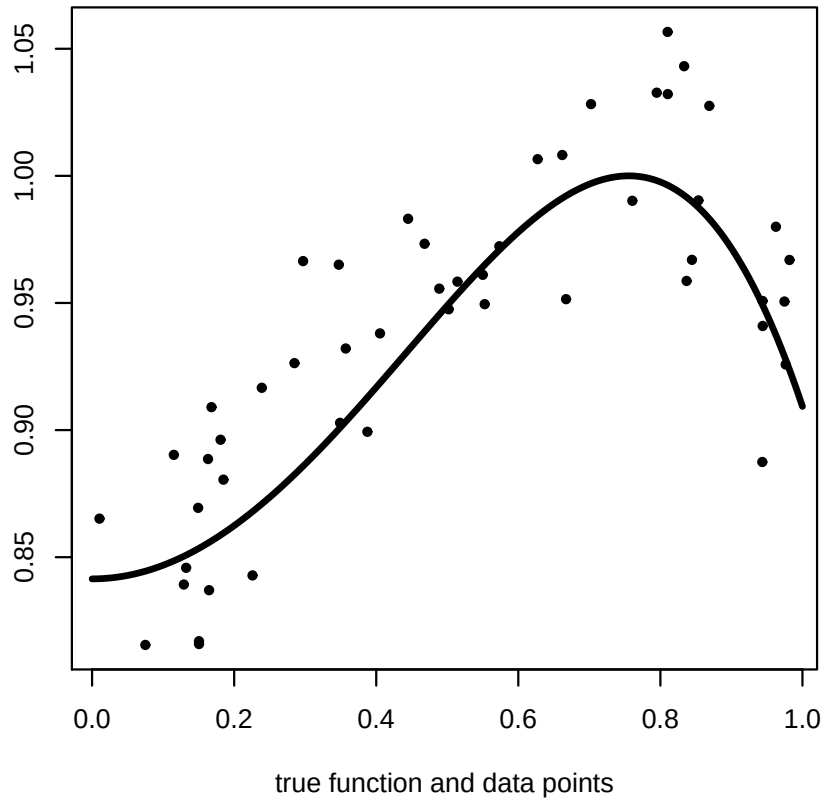
Averaging over complexity is the Bayesian approach — use as complex a model as might be needed, but don't choose a *single* set of parameter values. Instead, average the predictions found using *all* the parameter values that fit the data reasonably well, and which are plausible for the problem.

Example Using a Synthetic Data Set

Here are 50 points generated with x uniform from $(0, 1)$ and y set by the formula:

$$y \;=\; \sin(1 + x^2) \;+\; \text{noise}$$

where the noise has $N(0, 0.03^2)$ distribution.

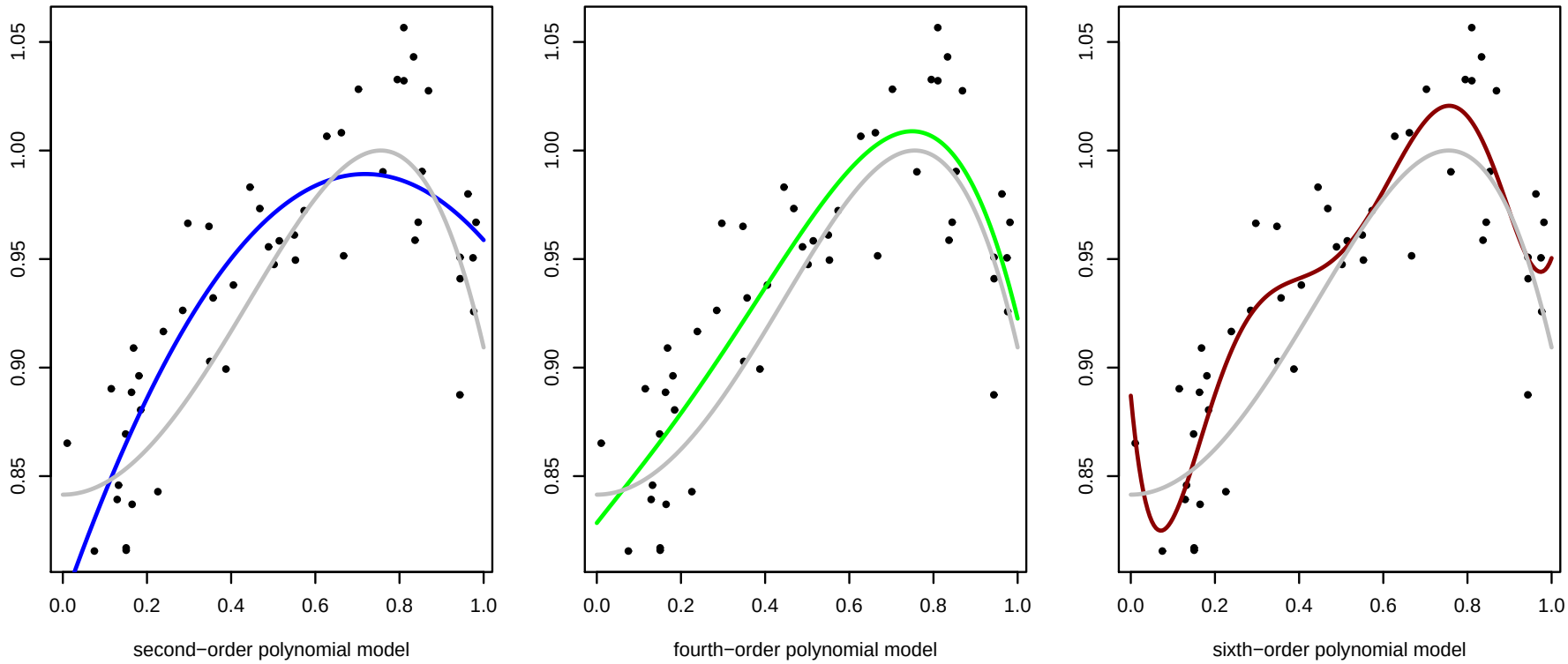


The noise-free function, $\sin(1 + x^2)$, is shown by the line.

Results of Fitting Polynomial Models of Various Orders

Here are the least-squares fits of polynomial models for y having the form (for $p = 2, 4, 6$) of

$$y \;=\; \beta_0 \;+\; \beta_1\, x \;+\; \cdots \;+\; \beta_p\, x^p \;+\; \text{noise}$$



The gray line is the true noise-free function. We see that $p = 2$ is too simple, but $p = 6$ is too complex — if we choose values for β_i that best fit the 50 data points.

Do We Really Need to Limit Complexity?

If we make predictions using the "best fitting" parameters of a model, we have to limit the number of parameters to avoid overfitting.

For this example, with this amount of data, the model with $p = 4$ was about right. We might be able to choose a good value for p using the method of "cross validation", which looks for the value that does best at predicting one part of the data from the rest of the data.

But we know that $\sin(1 + x^2)$ is not a polynomial function — it has an infinite series representation with terms of arbitrarily high order.

How can it be good to use a model that we know is false?

The Bayesian answer: It's not good. We should abandon the idea of using the "best" parameters and instead average over all plausible values for the parameters. Then we can use a model (perhaps a very complex one) that is as close to being correct as we can manage.

Reducing Dimensionality for the Digit Data

Consider the handwritten digit recognition problem previously mentioned. For this data, there are 7291 training cases, each with the true class (0–9) and 256 inputs (pixels). Can we replace these with many fewer inputs, without great loss of information?

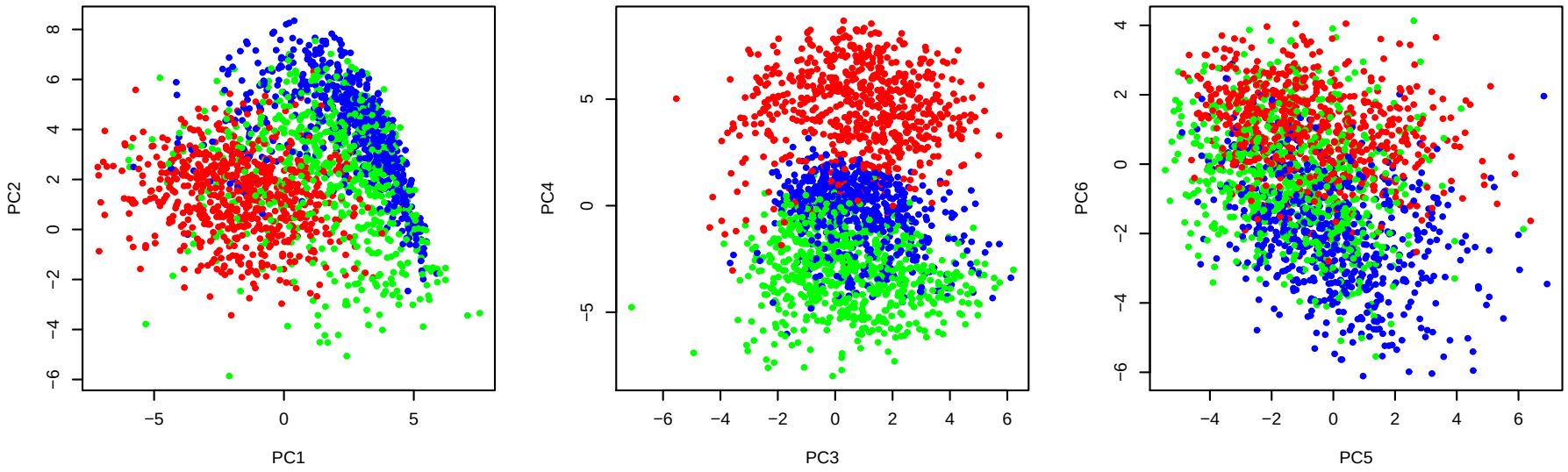
One simple way is by *Principal Components Analysis (PCA)*. We imagine the 7291 training cases as points in the 256 dimensional input space. We then find the direction of *highest variance* for these points, the direction of *second-highest variance* that is orthogonal to the first, etc.

We stop before sometime before we find 256 directions — say, after only 20 directions. We the replace each training case by the projections of the inputs on these 20 directions.

In general, this might discard useful information — perhaps the identity of the digit is actually determined by the direction of *least* variance. But often it keeps most of the information we need, with many fewer variables.

Principal Components for the Zip-Code Digits

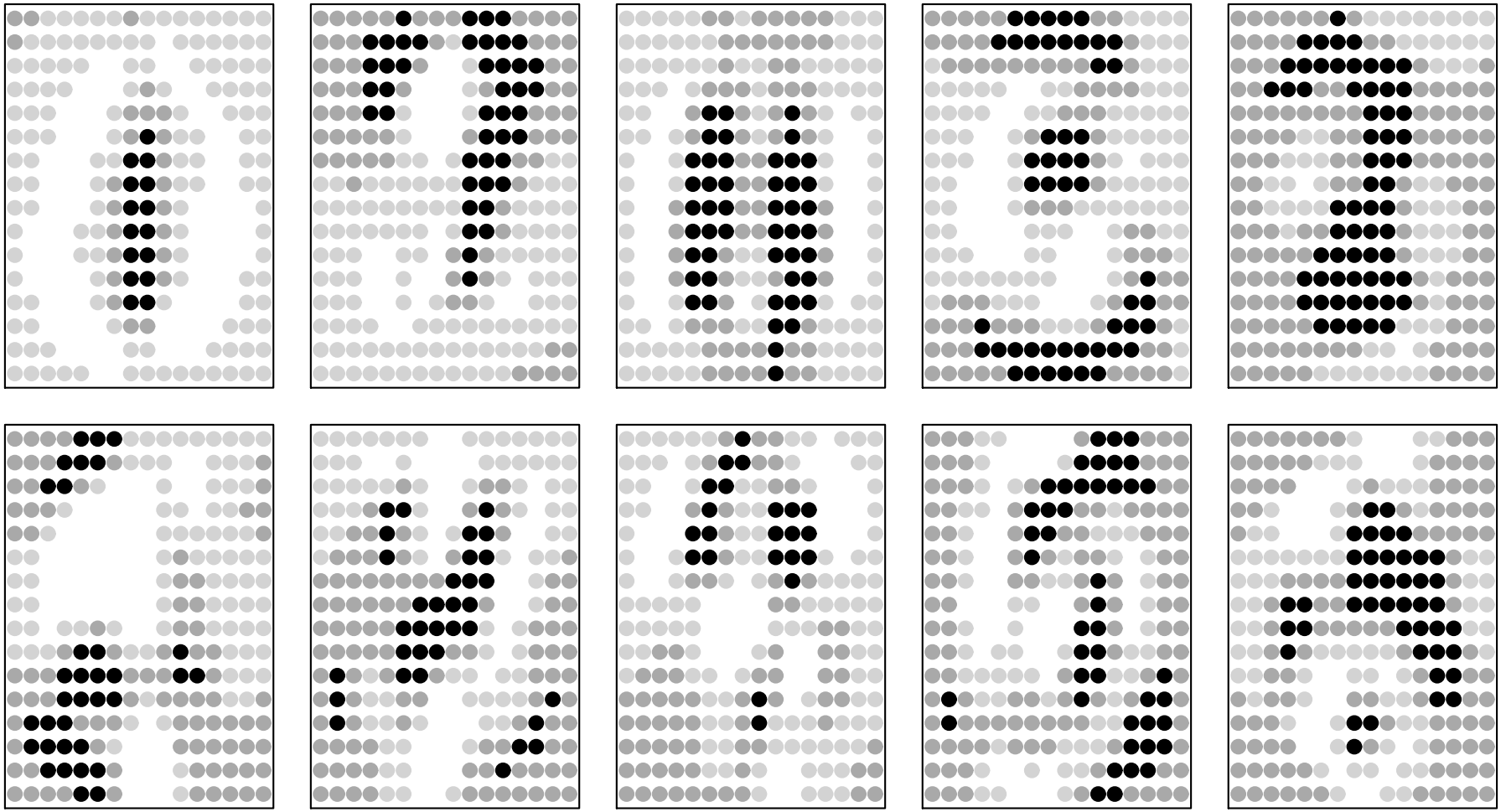
Here are plots of 1st versus 2nd, 3rd versus 4th, and 5th versus 6th principal components for training cases of digits "3" (red), "4" (green), and "9" (blue):



Clearly, these reduced variables contain a lot of information about the identity of the digit — probably much more than we'd get from any six of the original inputs.

Pictures of What the Principal Components Mean

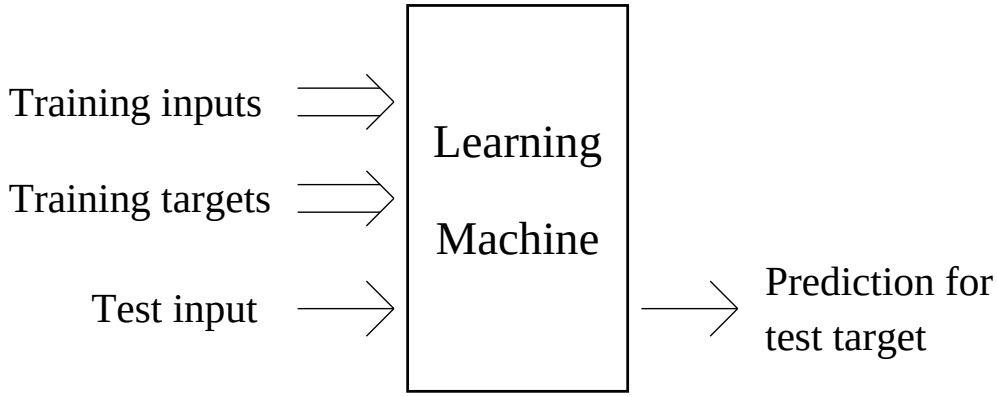
Directions of principal components in input space are specified by 256-dimensional unit vectors. We can visualize them as 16×16 "images". Here are the first ten:



Introduction to Supervised Learning

A "Supervised Learning Machine"

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

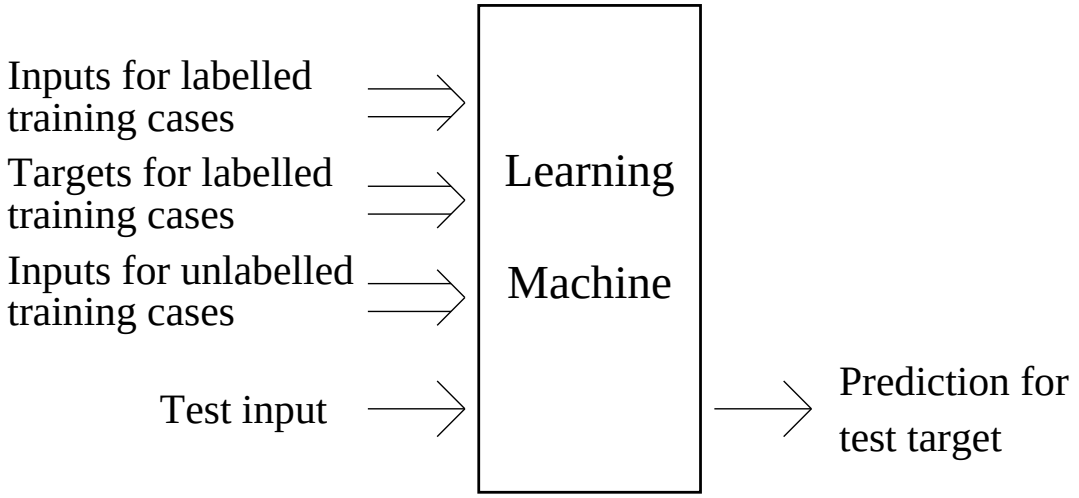


Any sort of statistical procedure for this problem *can* be viewed in this "mechanical" way, but is this a useful view? It does at least help clarify the problem...

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"

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 call the variable we want to predict the *target* or *response* variable, and denote it by y.

In a *classification* problem, y will take values from some finite set of class labels — binary classification, with $y = 0$ or $y = 1$, is one common type of problem. In a *regression* problem, y will be real-valued. (There are other possibilities, such as y being a count, with no upper bound.)

To help us predict y, we have measurements of p variables collected in a vector x, called *inputs*, *predictors*, or *covariates*.

We have a *training set* of n cases in which we know the values of both y and x, with these being denoted by y_i and x_i for the i'th training case. The value of input j in training case i is denoted by x_{ij}. The j'th input in a generic case is denoted by x_j. (Unfortunately, the meaning of x_5 is not clear with this notation.)

The above notation is more-or-less standard in statistics, but notation in the machine learning literature varies widely.

Predictions for Supervised Learning

In some *test case*, where we know the inputs, x, we would like to predict the value of the response y.

Ideally, we would produce a probability distribution, $P(y \mid x)$, 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 $\widehat{y}$.

For a real-valued response, we might set $\widehat{y}$ 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 $\widehat{y}$ 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 $\widehat{y}$ 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 $P(y \mid x)$ 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:

$$\widehat{y}(x) \quad = \quad \frac{1}{K} \sum_{i \in N_K(x)} y_i$$

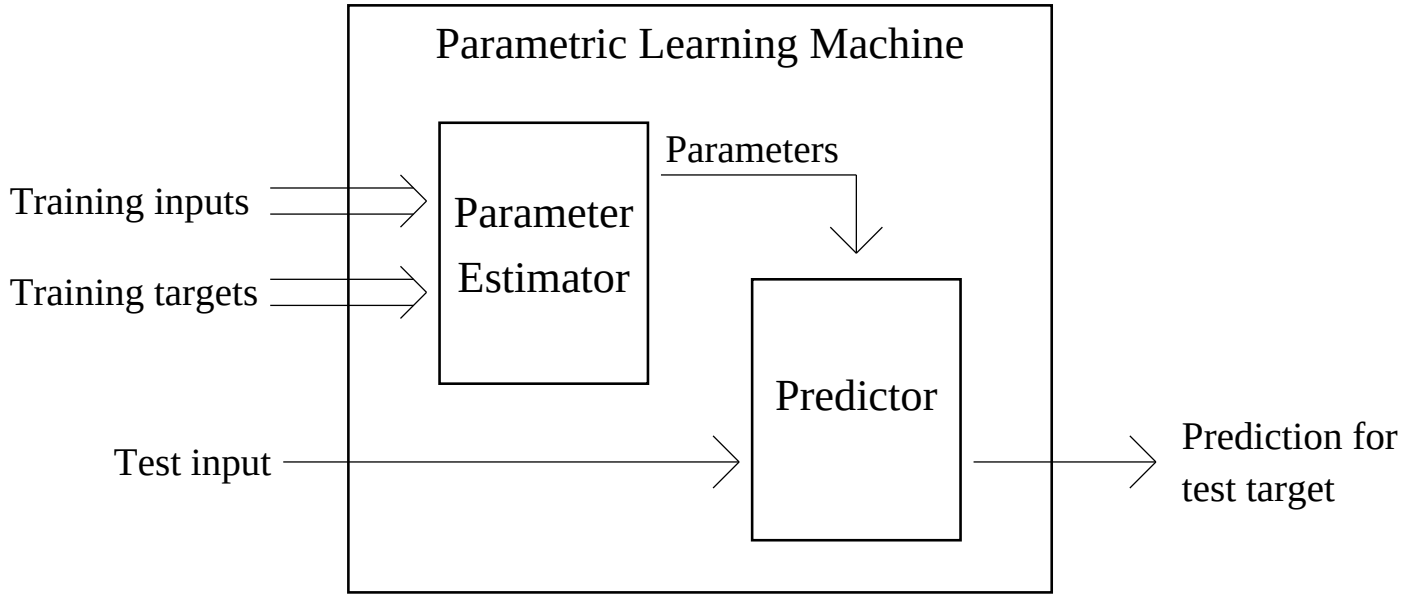
where $N_K(x)$ is the set of K training cases whose inputs are closest to the test inputs, x. (We'll ignore the possibility of ties.)

Big question: How should we choose K?

If K is too small, we may "overfit", 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 we make predictions for many test cases — we can estimate the parameters just once, then use them many times.

A hybrid 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 learning methods is *linear regression*.

The predictor for this method takes parameter estimates $\widehat{\beta}_0, \widehat{\beta}_1, \ldots, \widehat{\beta}_p$, found using the training cases, and produces a prediction for a test case with inputs x by the formula

$$\widehat{y} \;=\; \widehat{\beta}_0 \;+\; \sum_{j=1}^{p} \widehat{\beta}_j \, x_j$$

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.

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

$$\sum_{i=1}^{n} \left(y_i - \left(\beta_0 \;+\; \sum_{j=1}^{p} \beta_j \, x_{ij} \right) \right)^2$$

As is well-known, the $\widehat{\beta}$ that minimizes this can be found using matrix operations.

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 $\widehat{\beta}_0, \widehat{\beta}_1, \ldots, \widehat{\beta}_p$ 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 (if K is small), but consequently has a high potential for *overfitting*.

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

Supervised Learning Topics

In this course, we'll look at various supervised learning methods:

- Linear regression and logistic classification models, including where rather than the original inputs, we use "features" that are obtained by applying "basis functions" to the inputs.
- Gaussian process methods for regression and classification.
- Support Vector Machines.
- Neural networks.

We'll also see some ways of controlling the complexity of these methods to avoid overfitting, particularly

- Cross-validation.
- Regularization.
- Bayesian learning.