Introduction to Data Mining

David Banks

Institute of Statistics & Decision Sciences
Duke University

Abstract

This shortcourse begins with an introductory overview of data mining: its scope, classical approaches, and the heuristics that guided the initial development of theory and methods. Then the course moves towards the treatment of more modern issues such as boosting, overcompleteness, and large-p-small-n problems. This leads to a survey of currently popular techniques, including random forests, support vector machines, and wavelets. The main focus is upon regression inference, as this is a paradigm that informs all data mining applications, but we also discuss clustering, classification, and multidimensional scaling. The only prerequisites for the course are a basic knowledge of applied multivariate inference and a general level of statistical knowledge comparable to a strong master's degree. Any math will focus upon conveying general insight rather than specific details.

0. Table of Contents

1. Background and Overview

Data mining tries to find hidden structure in large, high-dimensional datasets.

Interesting structure can arise in regression analysis, discriminant analysis, cluster analysis, or more exotic situations, such as multidimensional scaling.

Classic applications include:

- Regression models for climate change, wine price, cost of software development.
- Classification models for fraudulent credit card transactions and good credit risk.
- Cluster analyses for market segmentation and microarray data.
- Multidimensional scaling analyses for document retrieval systems.

Data mining grew at the interface of statistics, computer science, and database management.

- Statistical work began in the 1980s, with the invention of CART, and expanded through increased research on visualization, nonparametric regression, and data quality.
- Computer scientists coined the term data mining, pioneered neural nets, pursued aggressive analysis of high-profile problems, and developed a body of ad hoc techniques.
- Database scientists developed SQL, relational databases, and other key tools.

Data mining has become an important research area, and was the topic of a year-long program at the Statistical and Applied Mathematical Sciences Institute, 2003-2004. That program was the impetus to the development of this shortcourse.

1.1 Example: Nonparametric Regression

Nonparametric regression is a natural way to introduce the main ideas in data mining. The core model is

$$y = f(\boldsymbol{x}) + \epsilon$$

where $\boldsymbol{x} \in \mathbb{R}^p$ and f is unknown. The emphasis is upon estimating the function f.

In real problems one observes $\{y_i, \boldsymbol{x}_i\}$, $i = 1, \ldots, n$. We assume that:

- the $\boldsymbol{x}_i$ are measured without error
- the ϵ_i are i.i.d. with mean zero
- the variance of the ϵ_i values is an unknown constant σ.

These are the minor assumptions, and can be weakened in customary ways.

The main assumption regards the class of functions to which f belongs. Common assumptions include:

- f is in a Sobolev space (essentially, these are functions with bounded derivatives)
- f has a bounded number of discontinuities.

For now, we require only that f be vaguely smooth.

The problem of estimating f becomes vastly harder as p, the dimension of $\boldsymbol{x}$, increases. This is called the Curse of Dimensionality (COD). The term was coined by Richard Bellman in the context of approximation theory (*Adaptive Control Processes*, 1961, Princeton University Press).

In order to minimize or evade the COD, data miners have invented many computer-intensive techniques. Some of these include: MARS, CART, Projection Pursuit Regression, Loess, random forests, support vector machines.

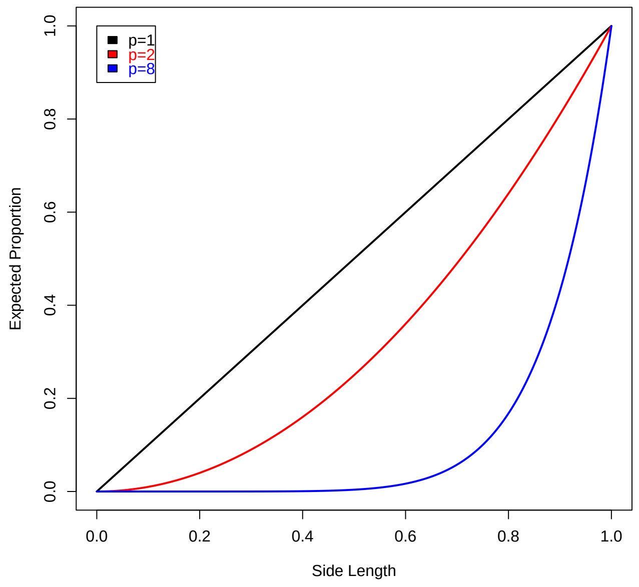
The COD applies to all multivariate analyses that choose not to impose strong modeling assumptions (e.g., that the relationship between $\boldsymbol{x}$ and $\mathbb{E}[Y]$ is linear, or that f belongs to a particular parametric family of curves).

Although we use regression as our example, the COD applies equally to classification, cluster analysis, and multidimensional scaling.

In terms of the sample size n and dimension p, the COD has three nearly equivalent descriptions:

- For fixed n, as p increases, the data become sparse.
- As p increases, the number of possible models explodes.
- For large p, most datasets are multicollinear (or concurve, which is a nonparametric generalization).

To explain the sparsity description of the COD, assume that n points are uniformly distributed in the unit cube in $\mathbb{R}^p$. What is the side-length ℓ of a subcube that is expected to contain a fraction d of the data? Ans: $\ell = \sqrt[p]{d}$



This means that for large p, the amount of local information that is available to fit bumps and wiggles in f is too small.

To explain the model explosion description of the COD, suppose we restrict attention to just linear models of degree 2 or fewer. For $p = 1$ these are:

$$\begin{array}{lll} \mathbb{E}[Y] = \beta_0 & \mathbb{E}[Y] = \beta_1 x_1 & \mathbb{E}[Y] = \beta_2 x_1^2 \\ \mathbb{E}[Y] = \beta_0 + \beta_1 x_1 & \mathbb{E}[Y] = \beta_0 + \beta_2 x_1^2 & \mathbb{E}[Y] = \beta_1 x_1 + \beta_2 x_1^2 \\ \mathbb{E}[Y] = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 & & \end{array}$$

For $p = 2$ this set is extended to include expressions with the terms $\alpha_1 x_2$, $\alpha_2 x_2^2$, and $\gamma_{12} x_1 x_2$.

For general p, combinatorics shows that the number of possible models is

$$2^{1+2p+\binom{p}{2}} - 1.$$

This increases superexponentially in p, and there is not enough sample to enable the data to discriminate among these models.

To explain the multicollinearity description of the COD, recall that multicollinearity occurs when the explanatory values concentrate on an affine subspace in $\mathbb{R}^p$. And multicollinearity implies that the predictive value of the fitted model breaks down quickly as one moves away from the subspace in which the data concentrate.

For large p, the number of possible subspaces is enormous $(2^p - 2)$, and so the probability that a sample of fixed size n concentrates on an affine shift of one of them, just by chance, is large.

Concurvity is the nonparametric analogue of multicollinearity, and it occurs when the data concentrate on some smooth manifold within $\mathbb{R}^p$. Since the number of smooth manifolds is larger than the number of affine shifts, the nonparametric version of the problem is worse.

Recently, several researchers have obtained results that purport to evade the COD.

- Barron (1994; *Machine Learning*, **14**, 115-133) shows that in a technical sense which we describe in 4.4, neural networks avoid the COD.
- Zhao and Atkeson (1991; *NIPS'91*, **4**, 936-943) show that in a sense similar to Barron's, Projection Pursuit Regression can evade the COD.
- Wozniakowski (1991; *Bulletin of the American Mathematical Society*, N.S., **24**, 184-194) uses a modification of Hammersley points (chosen to minimize the discrepancy from the uniform distribution in a Kolmogorov-Smirnov test) to dodge the COD in the context of multivariate integration.

None of these results has much practical significance to data miners. The results are *very* asymptotic, and rely upon some awkward fine print.

To counteract the COD, analysts appeal to the Principle of Parsimony. This holds that simple models have better predictive accuracy than complex models.

There are two kinds of wisdom in this rule:

- If the true model is complex, one cannot make accurate predictions in any case.
- If the true model is simple, then one can improve the fit by forcing the estimate to find a simple solution.

The principle is a guideline, not a rule. Fitting a foolish simple model is is wrong when the application is intrinsically complex. And if a simple model outperforms a complex model, perhaps one has chosen the wrong complex model.

1.2 Two Tools

In regression, classification, and (often) clustering, there are two key tasks:

- Assessment of model fit.
- Estimation of uncertainty.

The first problem is handled by some variant of cross-validation. The second problem is handled by the bootstrap.

Both cross-validation and bootstrapping are key methodologies in data mining, and we review them briefly.

1.2.1 Cross-Validation

To assess model fit in complex, computer-intensive situations, the ideal strategy is to hold out a random portion of the data, fit a model to the rest, then use the fitted model to predict the response values from the values of the explanatory variables in the hold-out sample.

This allows a straightforward estimate of predicted mean squared error (PMSE) for regression, or predictive classification error, or some similar fit criterion. But this wastes data.

Also, we usually need to compare fits among *many* models. If the same hold-out sample is re-used, then the comparisons are not independent and (worse) the model selection process will tend to choose a model the overfits the hold-out sample, causing spurious optimism.

Cross-validation is a procedure that balances the need to use data to select a model and the need to use data to assess prediction.

Specifically, v-fold cross-validation is as follows:

- randomly divide the sample into v portions;
- for $i = 1, \ldots, v$, hold out portion i and fit the model from the rest of the data;
- for $i = 1, \ldots, v$, use the fitted model to predict the hold-out sample;
- average the PMSE over the v different fits.

One repeats these steps (including the random division of the sample!) each time a new model is assessed.

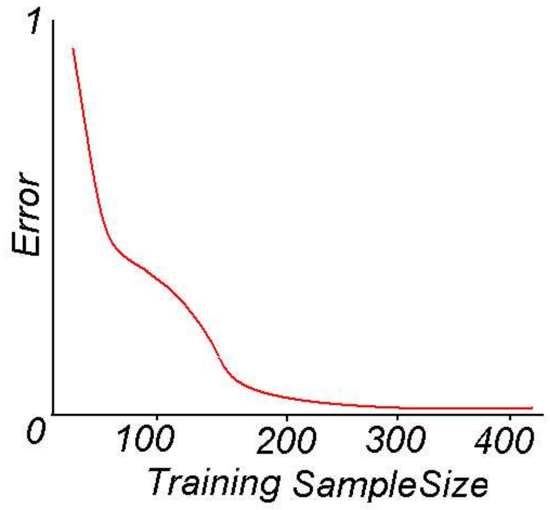
The choice of v requires judgment. If $v = n$, then cross-validation has low bias but possibly high variance, and computation is lengthy. If v is small, say 4, then bias can be large. A common choice is 10-fold cross-validation.

Intentional blank.

Intentional blank.

Leave-one-out cross-validation takes $v = n$, and calculates the predicted value for each observation using all of the other data. It is almost unbiased for the true predictive error. But the variance can be large, because the samples are so similar, and it is lengthy to calculate since it requires n runs.

It is tricky to know what value of v to use. If one wants to minimize mean squared error, then one must balance the variance in the estimate against the bias. One strategy is to plot the error as a function of the size of the training sample—when the curve levels off, there is no need to increase v.



Cross-validation is not perfect—some dependency remains, and the process can absorb a lot of computer time. Many of the data mining techniques use computational shortcuts to approximate cross-validation.

For example, in a regression model where the estimated values have linear dependence on the observed values, or $\hat{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{y}$, then often

$$n^{-1}\sum_{i=1}^{n}[y_i - \hat{f}^{-1}(\boldsymbol{x}_i)]^2 = n^{-1}\sum_{i=1}^{n}\left[\frac{y_i - \hat{f}(\boldsymbol{x}_i)}{1 - h_{ii}}\right]^2$$

where $\hat{f}^{-1}(\boldsymbol{x}_i)$ is the leave-one-out cross-validation estimate of f at $\boldsymbol{x}_i$. This reduces computational time by requiring only one calculation of $\hat{f}$.

To avoid calculating h_{ii}, Generalized Cross-Validation (GCV) estimates the total squared error as

$$n^{-1}\sum_{i=1}^{n}\left[\frac{y_i - \hat{f}(\boldsymbol{x}_i)}{1 - h_{ii}}\right]^2 = n^{-1}\sum_{i=1}^{n}\left[\frac{y_i - \hat{f}(\boldsymbol{x}_i)}{1 - \mathbf{tr}(\boldsymbol{H})/n}\right]^2.$$

Cross-validation fails when the explanatory values in the sample are not representative of future observations. One example is the presence of *twin data*.

Twin data often arise in drug discovery. Pharmaceutical companies keep libraries of the compounds they have studied, and use this to build data mining models that predict the chemical structure of biologically active molecules. But when the company finds a good molecule they promptly make a number of similar “twin” molecules (partly to optimize efficacy, partly to ensure an adequately broad patent).

If cross-validation is applied to this library, the hold-sample usually contains twins of molecules in the training sample. Thus the predictive accuracy of the fitted model will seem spuriously good.

1.2.2 The Bootstrap

The bootstrap is a popular tool for setting approximate confidence regions on estimated quantities when the underlying distribution is unknown. It relies upon samples drawn from the empirical cumulative distribution function (ecdf).

Let $X_1, \ldots, X_n$ be iid with cdf F. Then

$$\hat{F}_n(x) = \frac{1}{n} \sum_{i=1}^{n} I_{(-\infty, X_i]}(x)$$

is the ecdf. The ecdf is bounded between 0 and 1 with jumps of size n^{-1} at each observation.

The ecdf is a nonparametric estimator of the true cdf. It is the basis for Kolmogorov's goodness-of-fit test, and plays a key role in many aspects of statistical theory.

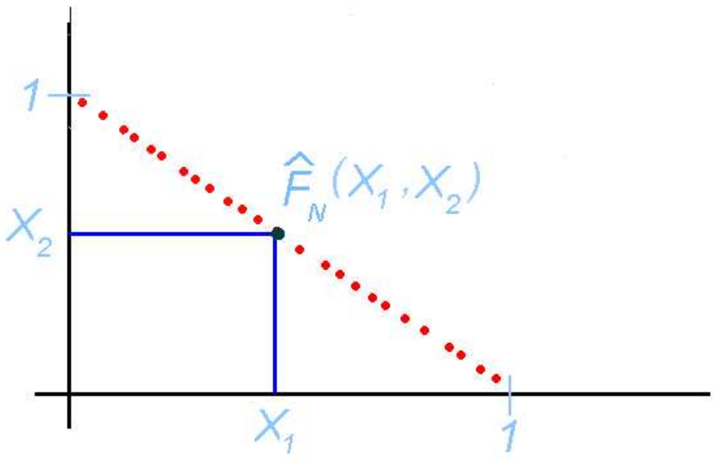
The Glivenko-Cantelli theorem implies

$$\mathbb{P}[\limsup_{x} |\hat{F}_n(x) - F(x)| < \epsilon] = 1 \textbf{ a.s.}$$

This fails in higher dimensions, but convergence in distribution holds, i.e., for each continuity point $\boldsymbol{x}$ in $\mathbb{R}^p$,

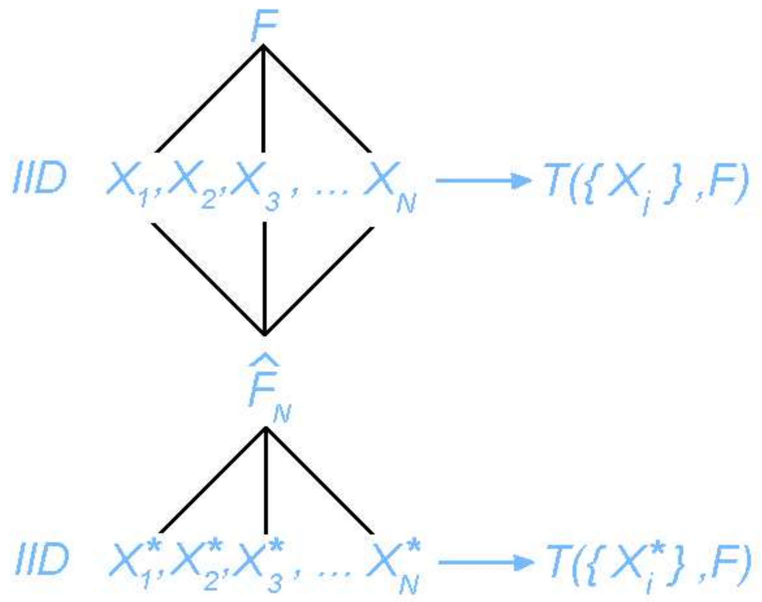
$$\lim_n \hat{F}_n(\boldsymbol{x}) = F(\boldsymbol{x}).$$

This is sufficient for bootstrap purposes.



Denote the estimate of a functional of F using the sample $\{\boldsymbol{X}_i\}$ by $T(\{\boldsymbol{X}_i\}, F)$. Here $T(\{\boldsymbol{X}_i\}, F)$ might be the population variance, or the ratio of the 7th moment to the 12th moment, or any other complex function for which one wants a confidence region.

To set a confidence region one needs to know how the sampling variation affects the estimator. The strategy behind the bootstrap reflects the reflexivity in its name. See Efron (1979; *Annals of Statistics*, **7**, 1-26)



Since $\hat{F}_n \to F$, the distribution of $T(\{X_i^*\}, \hat{F}_n)$ converges to the distribution of $T(\{X_i\}, F)$, the quantity of interest.

Suppose $X_1, \ldots, X_n$ are iid f and we want to find the distribution of

$$T(\{X_i\}, F) = \sqrt{n}\frac{\bar{X} - \mu}{s}$$

or, equivalently,

$$\mathbb{P}_F[\sqrt{n}\frac{\bar{X} - \mu}{s} \leq t] \quad \forall\, t \in \mathbb{R}.$$

The bootstrap approximation to this is

$$\mathbb{P}_{\hat{F}_n}[\sqrt{n}\frac{\bar{X}^* - \bar{X}}{s^*} \leq t] \quad \forall\, t \in \mathbb{R}$$

where $\bar{X}^*$ is the average of a random sample from $\hat{F}_n$ and s^* is its standard deviation. This can be numerically evaluated by repeated resamplings from $\hat{F}_n$.

Before Efron invented the bootstrap, statisticians used the Central Limit Theorem (CLT) to approximate the distribution of $T(\{X_i\}, F)$ by a standard normal distribution. So how good is the bootstrap? Is it better than the CLT?

To answer this, use an Edgeworth expansion argument as in Hall (1992; *The Bootstrap and Edgeworth Expansion*, Springer). Under reasonable technical conditions,

$$\mathbb{P}_F[\sqrt{n}\frac{\bar{X}-\mu}{s} \leq t] = \Phi(t) + n^{-1/2}p_1(t)\phi(t) + \ldots + n^{-j/2}p_j(t)\phi(t) + o(n^{-j/2})$$

where $\Phi(t)$ is the cdf of the standard normal, $\phi(t)$ is its density function, and the $p_j(t)$ functions are polynomials related to the Hermite polynomials and involve the jth and lower moments of F.

The "oh" notation $o(h(n))$ means that the error gets small faster than $h(n)$; i.e.,

$$\lim_{n\to\infty} \mathbf{error}/h(n) = 0.$$

If this happens in probability, we denote it by $o_p(h(n))$.

Recall that a pivot is a function of the data (and usually the unknown parameters) whose distribution does not depend upon the unknown parameters. For example,

$$T(\{X_i\}, F) = \sqrt{n}\frac{\bar{X} - \mu}{s}$$

is a pivot in the class of normal distributions, since it since this has the student's-t distribution.

And in the class of distributions for which the first two moments are finite, $T(\{X_i\}, F)$ is an asymptotic pivot, since its asymptotic distribution is the standard normal.

For functionals that are asymptotic pivots with standard normal distribution, the Edgeworth expansion implies

$$\begin{aligned} G(y) &= \mathbb{P}[T(\{X_i\}, F) \leq y] \\ &= \Phi(y) + n^{-1/2} p_1(y)\phi(y) + \mathcal{O}(n^{-1}) \end{aligned}$$

where $\mathcal{O}(n^{-1})$ means that the ratio of the absolute error to n^{-1} is bounded for all $n > M$.

The bootstrap estimate for $G(y)$ turns out to be

$$\begin{aligned} G^*(y) &= \mathbb{P}[T(\{X_i^*\}, \hat{F}_n) \leq y \,|\, \{X_i\}] \\ &= \Phi(y) + n^{-1/2}\hat{p}(y)\phi(y) + \mathcal{O}_p(n^{-1}) \end{aligned}$$

where

$$T(\{X_i^*\}, \hat{F}_n) = \sqrt{n}\frac{\bar{X}^* - \bar{X}}{s^*}$$

as before, and $\hat{p}(y)$ is obtained from $p(y)$ by replacing the jth and lower moments of F by the corresponding moments of the ecdf.

The $\mathcal{O}_p(n^{-1})$ is a random variable that means the error term is $\mathcal{O}(n^{-1})$ in probability, or

$$\lim_{\lambda\to\infty} \limsup_{n\to\infty} \mathbb{P}[\frac{\textbf{error}}{n^{-1}} > \lambda] = 0.$$

One can show (in a course in asymptotics) that $\hat{p}(y) - p(y) = \mathcal{O}_p(n^{-1/2})$.

Thus

$$\begin{aligned} G^*(y) - G(y) &= n^{-1/2}\phi(y)[\hat{p}(y) - p(y)] + \mathcal{O}_p(n^{-1}) \\ &= \mathcal{O}_p(n^{-1}) \end{aligned}$$

since the first term of the sum is also $\mathcal{O}_p(n^{-1})$ and big-oh errors add.

So using a bootstrap approximation to an asymptotic pivot statistics incurs an error of order n^{-1}.

In contrast, recall that

$$\begin{aligned} G(y) - \Phi(y) &= n^{-1/2}p(y)\phi(y) + \mathcal{O}(n^{-1}) \\ &= \mathcal{O}(n^{-1/2}). \end{aligned}$$

So the CLT has error of order $n^{-1/2}$, and thus is asymptotically worse than the bootstrap.

Suppose we had bootstrapped a function that was not a pivot. For example, the percentile bootstrap (cf. Efron, 1982; *The Jackknife, the Bootstrap, and Other Resampling Plans*, SIAM, Philadelphia) would use the distribution of

$$U^* = \sqrt{n}(\bar{X}^* - \bar{X})$$

as a proxy when making uncertainty statements about $U = \bar{X} - \mu$.

In this case,

$$\begin{aligned} H(y) &= \mathbb{P}[U \leq y] \\ &= \mathbb{P}[\frac{1}{s}U \leq \frac{1}{s}y] \\ &= \mathbb{P}[T \leq y/s] \\ &= \Phi(y/s) + n^{-1/2}p(y/s) + \mathcal{O}(n^{-1}), \end{aligned}$$

which uses the Edgeworth expansion again.

Similarly,

$$\begin{aligned} H^*(y) &= \mathbb{P}[U^* \leq y \,|\, \{X_I\}] \\ &= \Phi(y/s^*) + n^{-1/2}\hat{p}(y/s^*)\phi(y/s^*) + \mathcal{O}(n^{-1}). \end{aligned}$$

From asymptotics, it can be shown that:

$$\begin{aligned} p(y/s) - \hat{p}(y/s^*) &= \mathcal{O}_p(n^{-1/2}) \\ s - s^* &= \mathcal{O}_p(n^{-1/2}). \end{aligned}$$

Thus

$$H(y) - H^*(y) = \Phi(y/s) - \Phi(y/s^*) + n^{-1/2}[p(y/s)\phi(y/s) - \hat{p}(y/s^*)\phi(y/s^*)] + \mathcal{O}_p(n^{-1}).$$

The second term has order $\mathcal{O}_p(n^{-1})$ but the first has order $\mathcal{O}_p(n^{-1/2})$.

So when the statistic is not an asymptotic pivot, the bootstrap and the CLT have the same asymptotics. It pays to bootstrap a studentized pivot.

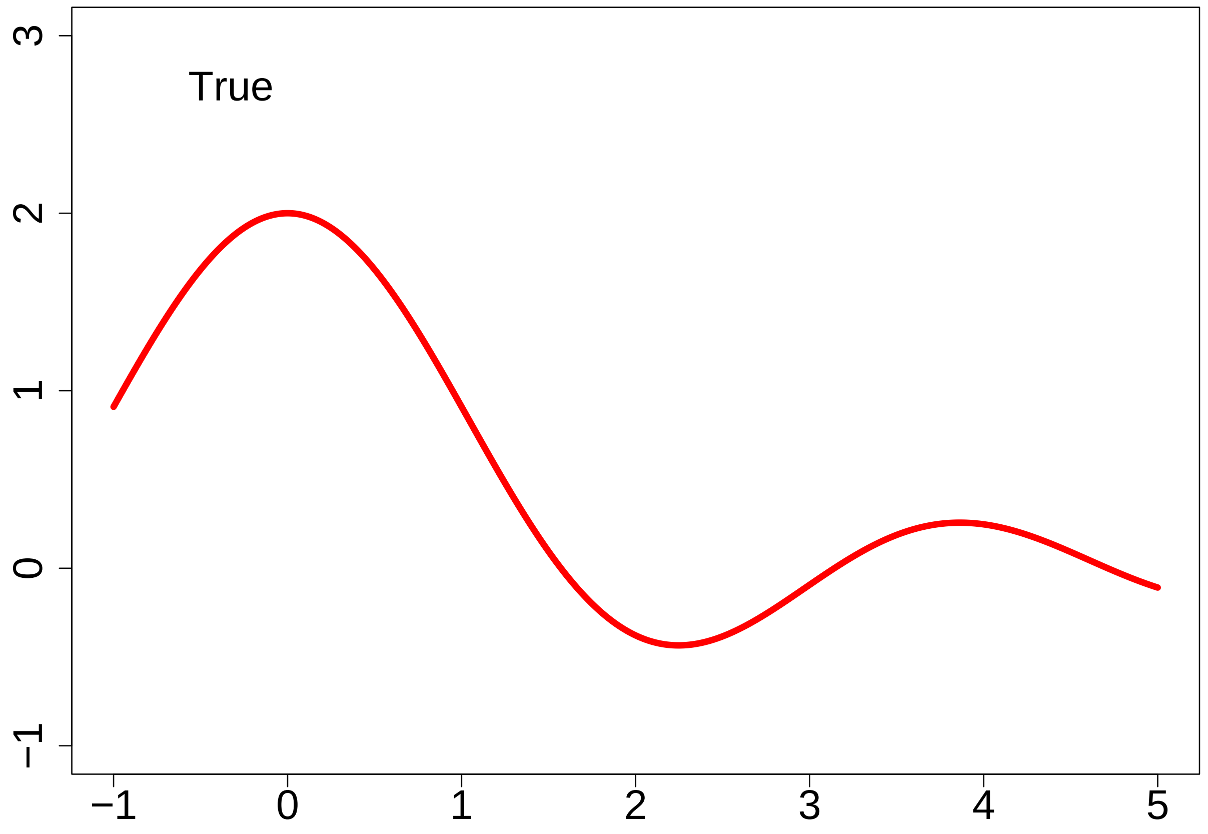
2. Smoothing

A smoothing algorithm is a summary of trend in Y as a function of explanatory variables $X_1, \ldots, X_p$. The smoother takes data and returns a function, called a smooth.

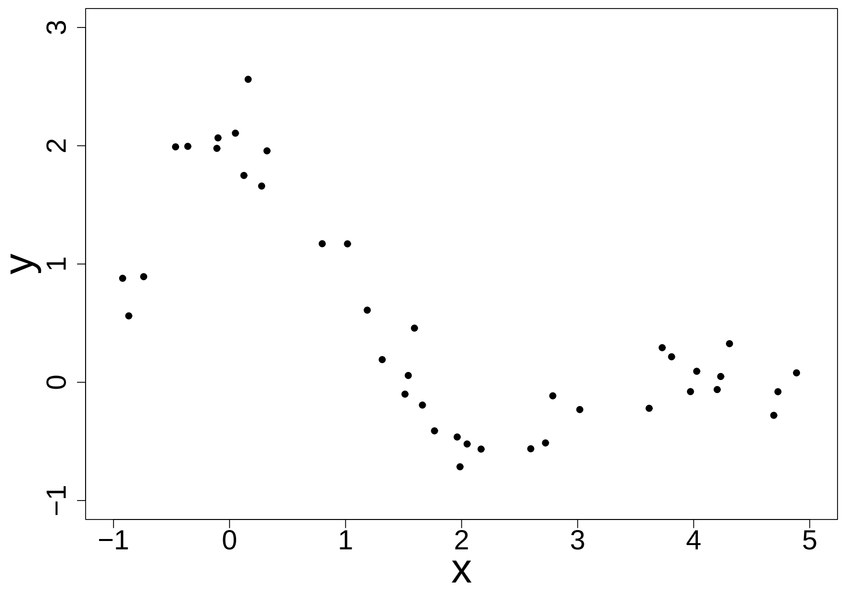
We focus on scatterplot smooths, for which $p = 1$. These usually generalize to $p = 2$ and $p = 3$, but the COD quickly renders them impractical. Even so, they can be used as building blocks for more sophisticated smoothing algorithms that break down less badly with large p, and we discuss several standard smoothers it illustrate the issues and tradeoffs.

Essentially, a smooth just finds an estimate of f in the nonparametric regression function $Y = f(x) + \epsilon$.

As a running example for the next several sections, assume we have data generated from the following function by adding $N(0..25)$ noise.

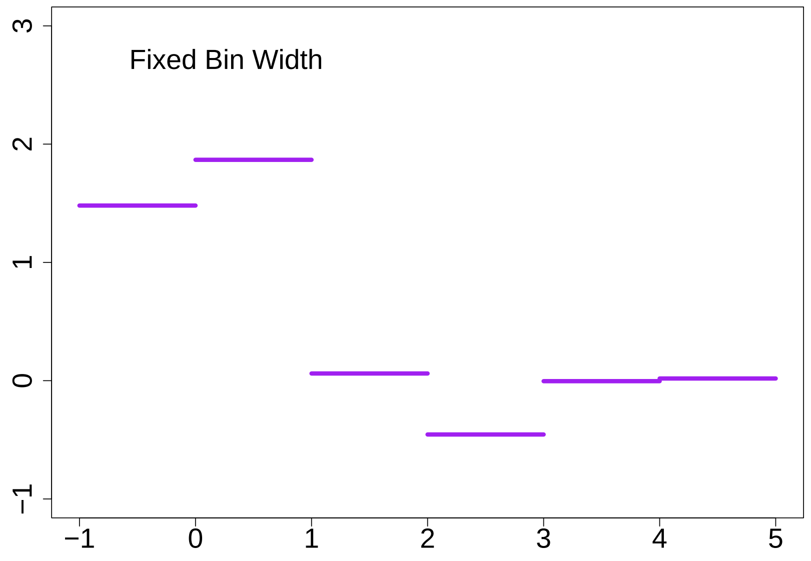


The x values were chosen to be spaced out at the left and right sides of the domain, and the raw data are shown below.



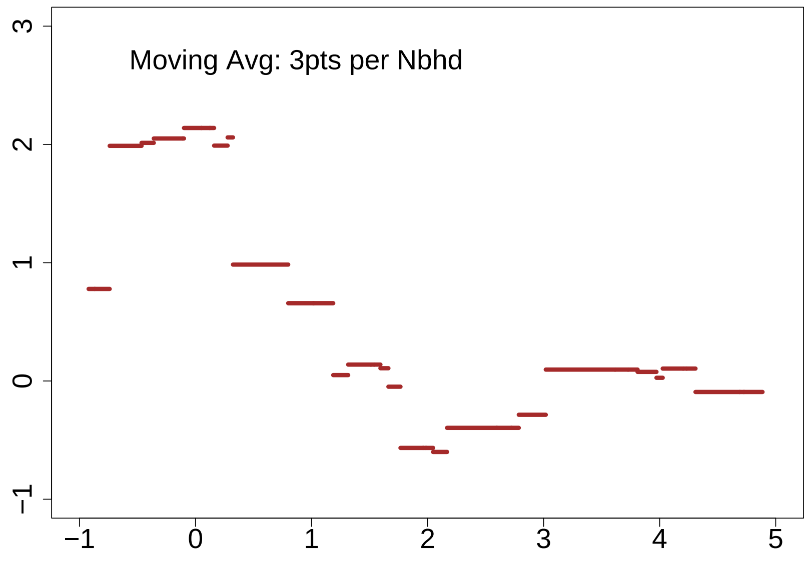
2.1 Bin Smoothing

Here one partitions $\mathbb{R}^p$ into disjoint bins; e.g., for $p = 1$ take $\{[i, i+1), i \in \mathcal{Z}\}$. Within each bin average the Y values to obtain a smooth that is a step function.



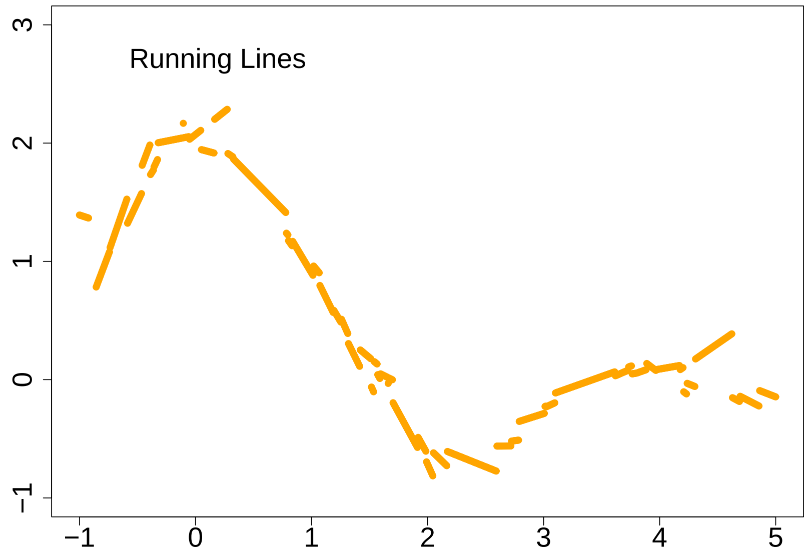
2.2 Moving Averages

Moving averages use variable bins containing a fixed number of observations, rather than fixed-width bins with a variable number of observations. They tend to wiggle near the center of the data, but flatten out near the boundary of the data.



2.3 Running Line

This improves on the moving average by fitting a line rather than an average to the data within a variable-width bin. But it still tends to be rough.



2.4 Loess

Loess was developed by Cleveland (1979; *Journal of the American Statistical Association*, **84**, 829-836).

Loess extends the running line smooth by using weighted linear regression inside the variable-width bins. Loess is more computationally intensive, but is often satisfactorily smooth and flexible.

LOESS fits the model

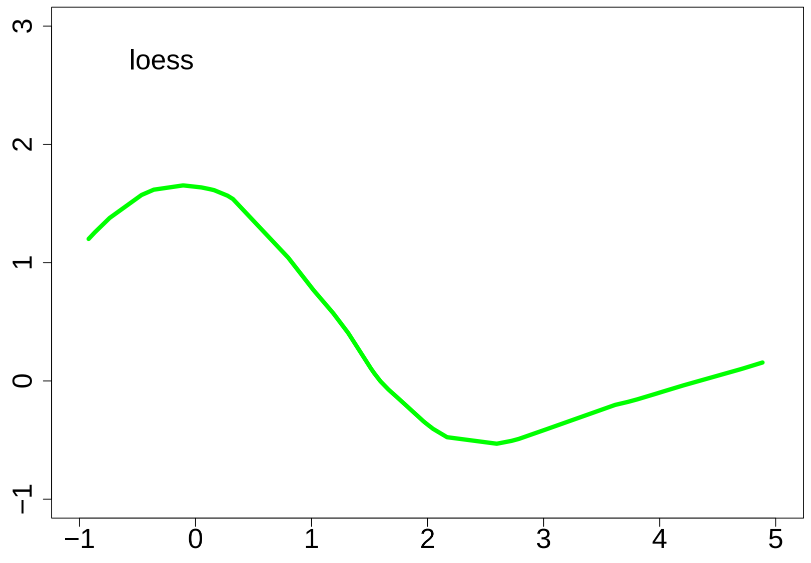
$$\mathbb{E}[Y] = \boldsymbol{\theta}(\boldsymbol{x})'\boldsymbol{x}$$

where

$$\hat{\boldsymbol{\theta}}(\boldsymbol{x}) = \textbf{argmin}_{\boldsymbol{\theta}\in\mathbb{R}^p} \sum_{i=1}^{n} w_i(\boldsymbol{x})(Y_i - \boldsymbol{\theta}'\boldsymbol{X}_i)^2$$

and w_i is a weight function that governs the influence of the ith datum according to the direction and distance of $\boldsymbol{X}_i$ from $\boldsymbol{x}$.

LOESS is a consistent estimator, but may be inefficient at finding relatively simple structures in the data. Although not originally intended for high-dimensional regression, LOESS is often used.



2.5 Kernel Smoothers

These are much like moving averages except that the average is weighted and the bin-width is fixed. Kernel smoothers work well and are mathematically tractable. The weights in the average depend upon the kernel $K(\boldsymbol{x})$. A kernel is usually symmetric, continuous, nonnegative, and integrates to 1 (e.g., a Gaussian kernel).

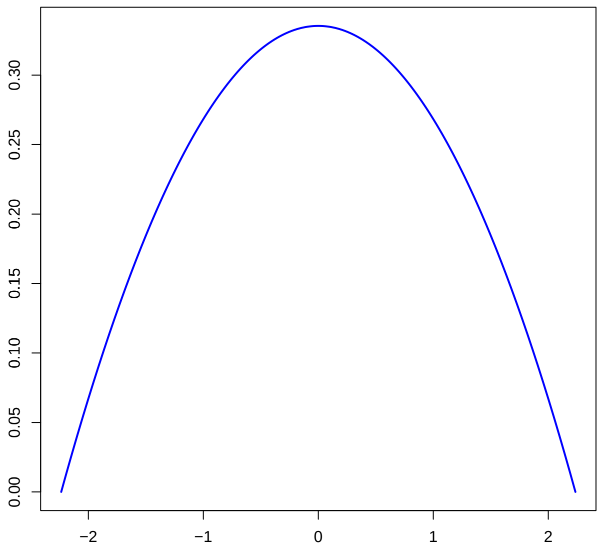
The bin-width is set by h, also called the bandwidth. This can vary, adapting to information in the data on the roughness of the function.

Let $\{(y_i, \boldsymbol{x}_i)\}$ denote the sample. Set $K_h(\boldsymbol{x}) = h^{-1}K(h^{-1}\boldsymbol{x})$. Then the Nadaraya-Watson estimate of f at $\boldsymbol{x}$ is

$$\hat{f}(\boldsymbol{x}) = \frac{\sum_{i=1}^n K_h(\boldsymbol{x} - \boldsymbol{x}_i)y_i}{\sum_{i=1}^n K_h(\boldsymbol{x} - \boldsymbol{x}_i)}.$$

See Watson (1964; *Theory and Probability Applications*, **10**, 186-190) and Nadaraya (1964; *Sankha A*, **26**, 359-372).

For kernel estimation, the Epanechnikov function has good properties (see Silverman; 1986, *Density Estimation for Statistics and Data Analysis*, Chapman & Hall).



The function is

$$K(x) = \frac{3}{4}(1-x)^2 \quad \textbf{on} \;\; -1 \le x \le 1$$

and is zero elsewhere.

Intentional blank.

2.6 Splines

If one estimates f by minimizing the equation that balances least squares fit with a roughness roughness penalty, e.g.,

$$\min_{f \in \mathcal{F}} \sum_{I=1}^{n} [y_i - f(\boldsymbol{x}_i)]^2 + \lambda \int [f^{(k)}(\boldsymbol{x})]^2 \, d\boldsymbol{x} \tag{1}$$

over an appropriate set of functions (e.g., the usual Hilbert space of square-integrable functions), then the solution one obtains are smoothing splines.

Smoothing splines are piecewise polynomials, and the pieces are divided at the sample values $\boldsymbol{x}_i$.

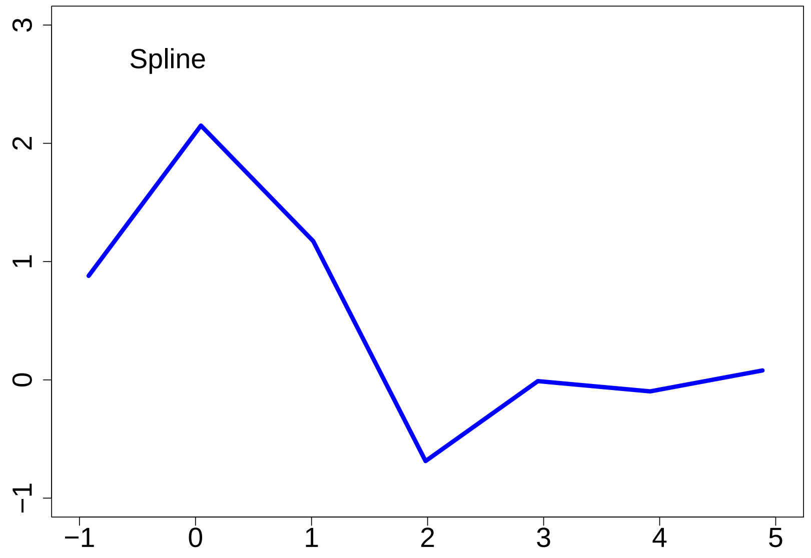
The $\boldsymbol{x}$ values that divide the fit into polynomial portions are called knots. Usually splines are constrained to be smooth across the knots.

intentional blank

intentional blank

Regression splines have fixed knots that need not depend upon the data. Also, knot selection techniques enable one to find good knots automatically.

Splines are computationally fast, enjoy strong theory, work well, and are widely used.



2.7 Comparing Smoothers

Most smoothing methods are approximately kernel smoothers, with parameters that correspond to the kernel $K(\boldsymbol{x})$ and the bandwidth h.

In practice, one can:

- fix h by judgment,
- find the optimal fixed h,
- fit h adaptively from the data,
- fit the kernel $K(\boldsymbol{x})$ adaptively from the data.

There is a point of diminishing returns, and this is usually hit when one fits the h adaptively.

Scott (1992; *Multivariate Density Estimation*, Wiley) provides a nice discussion of smoothing issues in the context of density estimation.

Breiman and Peters (1992; *International Statistics Review*, **60**, 271-290) give results on a simulation experiment to compare smoothers. Broadly, they found that:

- adaptive kernel smoothing is good but slow,
- smoothing splines are accurate but too rough in large samples,
- everything else is not really competitive in hard problems (i.e., those with large sample sizes and/or high dimensions).

Theoretical understanding of the properties of smoothing depends upon the eigenstructure of the smoothing matrix. Hastie, Tibshirani, and Freidman (2001; *The Elements of Statistical Learning*, Springer) provide an introduction and summary of this area in Chapter 5.

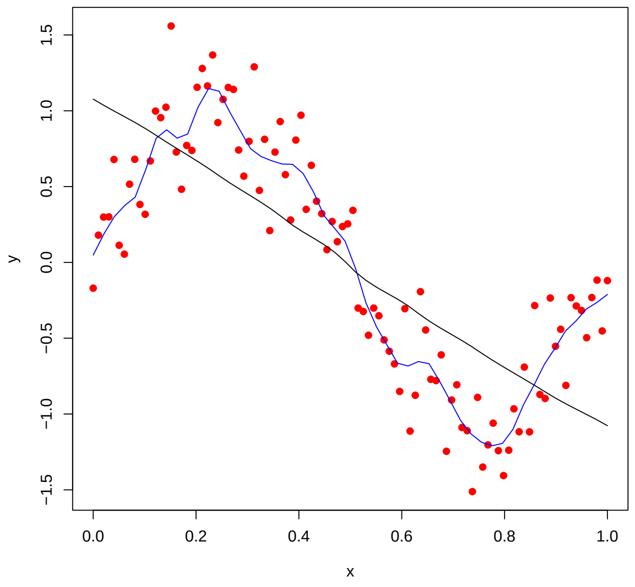
In linear regression one starts with a quantity of information equal to n degrees of freedom where n is the number of independent observations in the sample. Each estimated parameter reduces the information by 1 degree of freedom. This Occurs because each estimate corresponds to a linear constraint in $\mathbb{R}^n$, the space in which the n observations lie.

Smoothing is a nonlinear constraint and costs more information. But most smoothers can be expressed as a linear operator (matrix) $\boldsymbol{S}$ acting on the response vector $\boldsymbol{y} \in \mathbb{R}^n$. It turns out that the degrees of freedom lost to smoothing is then $\text{tr}(\boldsymbol{S})$.

Most smoothers are shrinkage estimators. Mathematically, they pull the weights on the coefficients in the basis expansion for $\boldsymbol{y}$ towards zero.

Shrinkage is why smoothing has an effective degrees of freedom between p (as in regression, which does not shrink) and n (which is what one would expect from a naive count of the number of parameters).

Smoothing entails a tradeoff between the bias and variance in $\hat{f}$. If one undersmooths, $\hat{f}$ is rough (high variance) but has low bias. If one smooths too much, $\hat{f}$ has small variance but high bias.



Mean squared error is a criterion that captures both aspects. At $\boldsymbol{x}$,

$$\mathbf{MSE}[\hat{f}] = \mathbb{E}[(\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x}))^2] = \mathrm{Var}\,[\hat{f}(\boldsymbol{x})] + \mathbf{bias}^2[\hat{f}(\boldsymbol{x})].$$

One wants a smooth that minimizes MSE$[\hat{f}(\boldsymbol{x})]$ over all $\boldsymbol{x}$.

intentional blank

intentional blank

3. Search and Variable Selection

Data mining entails many kinds of search. Some searches are easier than others.

A relatively easy kind of search is univariate. For example, one might want to find an appropriate bandwidth h for a smoother or the appropriate degree for fitting a polynomial regression or the number of terms to include in a model.

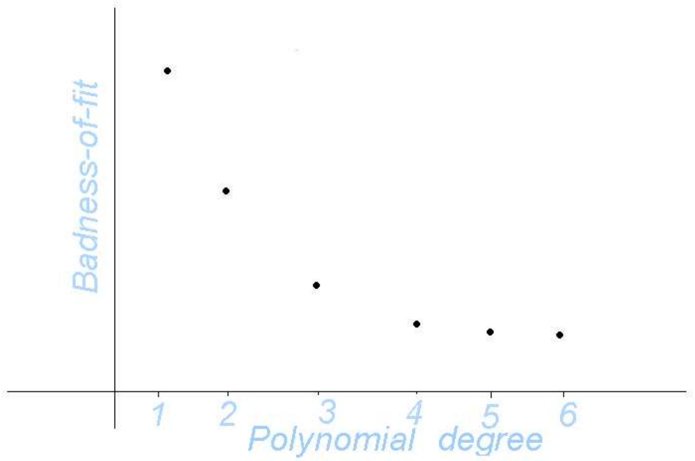
Search becomes harder in multivariate cases, such as finding an appropriate set of knots for spline smoothing or a set of weights in a neural network.

Even more difficult is combinatorial search, where there is no natural Euclidean space. This occurs in variable selection (a/k/a feature selection).

The hardest search is list search, where there is no structure in the problem at all. This occurs, for example, when there is a list of possible models, each of which is largely unrelated to the others, and one wants to find the best model for the data.

3.1 Univariate Search

In univariate search, one good strategy is to plot some criterion (e.g., goodness-of-fit, predictive mean squared error) against the index (e.g., degree in polynomial regression) and look for a knee in the curve.



If the criterion is monotonic in the index but graphical solution is difficult, try Fibonnacci search. If the criterion is not monotonic in the index, then other kinds of search (e.g., random restart hill-climbing, simulated annealing) should be considered.

intentional blank

3.2 Multivariate Search

For multivariate search, there are dozens of smart algorithms, and the best choice depends upon the nature of the response surface.

If the surface has a single bump, then steepest ascent works very well. If the surface has many bumps, then random restart combined with steepest ascent is a strategy that enables one to make probability statements about the chance that one has found a global maximum.

The Nelder-Mead algorithm is simple to program and very robust. It sets a simplex in the space $\mathbb{R}^k$ and passes the worst vertex through the center to the opposite side. (Nelder and Mead; 1965, *Computer Journal*, **7**, 308-313.)

Hybrid methods combine features from standard operations research algorithms and allow one to learn about the surface as one seeks the maximum.

3.3 Variable Selection

For combinatorial search, one wants to take advantage of whatever structure the problem enjoys.

For example, variable selection is a key problem in data mining. Often one has very large p and one wants (needs) to discard those that have no or little predictive value. If one looked at all possible subsets of variables, there are 2^p cases to consider. Something smarter is needed.

The 2^p possible models can be identified with the 2^p vertices of the unit hypercube in $\mathbb{R}^p$. The $(0, 0, \ldots, 0)$ vertex corresponds to the model with all variables excluded, whereas the $(1, 1, \ldots, 1)$ model is the regression on all variables. From this perspective, a clever search of the hypercube would be an attractive way to find a good regression model.

3.3.1 Gray Codes

A Hamiltonian circuit of the unit hypercube is a traversal that reaches each vertex exactly once. There are many possible Hamiltonian circuits—the exact number is not known.

From the standpoint of model search, one wants a Hamiltonian circuit that has desirable properties of symmetry, treating all vertices in the same way.

The Gray code is a procedure for listing the vertices of the hypercube in such a way that there is no repetition, each vertex is one edge from the previous vertex, and all vertices in a neighborhood are explored before moving on to a new neighborhood. Wilf (1989; *Combinatorial Algorithms: An Update*, SIAM) describes the mathematical theory and properties of the Gray code system.

To explain the Gray code algorithm, consider the case of four explanatory variables, or the unit hypercube in $\mathbb{R}^4$. The table shows the rank of the vertex in the Gray code traversal, the binary digit representation of the rank, and the bit string of the visited vertex on the hypercube.

Gray code vertex rank, binary rank, and vertex string.

0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

The Gray code has subtle balance. For example, it can be generated by reflection and recursion. Let L_p be the list of all possible binary bit strings of length p, arranged in Gray code order. Then generate the first half of L_{p+1} by writing a zero in front of each element in the list L_p. For the second half of L_{p+1}, write L_p in reverse order, and then prefix each element with a one. By concatenating the lists, one obtains L_{p+1}.

Suppose one prefixes each Gray code string with an infinite number of zeroes. This makes it possible to consider the numbers corresponding to the Gray code strings as an infinite series:

$$0, 1, 3, 2, 6, 7, 5, 4, 12, 13, 15, 14, 10, 11, 9, 8, \ldots$$

Note that each number in the sequence is relatively close to its neighbors. Yuen (1974; *IEEE Transactions on Information Theory*, **20**, 688) shows that two strings in the Gray code whose Hamming distance is at least d must have ranks that differ by at least $[2d/3]$ (here $[\cdot]$ is the nearest-integer function), and this provides the greatest possible separation. This means that the traversal explores models locally and exhaustively, rather than swooping back after a distant excursion.

From our perspective, the key point from Yuen's theorem is that it starts at an arbitrary model, then goes a large number of steps in the Gray code traversal, one ends up at a vertex corresponding to a model that is very different from the starting point. This property suggests that by taking every dth step, for d large, and then testing the corresponding model, one is performing a thorough search of the set of possible models.

To implement this search strategy one needs to be able to efficiently generate the Gray code vertex for step m. Wilf gives a theorem that does this.

Theorem: Let $m = \sum a_i 2^i$ be the representation of integer m in binary notation. Let $\ldots, b_3 b_2 b_1 b_0$ be the string for the vertex of rank m in the Gray code. Then

$$b_i = a_i + a_i + 1 \pmod{2}$$

for $i = 0, 1, 2, \ldots$.

To use this ranking theorem to efficiently explore a set of models, suppose one decides to examine only 100 models and then infer the final fit.

If there are p explanatory variables, one takes $d = [2^p/100]$, and then finds the Gray code vertex sequences of rank $d, 2d, \ldots, 100d$.

Each sequence defines a particular set of variables that may be included or excluded in the regression. In practice, one would examine the 100 model fitting results, probably in terms of the square root of the adjusted R^2, and then home in on the region of the cube that provides good explanation.

This enables one to quickly identify the vertex or bit string corresponding to a set of variables that provides good explanation. One might make additional Gray code searches in the region of the best results from the first search, and iterate to find the final model.

3.3.3 Experimental Design Selection

Another approach to variable selection uses ideas from experimental design. The method is due to Clyde (1999; *Bayesian Statistics 6*, 157-185, Oxford University Press).

To implement this approach, one views each explanatory variable as a factor in an experimental design. All factors have two levels, corresponding to whether or not the explanatory variable is included in the model. This enables one to perform a 2^{p-k} fractional factorial experiment in which one fits a multiple regression model to the included variables and records some measure of goodness-of-fit.

Obviously, one takes k to be sufficiently large that it is possible to perform the computations in a reasonable amount of time and also to limit the effect of multiple testing.

One uses analysis of variance to examine which factors and factor combinations have a significant influence on the observations. Significant main effects correspond to explanatory variables that contribute on their own. Significant interaction terms correspond to subsets of variables whose joint inclusion in the model provides explanation.

In multiple linear regression, these results are implicit in significance tests on the coefficients. But if one using one of the nonparametric regression techniques popular in data mining (e.g., MARS, PPR, neural nets), this helps find influential variables.

Based on the results of the designed experiment, one can ultimately find and fit the model that includes all variables corresponding to significant main effects or interactions. And the factorial design reduces the penalty one pays for multiple testing, as compared to exhaustive search or other less-efficient searches.

Possible measures of goodness-of-fit include:

- R^2, the proportion of variance in the observations that is explained by the model;
- Adjusted R^2, the proportion of variance in the observations that is explained by the model, but with an adjustment to account for the number of variables in the model;
- Mallows' C_p, a measure of predictive accuracy that takes account of the number of terms in the model.
- MISE, the mean integrated squared error of the fitted model over a given region (often the hyperrectangle defined by the minimum and maximum values taken by each explanatory variable used in the model.
- The square root of the adjusted R^2, since this transformation appears to stabilize the variance and thereby supports use of analysis of variance and response surface methodology in the model search.

Weisberg (1985, *Applied Linear Regression*, 185-190, Wiley) discusses the first three. Scott (1992, *Multivariate Density Estimation*, chapter 2.4, Wiley) discusses MISE.

3.4 List Search

With list search, there is no exploitable structure that links the elements of the list, and the list is usually so long that exhaustive search is infeasible.

There is not much that one can do. If one tests entries on the list at random, then one can try some of the following:

- Estimate the proportion of list entries that give results above some threshold.
- Use some modeling to estimate the maximum value on the list from a random sample of list entries.
- Estimate the probability that further search will discover a new maximum within a fixed amount of time.
- Use the solution to the Secretary Problem.

A strategy invented by computer scientists (Maron and Moore, 1997, *Artificial Intelligence Review*, **11** 193-225) is to **race** the testing.

One does pairwise comparisons of models. At first, one fits only a small random fraction of the data (say a random 1%) to each model on the list. Usually this is sufficient to discover which model is best and one discards the other.

If that small fraction does not distinguish the models, then one fits another small fraction. Only very rarely is it necessary to fit all or most of the data to select the better model.

Racing is an easy way to extend one's search capability by about 100-fold.

4. Nonparametric Regression

The multiple linear regression model is

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \epsilon$$

where $\mathbb{E}[\epsilon] = 0$, $\mathrm{Var}\,[\boldsymbol{\epsilon}] = \sigma^2$, and ϵ is independent of $x_1, \ldots, x_p$.

The model is useful because:

- it is interpretable—the effect of each explanatory variable is captured by a single coefficient
- theory supports inference and prediction is easy
- simple interactions and transformations are easy
- dummy variables allow use of categorical information
- computation is fast.

4.1 Additive Models

But additive linear fits are too flat. And the class of all possible smooths is too large—the COD makes it hard to smooth in high dimensions. The class of **additive models** is a useful compromise.

The additive model is

$$Y = \beta_0 + \sum_{k=1}^{p} f_k(x_k) + \epsilon$$

where the f_k are unknown smooth functions fit from the data.

The basic assumptions are as before, except we must add $\mathbb{E}[f_k(X_k)] = 0$ in order to prevent identifiability problems.

The parameters in the additive model are $\{f_k\}$, β_0, and σ^2. In the linear model, each parameter that is fit costs one degree of freedom, but fitting the functions costs more, depending upon what kind of univariate smoother is used.

Some notes:

- one can require that some of the f_k be linear or monotone;
- one can include some low-dimensional smooths, such as $f(X_1, X_2)$;
- one can include some kinds of interactions, such as $f(X_1 X_2)$;
- transformation of variables is done automatically;
- many regression diagnostics, such as Cook's distance, generalize to additive models;
- ideas from weighted regression generalize to handle heteroscedasticity;
- approximate deviance tests for comparing nested additive models are available;
- one can use the bootstrap to set pointwise confidence bands on the f_k (if these include the zero function, omit the term);

However, model selection, overfitting, and multicollinearity (concurvity) are serious problems. And the final fit may still be poor.

4.1.1 The Backfitting Algorithm

The backfitting algorithm is used to fit additive models. It allows one to use an arbitrary smoother (e.g., spline, Loess, kernel) to estimate the $\{f_k\}$

As motivation, suppose that the additive model is exactly correct. The for all $k = 1, \ldots, p$,

$$\mathbb{E}[Y - \beta_0 - \sum_{k \neq j} f_k(X_k) \mid x_j] = f_j(x_j).$$

The backfitting algorithm solves these p estimating equations iteratively. At each stage it replaces the conditional expectation of the partial residuals, i.e., $Y - \beta_0 - \sum_{k \neq j} \hat{f}_k(X_k)$ with a univariate smooth.

Notation: Let $\boldsymbol{y}$ be the vector of responses, let $\boldsymbol{X}$ be the $n \times p$ matrix of explanatory values with columns $\boldsymbol{x}_{\cdot k}$. Let $\boldsymbol{f}_k$ be the vector whose ith entry is $f_k(x_{ik})$ for $i = 1, \ldots, n$.

For $\boldsymbol{z} \in \mathbb{R}^n$, let $S(\boldsymbol{z} \,|\, \boldsymbol{x}_{\cdot k})$ be a smooth of the scatterplot of $\boldsymbol{z}$ against the values of the kth explanatory variable.

The backfitting algorithm works as follows:

1. Initialize. Set $\hat{\beta}_0 = \bar{Y}$ and set the f_k functions to be something reasonable (e.g., a linear regression). Set the $\boldsymbol{f}_k$ vectors to match.
2. Cycle. For $j = 1, \ldots, p$ set

$$f_k = S(\boldsymbol{Y} - \hat{\beta}_0 - \sum_{k \neq j} \boldsymbol{f}_k \,|\, \boldsymbol{x}_{\cdot k})$$

   and update the $\boldsymbol{f}_k$ to match.
3. Iterate. Repeat step (2) until the changes in the f_k between iterations is sufficiently small.

One may use different smoothers for different variables, or bivariate smoothers for predesignated pairs of explanatory variables.

The estimating equations that are the basis for the backfitting algorithm have the form:

$$\boldsymbol{P}\boldsymbol{f} = \boldsymbol{Q}\boldsymbol{Y}$$

for suitable matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$.

The iterative solution for this has the structure of a Gauss-Seidel algorithm for linear systems (cf. Hastie and Tibshirani; 1990, *Generalized Additive Models*, chap. 5.2).

This structure ensures that the backfitting algorithm converges for smoothers that correspond to a symmetric smoothing matrix with all eigenvalues in $(0, 1)$. This includes smoothing splines and most kernel smoothers, but not Loess.

If it converges, the solution is unique unless there is concurvity. In that case, the solution depends upon the initial conditions.

Concurvity occurs when the $\{\boldsymbol{x}_i\}$ values lie upon a smooth manifold in $\mathbb{R}^p$. In our context, a manifold is smooth if the smoother used in backfitting can interpolate all the $\{\boldsymbol{x}_i\}$ perfectly.

This is exactly analogous to the non-uniqueness of regression solutions when the $\boldsymbol{X}$ matrix is not full-rank.

Let $\boldsymbol{P}$ be an operator on p-tuples of functions $\boldsymbol{g} = (g_1, \ldots, g_p)$ and let $\boldsymbol{Q}$ be an operator on a function h. Then the **concurvity space** of

$$\boldsymbol{P}\boldsymbol{g} = \boldsymbol{Q}h$$

is the set of additive functions $g(\boldsymbol{x}) = \sum g_j(x_j)$ such that $\boldsymbol{P}\boldsymbol{g} = \mathbf{0}$. That is,

$$g_j(x_j) + \mathbb{E}[\sum_{k \neq j} g_k(x_k) \,|\, x_j] = 0.$$

We shall now consider several extensions of the general idea in additive modeling.

4.2 Generalized Additive Model

The generalized additive model assumes that the response variable $\boldsymbol{Y}$ comes from an exponential family (e.g., binomial or Poisson). This is like analysis with the generalized linear model of McCullagh and Nelder (1989; *Generalized Additive Models*, 2nd ed., Chapman and Hall).

Recall that in generalized linear models the explanatory values are related to the response through a link function g. If

$$\mu = \mathbb{E}[Y \mid \boldsymbol{X}], \;\; \textbf{then} \; g(\mu) = \alpha + \boldsymbol{x}'\boldsymbol{\beta}.$$

For example, if Y is Bernoulli, then $\mathbb{E}[Y \mid \boldsymbol{X} = \boldsymbol{x}] = p(\boldsymbol{x}) = \mathbb{P}[Y = 1 \mid \boldsymbol{x}]$. Then

$$g(p(\boldsymbol{x}) = \textbf{logit}(p(\boldsymbol{x}) = \ln \frac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})}$$

which yields logistic regression.

The generalized additive model expresses the link function as an additive, rather than linear, function of $\boldsymbol{x}$:

$$g(\boldsymbol{\mu}) = \beta_0 + \sum_{j=1}^{p} f_j(x_j).$$

As before, the link function is chosen by the user based on domain knowledge. Only the relation to the explanatory variables is modeled.

Thus an additive model version of logistic regression is

$$\mathbf{logit}(p(\boldsymbol{x})) = \beta_0 + \sum_{j=1}^{p} f_j(x_j).$$

Generalized linear models are fit by iterative scoring, a form of iteratively reweighted least squares. The generalized additive model modifies backfitting in a similar way (cf. Hastie and Tibshirani; 1990, *Generalized Additive Models*, chap. 6).

intentional blank

4.3 Projection Pursuit Regression

A different extension of the additive model is Projection Pursuit Regression (PPR). This treats models of the form:

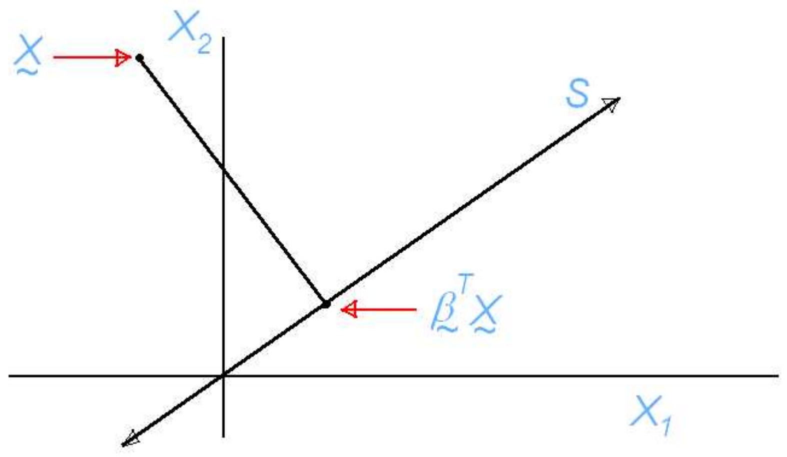
$$Y = \beta_0 + \sum_{j=1}^{r} f_j(\boldsymbol{\beta}'\boldsymbol{X}) + \epsilon$$

where r is found from the data by cross-validation, the f_j are backfitting smooths, and the $\boldsymbol{\beta}_j$ are predictive linear combinations of explanatory variables.

Friedman and Stuetzle (1981; *Journal of the American Statistical Association*, **76**, 817-823) based PPR on exploratory data analysis strategies used to rotate point clouds in order to visualize interesting structure.

PPR tends to work when the explanatory variables are commensurate; e.g., in predicting lifespan, similar biometric measurements might be bundled into one linear combination, and education measurements might form another.

Picking out a linear combination is equivalent to choosing a one-dimensional projection of $\boldsymbol{X}$. For example, take $\boldsymbol{\beta}' = (1, 1)$ and $\boldsymbol{x} \in \mathbb{R}^2$. The $\boldsymbol{\beta}'\boldsymbol{x}$ is the projection of $\boldsymbol{x}$ onto the subspace $\mathcal{S} = \{\boldsymbol{x} : x_1 = x_2\}$.



If $r = 1$, then the fitted PPR surface is constant along lines orthogonal to $\mathcal{S}$. If f_1 were the sine function, then the surface would look like corrugated aluminum, but oriented so that the ridges were perpendicular to $\mathcal{S}$.

When $r > 1$ the surface is hard to visualize, especially since the $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_r$ need not be mutually orthogonal. As $r \to \infty$, the PPR fit is a consistent estimator of smooth surfaces (Chen, 1991; *Annals of Statistics*, **19**, 142-157).

The PPR algorithm alternately applies backfitting (to estimate the f_j) and Gauss-Newton search (to estimate the β_j). It seeks $\{\hat{f}_j\}$ and $\{\hat{\boldsymbol{\beta}}_j\}$ that minimize:

$$\sum_{i=1}^{n} [Y_i - \sum_{j=1}^{r} \hat{f}_j(\hat{\boldsymbol{\beta}}_j' \boldsymbol{x}_j)]^2.$$

The algorithm assumes a fixed r, but this can be relaxed by doing univariate search on r.

The Gauss-Newton step starts with initial guesses for $\{\hat{f}_j\}$ and $\{\hat{\boldsymbol{\beta}}_j\}$ and uses the multivariate first-order Taylor expansion around the initial $\{\hat{\boldsymbol{\beta}}_j\}$ to improve the estimated projection directions.

The PPR algorithm works as follows:

1. Fix r.
1. Initialize. Get initial estimates for $\{\hat{f}_j\}$ and $\{\hat{\boldsymbol{\beta}}_j\}$.
2. Loop.

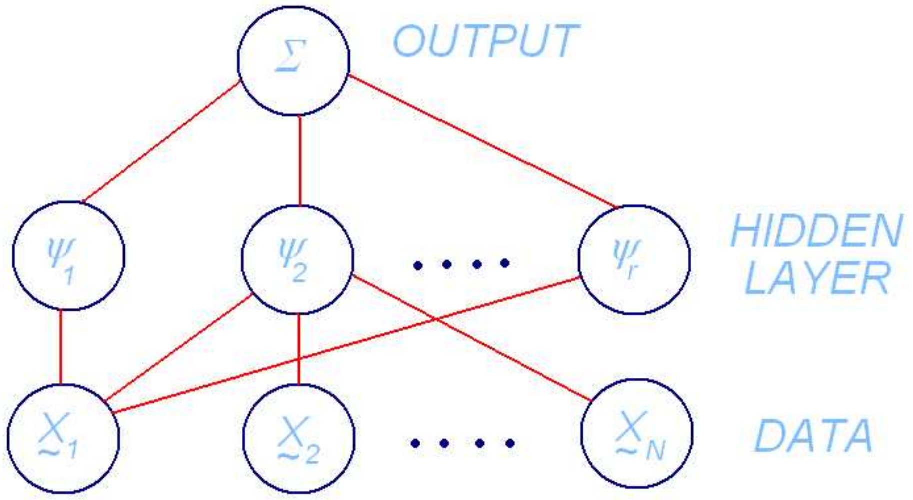
```
    For $j = 1, \ldots, r$ do:
    $\hat{f}_j(\hat{\boldsymbol{\beta}}_j' \boldsymbol{x}) = S(\boldsymbol{Y} - \sum_{j \neq k} \hat{f}_k(\hat{\boldsymbol{\beta}}_k' \boldsymbol{x}) \,|\, \boldsymbol{\beta}_j)$
    End For.
    Find new $\hat{\boldsymbol{\beta}}_j$ by Gauss-Newton
    If the maximum change in $\{\hat{\boldsymbol{\beta}}_j\}$ is less than some threshold, exit.
    End Loop.
```

This converges uniquely under essentially the same conditions as for the AM.

4.4 Neural Networks

A third version of the additive model is **neural networks**. These methods are very close to PPR.

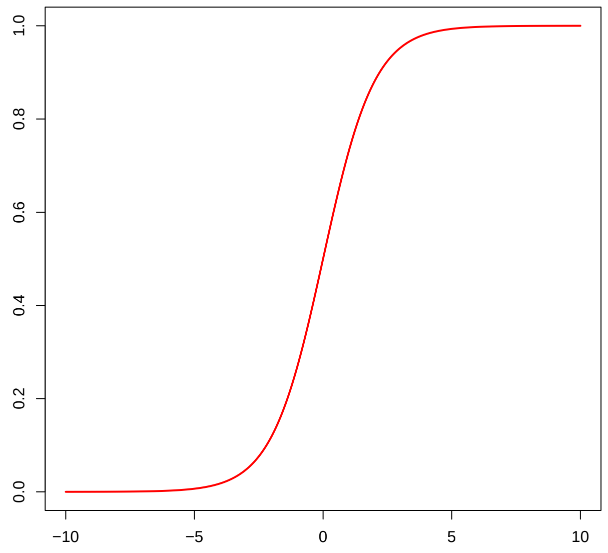
There are many different fiddles on the neural network strategy. We focus on the basic feed-forward network with one hidden layer.



Neural networks fit a model of the form

$$Y = \beta_0 + \sum_{j=1}^{r} \gamma_j \psi(\boldsymbol{\beta}_j' \boldsymbol{x} + \nu_j)$$

where ψ is a sigmoidal (or logistic) function and the other parameters (except r) are estimated from the data.



The only difference between PPR and the neural net is that neural nets assumes that the additive functions have a parametric (logistic) form:

$$\psi(\boldsymbol{x}) = \frac{1}{1 + \exp(\alpha_0 + \boldsymbol{\beta}'\boldsymbol{x})}.$$

The parametric assumption allows neural nets to be trained by backpropagation, an iterative fitting technique. This is very similar to backfitting, but somewhat faster because it does not require smoothing.

Barron (1993; *IEEE Transactions on Information Theory*, **39**, 930-945) showed that neural networks evade the Curse of Dimensionality in specific, rather technical, sense. We sketch his result.

A standard way of assessing the performance of a nonparametric regression procedure is in terms of **Mean Integrated Square Error** (MISE). Let $g(\boldsymbol{x})$ denote the true function and $\hat{g}(\boldsymbol{x})$ denote the estimated function. Then

$$\mathbf{MISE}[\hat{g}] = \mathbb{E}_F \left[\int [\hat{g}(\boldsymbol{x}) - g(\boldsymbol{x})]^2 \, d\boldsymbol{x} \right]$$

where the expectation is taken with respect to the randomness in the data $\{(Y_i, \boldsymbol{X}_i)\}$.

Before Barron's work, it had been thought that the COD implied that for any regression procedure, the MISE had to grow faster than linearly in p, the dimension of the data. Barron showed that neural networks could attain an MISE of order $\mathcal{O}(r^{-1}) + \mathcal{O}(rp/n) \ln n$ where r is the number of hidden nodes.

Recall that $a_n = \mathcal{O}(h(n))$ means there exists c such that for n sufficiently large, $a_n \leq ch(n)$. Thus Barron's error increases only linearly in p.

Barron's theorem is technical. It applies to the class of functions $g \in \Gamma_c$ on $\mathbb{R}^p$ whose Fourier transforms $\tilde{g}(\omega)$ satisfy

$$\int |\omega| \tilde{g}(\omega)\, d\omega \leq c$$

where the integral is in the complex domain and $|\cdot|$ denotes the complex modulus.

The class Γ_c is **thick**, meaning that it cannot be parameterized by a finite-dimensional parameter. But it excludes important functions such as hyperflats.

The strategy in Barron's proof is:

- Show that for all $g \in \Gamma_c$, there exists a neural net approximation $\hat{g}^*$ such that $\|g - \hat{g}^*\|^2 \leq c^*/n$.
- Show that the MISE in estimating any of the $\hat{g}^*$ functions is bounded.
- Combine these results to obtain a bound on the MISE of a neural net estimate $\hat{g}$ for an arbitrary $g \in \Gamma_c$.

4.5 ACE and AVAS

Alternating Conditional Expectations (ACE) seeks transformations $f_1, \ldots, f_p$ and g of the p explanatory variables and the response variable Y that maximize the correlation between

$$g(Y) \text{ and } \sum_{j=1}^{p} f_j(X_j).$$

This is equivalent to minimizing

$$\mathbb{E}[(g(Y) - \sum_{j=1}^{p} f_j(X_j))^2]/\mathbb{E}[g^2(Y)]$$

where the expectations are taken with respect to $\{(Y_i, \boldsymbol{X}_i)\}$.

ACE modifies the additive modeling strategy by

- allowing arbitrary transformations of the response variable,
- using maximum correlation, squared error, for optimization.

ACE was developed by Breiman and Friedman (1985; *Journal of the American Statistical Association*, **80**, 580-619). It resembles canonical correlation.

The ACE algorithm works as follows:

1. Initialize. Set $g(y_i) = (y_i - \bar{y})/s_y$; set $f_j(x_j)$ as the regression of Y on X_j.
2. Backfit. Fit an additive model to $g(y)$ to obtain new functions $f_1(x_1), \ldots, f_p(x_p)$.
3. Compute. Use smoothing to estimate

   $$\tilde{g}(y) = \mathbb{E}[\sum_{j=1}^{p} f_j(x_j) \,|\, Y_i = y_i]$$

   and standardize a new $g(y)$ as

   $$g(y) = \tilde{g}(y)/\sqrt{\mathrm{Var}\,[\tilde{g}(y)]}.$$

   (This standardization ensures that the trivial solution $g \equiv 0$ does not arise.)
4. Alternate. Do steps 2 and 3 until $\mathbb{E}[(g(Y) - \sum_{j=1}^{p} f_j(X_j))^2]$ converges.

This finds the unique optimal solution, given by the eigenfunctions associated with the largest eigenvalue of a certain operator.

intentional blank

intentional blank

From the standpoint of nonparametric regression, ACE has several undesirable features.

- If $g(Y) = f(X) + \epsilon$, then ACE generally will not find g and f but rather $h \circ g$ and $h \circ f$.
- The solution is sensitive to the marginal distributions of the explanatory variables.
- ACE treats the explanatory and response variables in the same way, but regression should be asymmetric.
- The eigenfunctions for the second-largest eigenvalue can provide better insight on the problem.

There are a few other pathologies. See the discussion of the Breiman and Friedman article for additional examples and details.

Additivity and Variance Stabilization (AVAS) is a modification of ACE that removes most of the undesirable features. It was developed by Tibshirani (1988; *Journal of the American Statistical Association*, **83**, 394-405).

Heteroscedasticity is a common problem in regression and lies at the root of ACE's difficulties.

It is known that if a family of distributions for Z has mean μ and variance $V(\mu)$, then the asymptotic variance stabilizing transformation for Z is

$$h(t) = \int_0^t V(s)^{-1/2}\, ds.$$

The AVAS algorithm is like the ACE algorithm except that in step 3 it applies the estimated variance stabilizing transformation to $\tilde{g}(Y)$ before standardization.

4.6 Recursive Partitioning Regression

Partition methods are designed to handle surfaces with significant interaction structure. The most famous of these methods is CART, for Classification and Regression Trees (Breiman, Friedman, Olshen, and Stone; 1984, Wadsworth).

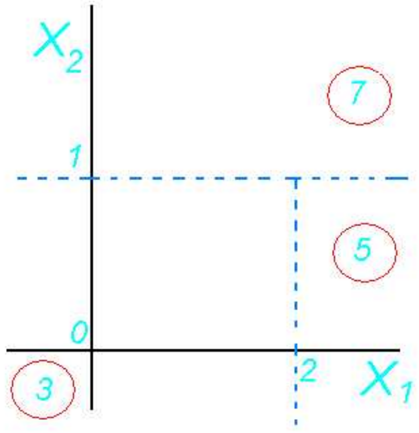
In regression, CART acts as a smart bin-smoother that performs automatic variable selection. Formally, it fits the model

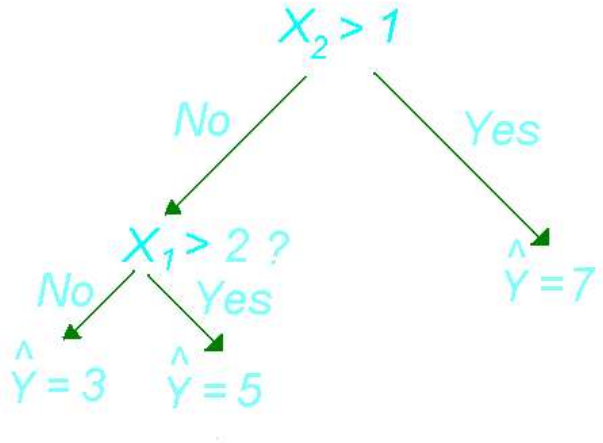
$$Y = \sum_{j=1}^{r} \beta_j I(\boldsymbol{x} \in R_j) + \epsilon$$

where the regions R_j and the coefficients β_j are estimated from the data. Usually the R_j are disjoint and the β_j is the average of the Y values in R_j.

The CART model produces a decision tree that is helpful in interpreting the results, and this is one of the keys to its enduring popularity.

The following partition and tree are equivalent:





Note that CART focuses on a different kind of interaction than the usual product term popular in multiple regression; it looks for thresholds. This leads to results that are not smooth, which some feel is a drawback in regression.

The CART algorithm has three parts:

1. A way to select a split at each intermediate node.
2. A rule for declaring a node to be terminal.
3. A rule for estimating Y at a terminal node.

The third part is easy—just use the sample average of the cases at that terminal node.

The first part is also easy—split on the value x_j^* which most reduces

$$SS\mathbf{error} = \sum_{i=1}^{n} (y_i - \hat{f}_c(\boldsymbol{x}_i))^2$$

Where $\hat{f}_c$ is the predicted value from the current tree.

The second part is the hard one. One must grow an overly complicated tree, and then use a pruning rule and cross-validation to find a tree with good predictive accuracy. This entails a complexity penalty.

The main problems with CART are:

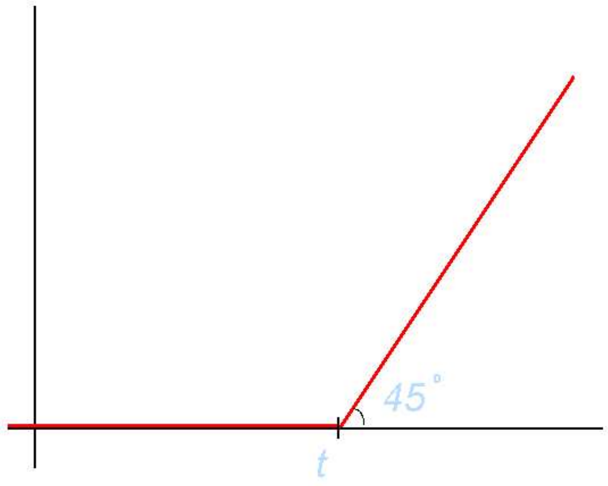
- discontinuous boundaries;
- it is difficult to approximate functions that are linear or additive in a small number of variables;
- it is usually not competitive in low dimensions;
- one cannot tell when a complex CART model is close to a simple model.

The Boston housing data gives an example of the latter situation.

4.7 MARS

Multivariate Adaptive Regression Splines (MARS) improves on CART by marrying it to PPR. It uses multivariate splines to let the data find flexible partitions or $\mathbb{R}^p$. And it incorporates PPR by letting the orientation of the region be non-parallel to the natural axes.

The basic building block is a "hockeystick" (first-order truncated basis) function $(x-t)^+$, which looks like:



This is a special kind of tensor spline. It has a knot at t.

The fitted model has the form

$$Y = \sum_{j=1}^{r} \beta_j B_m(\boldsymbol{x}) + \epsilon$$

where

$$B_m(\boldsymbol{x}) = \prod_{k \in \mathcal{K}} [s_{km}(x_{km} - t_{km})]^{+}$$

for $s_k m = \pm 1$ and $\mathcal{K}$ is a subset of the explanatory variables. Thus B_m is a product of hockeysticks, so $\hat{f}$ is continuous. The regions are determined by the knots $\{t_{km}\}$.

The MARS algorithms starts with $B_1(\boldsymbol{x}) = 1$ and constructs new terms until there are too many, as measured by generalized cross-validation.

Empirically, Friedman found that each term fit in a MARS model costs between 2 and 4 degrees of freedom. This reflects the variable selection involved in the fitting, but not smoothing.

For each basis function B_m:

1. For all variables not in B_m,
   - **A.** try putting $\pm$ hockeysticks at every observation (candidate knot) in the non-zero region of B_m;
   - **B.** select the best pair of hockeysticks according to an estimate of lack-of-fit.
2. Make two new basis functions from the product of B_m and the chosen pair of hockeystick functions.

After building too many basis functions, prune back as in CART using cross-validation.

Hockeysticks enter in pairs so that MARS is equivariant to sign changes.

Each B_m contains at most one term from each explanatory variable.

intentional blank

MARS is sort of interpretable, via an ANOVAesque decomposition:

$$\hat{f}(\boldsymbol{x}) = \beta_0 + \sum_{j \in \mathcal{J}} f_j(x_j) + \sum_{(j,k) \in \mathcal{K}} f_{jk}(x_j, x_k) + \dots$$

where β_0 is the coefficient of the B_1 basis function, the first sum is over those basis functions that involve a single explanatory variable, the second sum is over those basis functions that involve exactly two explanatory variables, and so forth.

These terms can be thought of as the grand mean, the main effects, the two-way interactions, etc. One can plot these functions to gain insight.

MARS uses a tensor product basis (hence no explanatory variable appears twice in any B_m). Additive effects are captured by splitting B_1 on several variables. Nonlinear effects are captured by splitting B_1 with the same variable more than once.

MARS and CART are available commercially from Salford Systems, Inc., and versions of them are available in S^+ and R.

To get an integrated suite of code that that performs ACE, AVAS, MARS, PPR, neural nets (the CASCOR version), recursive partitioning, and Loess, one can download the DRAT package from

www-2.cs.cmu.edu/~bobski/software/software.html

There are many kinds of neural network code. Users should be careful in using it—there are many possible twiddles and performance may vary. CASCOR is due to Fahlman and Lebiere (1990; *Advances in Neural Information Processing Systems 2*, 524-532, Morgan Kaufmann), and has the convenience of adaptively choosing the number of hidden nodes.

5. Comparing Methods

We have described eight methods so far: Loess, Additive Models (AM), PPR, Neural Nets (NN), ACE, AVAS, Recursive Partitioning Regression (RPR), and MARS. In addition, a practitioner might consider traditional stepwise linear regression (SLR) and multiple linear regression (MLR).

One would like to make comparisons among these. In general, one would like to have a practicum for making comparisons among complex statistical procedures that are too difficult to analyze using theory.

This section describes a model simulation experiment, and it lays out the general strategies for such comparisons.

Banks, Olszewski, and Maxion (2003; *Communications in Statistics: Simulation and Computation*, **32**, 541-571) describe a $10 \times 5 \times 3^4$ designed experiment to compare the methods.

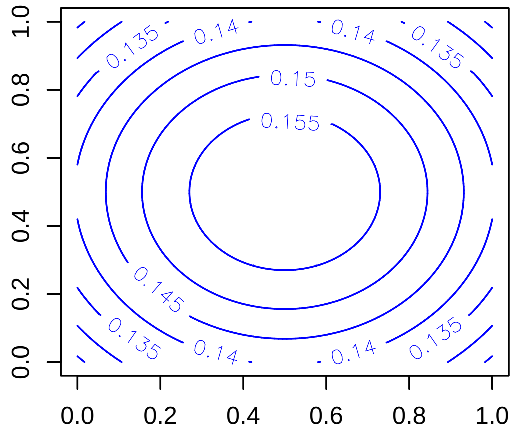
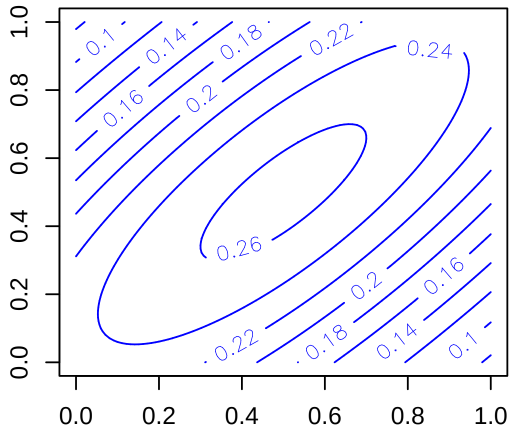
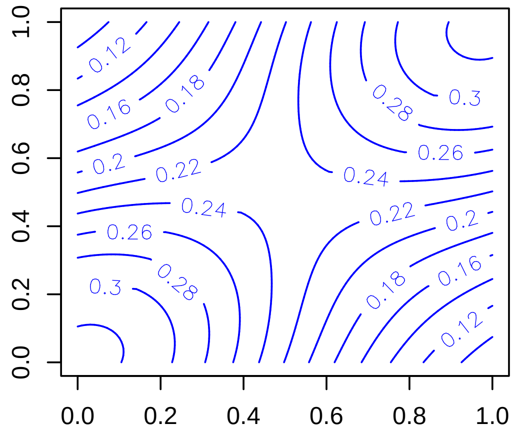
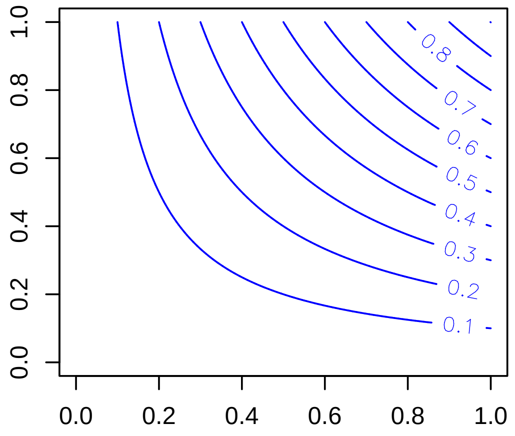
The six factors in the experiment were:

1. The ten regression methods.
2. Five target functions.
3. The dimension: $p = 2, 6, 12$.
4. The sample size: $n = 2^p k$ for $k = 4, 10, 25$.
5. The proportion of spurious variables: this took the values all, half, and none.
6. The noise: this is the variance in ϵ, the additive Gaussian noise, and took the values $\sigma = .02$, $.1$, $.5$.

The target functions consist multivariate functions that approximate features likely to be of scientific interest. The five functions are

- the constant function,
- the hyperflat,
- the standard normal recentered to $(.5, \ldots, .5)'$,
- a normal centered $(.5, \ldots, .5)'$ with covariance matrix $.8\boldsymbol{I}$
- a mixture of two standard normals, one centered at $(0, \ldots, 0)'$ and the other at $(1, \ldots, 1)'$
- the product function $x_1 x_2 \cdots x_p$.

The constant function is especially important because in that case the explanatory values are irrelevant, but some methods tend to discover spurious signal in the noise.


0.135
0.14
0.15
0.155
0.145

0.1
0.14
0.16
0.18
0.2
0.22
0.24
0.26

0.12
0.16
0.18
0.2
0.22
0.24
0.26
0.28
0.3

0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

The simulation experiment had the following steps:

1. Generate a random sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ uniformly in the unit cube in $\mathbb{R}^p$.
2. Generate random $N(0, \sigma^2)$ errors.
3. Calculate $Y_i = f(\boldsymbol{x}_i) + \epsilon_i$, where f is one of the target functions.
4. Apply each of the regression methods to obtain estimate $\hat{f}$ of f.
5. Estimate the integrated squared error of each $\hat{f}$ over the unit cube (Monte Carlo, using 10,000 random points).
6. Repeat steps 1-5 twenty times and average to obtain an estimate of MISE.

Note that this procedure reuses the same data points across the estimators, which reduces the variance in contrasts.

σ	n	Sp.	Meth.	MISE	SE	MISE	SE	MISE	SE
				$p = 2$		$p = 6$		$p = 12$	
M	S	A	MLR	(27.95)	7.02	(3.00)	.26	(.09)	.01
M	S	A	SLR	(32.14)	10.65	(3.00)	.26	(.09)	.01
M	S	A	ACE	53.12	7.36	15.06	1.14	.41	.02
M	S	A	AM	(27.91)	7.01	(3.00)	.26	(.09)	.01
M	S	A	MARS	158.23	35.32	18.36	1.88	.30	.02
M	S	A	RPR	1248.37	307.78	176.32	10.89	61.53	.78
M	S	A	PPR	55.08	13.14	19.24	8.68	.11	.01
M	S	A	LOESS	59.50	9.12	9.14	.52	*	*
M	S	A	AVAS	79.40	18.14	14.73	.95	.38	.02
M	S	A	NN	124.03	13.05	100.42	2.43	*	*

σ	n	Sp.	Meth.	MISE	SE	MISE	SE	MISE	SE
				$p = 2$		$p = 6$		$p = 12$	
M	S	H	MLR	27.95	7.02	3.00	.26	(.09)	.01
M	S	H	SLR	(23.23)	6.75	(2.39)	.30	(.08)	.01
M	S	H	ACE	42.42	5.92	14.58	1.12	.40	.02
M	S	H	AM	27.68	7.00	3.00	.26	(.09)	.01
M	S	H	MARS	(17.99)	3.24	11.00	1.68	.40	.02
M	S	H	RPR	1821.76	64.32	239.41	4.50	100.04	2.47
M	S	H	PPR	35.31	5.61	13.95	8.09	.11	.01
M	S	H	LOESS	59.50	9.12	9.14	.52	*	*
M	S	H	AVAS	74.93	14.13	15.02	1.08	.36	.01
M	S	H	NN	103.08	8.41	94.92	2.65	*	*

MLR, SLR, and AM perform similarly over all situations considered, and represent broadly safe choices. They are never disastrous, though rarely the best (except for MLR when the target function is linear and all variables are used). For the constant function, SLR shows less overfit than MLR, which is better than AM; however, it is easy to find functions for which AM would outperform both MLR and SLR. SLR is usually slightly better with spurious variables, but its strategy becomes notably less effective as the number of spurious variables increases, especially for non-linear functions. All three methods have greatest relative difficulty with the product function, which has substantial curvature.

On theoretical grounds ACE and AVAS should be similar, but this is not always borne out. ACE is decisively better for the product function, and AVAS for the constant function. ACE and AVAS are the best methods for the product function (as expected—the log transformation produces a linear relationship), but among the worst for the constant function and the mixture of Gaussians; in other cases, their performance is not remarkable. Both methods are fairly robust to spurious variables.

Contrary to expectation, MARS does not show well in higher dimensions, especially when all variables are used, and especially for the linear function. However, for lower dimensions, MARS shows adequate performance across the different functions. MARS is well-calibrated for the constant function when $p = 2$, but finds spurious structure for larger values, which may account for some of its failures.

RPR was consistently bad in low dimensions, but sometimes stunningly successful in high dimensions, especially when all variables were used. Surprisingly, its variable selection capability was not very successful (MARS's implementation clearly outperforms it). Perhaps the CART program, with its flexible pruning, would surpass RPR, but previous experience with CART makes us dubious. Unsurprisingly, RPR's design made it uncompetitive on the linear function.

PPR and NN are theoretically similar methods, but PPR was clearly superior in all cases except the correlated Gaussian. This may reflect peculiarities of the Cascor implementation of neural nets. PPR was often among the best when the target function was the Gaussian, correlated Gaussian, or mixture of Gaussians, but among the worst with the product and constant functions. PPR's variable selection was generally good. In contrast, NN was generally poor, except for the correlated Gaussian when $p = 2, 6$ and all variables are used and when $p = 6$ and half the variables are used. The correlated Gaussian lends itself to approximation by a small number of sigmoidal functions whose orientations are determined by the data.

LOESS does well in low dimensions with the Gaussian, correlated Gaussian, and mixture of Gaussians. It is not as successful with the other target functions, especially the constant function. Often, it is not bad in higher dimensions, though its relative performance tends to deteriorate.

For the constant function, MARS is good when $p = 2$, SLR is good when $p = 6$, and RPR is good when $p = 12$. For the linear function, MLR and SLR are consistently strong. For the Gaussian function, with all variables used, LOESS and MARS are good when $p = 2$, SLR is good when $p = 6$, and RPR is good when $p = 12$; when half of the variables are used, MARS and PPR perform well. For the correlated Gaussian, with all variables used, LOESS works well for $p = 2$, LOESS and NN for $p = 6$, and ACE or AVAS for $p = 12$; with half the variables used, MARS is reliably good. For the mixture of Gaussians, with all variables used, LOESS works well for $p \leq 6$, and RPR for $p = 12$; with half of the variables, MARS is consistently good. There is considerable variability for the product function, but ACE is broadly superior.

Two kinds of variable selection strategies were used by the methods: global variable selection, as practiced by SLR, ACE, AVAS, and PPR, and local variable reduction, as practiced by MARS and RPR. Generally, the latter does best in high dimensions, but performance depends on the target function.

LOESS, NN, and sometimes AVAS proved infeasible in high dimensions. The number of local minimizations in LOESS grew exponentially with p. Cascor's demands were high because of the cross-validated selection of the hidden nodes; alternative NN methods fix these a priori, making fewer computational demands, but this is equivalent to imposing strong, though complex, modeling assumptions. Typically, fitting a single high-dimensional dataset with either LOESS or NN took more than two hours. AVAS was faster, but the combination of high dimension and large sample size also required substantial time.

intentional blank

6. Local Dimension

Nearly all methods for multivariate nonparametric regression do local model fitting (otherwise, they must make strong model assumptions, such as multiple linear regression). Local fitting is most likely to succeed if the response function $f(\boldsymbol{x})$ has **locally-low dimension**.

A function $f : \mathbb{R}^p \to \mathbb{R}$ has locally-low dimension if there exists a set of regions $R_1, R_2, \ldots$ and a set of functions $g_1, g_2, \ldots$ such that $\bigcup R_i \approx \mathbb{R}^p$ and for $\boldsymbol{x} \in R_i$, $f(\boldsymbol{x}) \doteq g_i(\boldsymbol{x})$ where g_i depends upon only q components of $\boldsymbol{x}$ for $q \ll p$.

This uses a vague sense of approximation, but it can be made precise.

The following functions have high local dimension:

$$\begin{aligned} f(\boldsymbol{x}) &= \beta_0 + \sum_{j=1}^{p} \beta_j x_j \ \textbf{for} \ \beta_j \neq 0 \\ f(\boldsymbol{x}) &= \prod_{j=1}^{p} x_j. \end{aligned}$$

In contrast, the following functions have locally-low dimension:

$$\begin{aligned} f(\boldsymbol{x}) &= \begin{cases} 3x_1 & \textbf{if } x_1 + x_2 < 7 \\ x_2^2 & \textbf{if } x_1 + x_2 > 7 \\ x_1 + & \textbf{if } x_1 = x_2 \end{cases} \\ f(\boldsymbol{x}) &= \sum_{k=1}^{m} \alpha_k I_{R_k}(\boldsymbol{x}). \end{aligned}$$

Regression analysis in high dimensions seems impossible without strong model assumptions or locally-low dimension.

Before attempting a statistical analysis, it would be good to know whether the local dimension is low. (If not, one should anticipate disappointment.) So how can one estimate the local dimension?

Let $\{(y_i, \boldsymbol{x}_i)\}$ denote the sample. Then iterate the following steps M times.

1. Select a random point $\boldsymbol{X}_m^*$ in the convex hull of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, for $m = 1, \ldots, M$
2. Find a ball centered at $\boldsymbol{X}^*$ that contains exactly k points (say $k = 4p$).
3. Perform a principal components regression on the k points within the ball.
4. Let c_m be the number of principal components needed to explain a fixed percentage (say 80%) of the variance in the Y_i values.

This suggests that the average of $c_1, \ldots, c_M$ may be a useful estimate of the average local dimension of f.

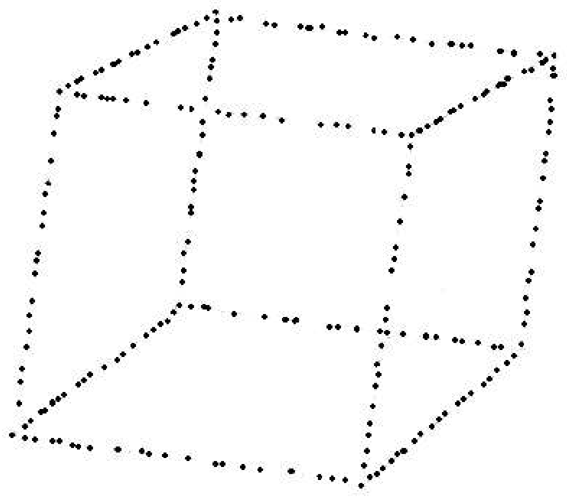
This heuristic approach assumes a locally linear functional relationship for points within the ball. The Taylor series motivates this, but the method will break down for some pathological functions or if the data are too sparse.

To test the approach, Banks and Olszewski (2003; *Statistical Data Mining and Knowledge Discovery*, 529-548, Chapman & Hall) performed a simulation experiment in which random samples were generated from q-cube submanifolds in $\mathbb{R}^p$, and the approach described above was used to estimate q.

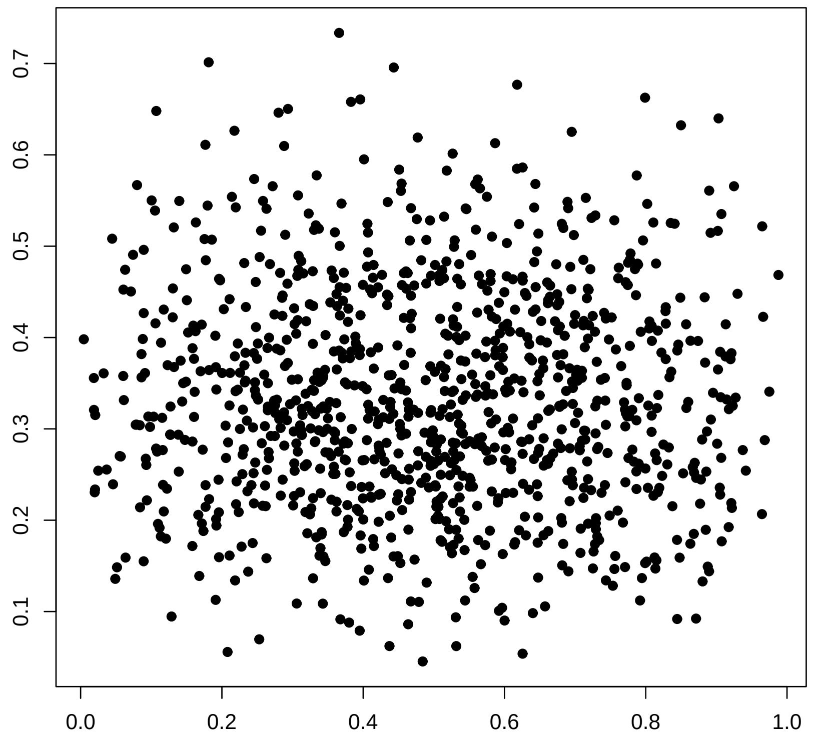
A **q-cube** in $\mathbb{R}^p$ is the q-dimensional boundary of a p-dimensional cube. Thus:

- a 1-cube in $\mathbb{R}^2$ is the perimeter of a square,
- a 2-cube in $\mathbb{R}^3$ are the faces of a cube,
- a 3-cube in $\mathbb{R}^3$ is the entire cube.

The following figure shows a 1-cube in $\mathbb{R}^3$, tilted 10 degrees from the natural axes in each coordinate.



The following figure shows a 1-cube in $\mathbb{R}^{10}$, tilted 10 degrees from the natural axes in each coordinate.



Diaconis and Freedman (1984; *Annals of Statistics*, **12**, 793-815) show that in high-dimensions, nearly all projections look normal.

The simulation study generated $10 * 2^q$ points at random on each of the $2^{p-q} \begin{pmatrix} p \\ q \end{pmatrix}$ sides of a q-cube in $\mathbb{R}^p$. Then iid $N(\mathbf{0}, .25\boldsymbol{I})$ noise was added to each observation and the principal components approach was used to estimate q for all values of q between 1 and p for $p = 1, \ldots, 7$.

The first following table shows that the method was reasonably successful in estimating the local dimension. The estimates are biased, since the principal components analysis identified the number of linear combinations needed to explain only 80% of the variance. One should probably do some kind of bias correction to account for the underestimate.

The second following table estimates the proportion of the data region that is sparse; the method puts random balls of fixed size into the dataset and counts the number of times the ball contains fewer than $2p$ observations.

q							
7							5.03
6						4.25	4.23
5					3.49	3.55	3.69
4				2.75	2.90	3.05	3.18
3			2.04	2.24	2.37	2.50	2.58
2		1.43	1.58	1.71	1.80	1.83	1.87
1	.80	.88	.92	.96	.95	.95	.98
	p=1	2	3	4	5	6	7

The value of p indicates the apparent dimension, while q is the true dimension of the data. Each entry is an estimate of q, and the largest standard error in the table is .03.

q							
7							41.0
6						39.1	52.2
5					34.4	45.3	53.2
4				32.3	36.2	46.1	55.9
3			29.1	27.0	34.7	48.6	57.9
2		28.4	26.5	32.0	41.6	46.6	55.2
1	28.5	40.1	45.8	51.0	51.3	51.0	52.5
	p=1	2	3	4	5	6	7

The value of p indicates the apparent dimension, while q is the true dimension of the data. The number in a row indicates the proportion of spheres that did not contain at least $2p$ observations.

7. Classification

Classification problems attempt to find rules that assign cases to categories, e.g.:

- correct versus incorrect tax returns
- medical diagnoses
- good versus bad credit risks
- terrorists versus non-terrorists.

One uses a training sample of cases for which the categories are known. Each case has a vector of covariates that are used to build the classification rule.

For a new observation, one looks at its covariate and classifies it according to the classification rule.

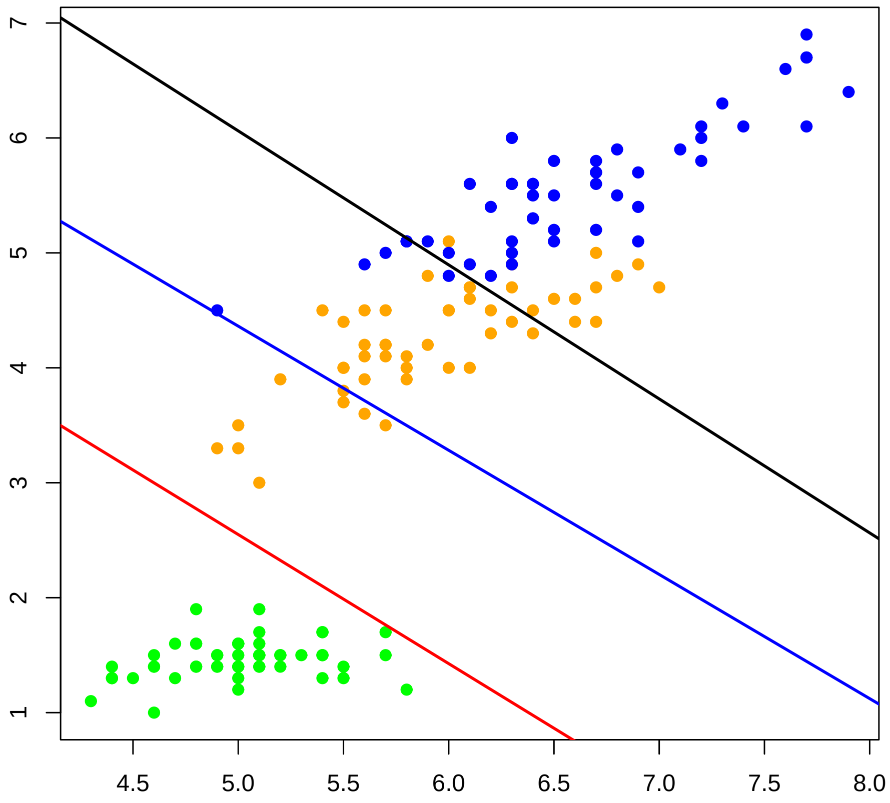
There are three main strategies for building classification rules:

- Geometric, which includes discriminant analysis, flexible discriminant analysis, and recursive partitioning.
- Algorithmic, which includes neural nets, nearest neighbor rules, support vector machines, and random forests.
- Probabilistic, which includes Bayes rules and hidden Markov models.

The probabilistic methods make strong assumptions about having random samples of data.

Most classification problems are very similar to regression problems. For problems with two categories, it is sometimes just a matter of fitting a model that predicts 1 or -1 according to the case label.

The geometric rules are easy to visualize. They started with Fisher's classification of iris species (1936; *Eugenics*, **7**, 179-188).



This plot uses only two of the four covariates Fisher recorded.

For two classes, Fisher's linear discriminant analysis assumes that the two populations have multivariate normal distribution with common unknown covariance matrix and different unknown mean vectors. It assigns a new observation to the population whose mean has the smallest Mahalanobis distance to the observation:

$$d_M(\boldsymbol{x}_i, \bar{\boldsymbol{x}}_j) = [(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_j)' \boldsymbol{S}^{-1} (\boldsymbol{x}_i - \bar{\boldsymbol{x}}_j)]^{1/2}$$

To analyze the effect of noise in linear discriminant analysis, suppose one has a fixed sample size n and assume the covariance matrices are known to be $\sigma^2 \boldsymbol{I}$. Write the estimates of the means as:

$$\begin{aligned} \hat{\boldsymbol{\mu}}_1 &= \boldsymbol{\mu}_1 + \frac{\sigma}{\sqrt{n}} \boldsymbol{v}_1 \\ \hat{\boldsymbol{\mu}}_2 &= \boldsymbol{\mu}_2 + \frac{\sigma}{\sqrt{n}} \boldsymbol{v}_2. \end{aligned}$$

Also, write the new observation to classify as:

$$\boldsymbol{x} = \boldsymbol{\mu}_1 + \boldsymbol{v}.$$

Fisher's classification rule assigns population 1 if

$$d_M(\boldsymbol{x}, \hat{\boldsymbol{\mu}}_1) < d_M(\boldsymbol{x}, \hat{\boldsymbol{\mu}}_2)$$

and under our assumptions, this is equivalent to:

$$(\boldsymbol{x} - \hat{\boldsymbol{\mu}}_1)'(\boldsymbol{x} - \hat{\boldsymbol{\mu}}_1) < (\boldsymbol{x} - \hat{\boldsymbol{\mu}}_2)'(\boldsymbol{x} - \hat{\boldsymbol{\mu}}_2).$$

Writing $\boldsymbol{x}$, $\hat{\boldsymbol{\mu}}_1$ and $\hat{\boldsymbol{\mu}}_2$ in terms of $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, and $\boldsymbol{v}$ shows this criterion is equivalent to:

$$(\boldsymbol{v} - \frac{\sigma}{\sqrt{n}}\boldsymbol{v}_1)'(\boldsymbol{v} - \frac{\sigma}{\sqrt{n}}\boldsymbol{v}_1) < (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) + \boldsymbol{v} - \frac{\sigma}{\sqrt{n}}\boldsymbol{v}_2)'(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) + \boldsymbol{v} - \frac{\sigma}{\sqrt{n}}\boldsymbol{v}_2)$$

or, after further simplification,

$$\left[(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - \frac{\sigma}{\sqrt{n}}(\boldsymbol{v}_1 + \boldsymbol{v}_2) + 2\sigma\boldsymbol{v}\right]' \cdot \left[(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - \frac{\sigma}{\sqrt{n}}(\boldsymbol{v}_1 - \boldsymbol{v}_2) + 2\sigma\boldsymbol{v}\right] > 0.$$

As $n \to \infty$, this criterion converges to

$$2\sigma \boldsymbol{v}'(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) > \|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|^2$$

and thus the asymptotic probability of misclassification is $\mathbb{P}[\boldsymbol{v} > \|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|/2\sigma]$. Thus the error rate depends on the signal-to-noise ratio $\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|/\sigma$.

However, without using asymptotics, then the rule assigning population 1 can be written as

$$2\sigma \boldsymbol{v}'(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2 + \sqrt{\frac{2}{n}}\sigma \boldsymbol{v}_1) > -\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|^2 + \frac{2}{n}\sigma^2 \boldsymbol{v}_1'\boldsymbol{v}_2$$

so that the probability of misclassification is

$$1 - \Phi\left[-\frac{1}{2\sigma}\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|\left(1 + \frac{2\sigma^2}{\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|^2}\frac{p}{n}\right)^{1/2}\right].$$

For $p >> n$, this error is very large. (See S. Raudys, *Journal of Multivariate Analysis*, **89**, 1-35).

intentional blank

A very reasonable approach to classification is kth nearest-neighbor classification. In order to make a prediction about a new observation, one looks at the labels of its k nearest neighbors and uses a majority vote to make the prediction.

As the number of neighbors used in making the prediction increases, the decision boundaries become smoother—the bias increases, but the variance decreases. The "degrees-of-freedom" for nearest-neighbor rules is n/k.

The following images show how nearest-neighbor classification rules peform on data simulated from a 20-component mixture of normals, half of which were red, half of which were green. The effect of the bias-variance tradeoff is quite clear.

intentional blank

intentional blank

intentional blank

intentional blank

intentional blank

intentional blank

intentional blank

7.1 Support Vector Machines

Support Vector Machines (SVMs) were invented by Vapnik (1996; *The Nature of Statistical Learning*, Springer). They use optimization methods to find surfaces that best separate categories.

SVMs have been developed in many different directions, and have become quite elaborate. This survey focuses upon classifying two groups, and develop the ideas in three steps:

- Linear machines trained on separable classes;
- Linear machines trained on overlapping classes;
- Nonlinear machines.

7.1.1 Linear Machines, Separable Data

Suppose the n observations are $\{(\boldsymbol{x}_i, y_i)\}$ where $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$. And suppose one seeks a simple linear classification rule of the form

$$g(\boldsymbol{x}) = \mathbf{sign}[\boldsymbol{\beta}^T \boldsymbol{x} + \beta_0]$$

where $\boldsymbol{\beta}$ is determined from the data. (WLOG, assume $\|\boldsymbol{\beta}\| = 1$.)

If the two classes are separable then there are many possible $\boldsymbol{\beta}$ that work. A natural strategy is to pick the $\boldsymbol{\beta}$ that creates the biggest margin between the two classes.

The margin is twice perpendicular distance from the closest value with label $+1$ to the separating hyperflat (or the sum of the two smallest distances, one from each class).

intentional blank

Denote the margin by d. Then the optimization problem is to

$$\begin{aligned} &\max_{\boldsymbol{\beta},\beta_0,\|\boldsymbol{\beta}\|=1} \quad d \\ \textbf{subject to} \qquad & y_i(\boldsymbol{\beta}^T\boldsymbol{x}_i+\beta_0)\geq d, \quad i=1,\dots n. \end{aligned}$$

One can rewrite this as a convex optimization problem with with a quadratic criterion and linear constraints:

$$\begin{aligned} &\min \ \|\boldsymbol{\beta}\| \\ \textbf{subject to} \qquad & y_i(\boldsymbol{\beta}^T\boldsymbol{x}_i+\beta_0)\geq 1, \quad i=1,\dots n \end{aligned}$$

where the requirement that $\boldsymbol{\beta}$ have unit norm is dropped. It turns out that $d=\|\boldsymbol{\beta}\|^{-1}$.

Note that

- this solution is not equivalent to the one obtained from linear discriminant analysis;
- the solution depends only upon the two closest points in the two classes.

To solve the rewritten problem we can use a Lagrangian:

$$L = \frac{1}{2}\|\boldsymbol{\beta}\|^2 - \sum_{i=1}^{n} \lambda_i y_i(\boldsymbol{x}_i'\boldsymbol{\beta} + \beta_0) + \sum_{i=1}^{n} \lambda_i$$

The goal is to minimize L with respect to $\boldsymbol{\beta}$ and β_0, while simultaneously requiring $\lambda_i \geq 0$ and that the derivatives of L with respect to the λ_i vanish.

The Lagrangian formulation has two advantages:

- the original constraints are replaced by constraints on the Lagrange multipliers, which are easier to handle;
- the training data only appear once, as dot products in a sum, which allows generalization to nonlinear machines.

The dual problem of the Lagrangian minimization is to maximize L subject to:

$$\begin{aligned} 0 &= \frac{\partial L}{\partial \beta_i} \textbf{ for all } i \\ 0 &= \frac{\partial L}{\partial \beta_0} \\ 0 &\leq \lambda_i \textbf{ for all } i. \end{aligned}$$

This is called the Wolfe dual (see Fletcher, 1987, *Practical Methods of Optimization*, Wiley).

The zero-gradient requirement generates equality constraints, leading to

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j$$

which is maximized under the constraint that $\sum_{i=1}^{n} \lambda_i y_i = 0$ and $\lambda_i \geq 0$ for all i.

Training the SVM amounts to solving this optimization problem. We used a primal-dual solution, but anything that satisfies the Karush-Kuhn-Tucker conditions will be necessary and sufficient.

Note that for each observation, there is a Lagrange multiplier λ_i. Those observations with $\lambda_i > 0$ are the support vectors, and determine the margin. For all the other observations the λ_i are zero. If the problem were reworked using only the support vectors data, the same solution would be found.

Also note that the β_0 is determined implicitly. Solve

$$\lambda_i[y_i(\boldsymbol{\beta}'\boldsymbol{x}_i + \beta_0) - 1] = 0$$

using any support vector observation.

7.1.2 Linear Machines, Nonseparable Data

In practice, it usually happens that the two classes are not separable. In that case one wants to find the hyperflat that minimizes the sum of the perpendicular distances for the data that violate the rule. This leads to a slightly more advanced optimization problem that includes slack variables. This can be solved using Lagrange multipliers.

Cortes and Vapnik (1995; *Machine Learning*, **20**, 273-297) developed support vector machines to extend the optimization strategy we have described.

One can also view this as a problem in physics, in which each of the support vectors exerts a force on a rigid sheet. The equilibrium solution determines the separator.

7.1.3 Nonlinear Machines

The mathematics in the SVM literature can be serious, but the key idea is to find linear combinations of basis functions of $\boldsymbol{x}$ that describe good separating surfaces.

As previously noted, in real problems one wants more flexible separating surfaces than hyperflats. Boser, Guyon, and Vapnik (1992; *Fifth Annual Workshop on Computational Learning Theory*, ACM) showed how to do this in the SVM context, using an old idea of Aizerman et al. (1964; *Automation and Remote Control*, **25**, 821-837).

The main innovation is to express the surfaces in terms of a vastly expanded set of basis functions. SVMs map the problem to a higher dimensional space, and then solve the linear separation problem using nonlinear basis elements.

Consider the problem of building a classification rule for just two types of objects, where each object has measurements in $\mathbb{R}^p$. One can find separating hyperplanes or quadratic surfaces, as in linear or quadratic discriminant analysis, or one can fit more complex separating surfaces.

In principle, one would like to be able to fit very curvy surfaces, when the data warrant, that can separate complex structure in the two classes. Such surfaces can be formed by linear combinations of basis functions.

Recall that some of the standard basis sets are the Fourier basis, the Legendre polynomials, and so forth.

A basis set is set of functions $\{f_i\}$ such that any function g in the space can be written as

$$g(\boldsymbol{x}) = \sum_{i=1}^{\infty} \beta_i f_i(\boldsymbol{x})$$

and no element of $\{f_i\}$ can be written as a linear combination of the other elements.

The SVM strategy is to greatly expand the dimension of the input space beyond p, and then find hyperflats in that expanded space that classify the training sample. Then one maps back down to the lower-dimensional space, and it generally happens that the linear separating rules in the high-dimensional space become non-linear rules in the p dimensional space.

Consider expanding the set of inputs to include additional functions of the inputs, say $\boldsymbol{h}(\boldsymbol{x}_i) = (h_1(\boldsymbol{x}_i, \ldots, h_q(\boldsymbol{x}_i))^T$. If these are just a single component of the $\boldsymbol{x}_i$ data then $q = p$ and this reduces to the simple case described in 7.1.1.

One can show that the separating surface has the form

$$g(\boldsymbol{x}) = \sum_{i=1}^{n} \beta_i < \boldsymbol{h}(\boldsymbol{x}), \boldsymbol{h}(\boldsymbol{x}_i) > +\beta_0$$

where $< \cdot, \cdot >$ denotes an inner product.

Formally, suppose we use a mapping H to take the data from $\mathbb{R}^p$ to a (possibly infinite-dimensional) Euclidean space $\mathcal{H}$.

Recall that the optimization problem can be written so that it only depends upon dot products of the data, $\boldsymbol{x}_i'\boldsymbol{x}_j$. Thus after the mapping, the solution in $\mathcal{H}$ depends only upon functions of the form $\boldsymbol{h}(\boldsymbol{x}_i)'\boldsymbol{h}(\boldsymbol{x}_j)$.

Therefore it is sufficient to find a function

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{h}(\boldsymbol{x}_i)'\boldsymbol{h}(\boldsymbol{x}_j).$$

The nice thing is that we do not need to know the mapping H.

Determining the functions in $\boldsymbol{h}(\boldsymbol{x})$ turns out to be equivalent to selecting a basis for a particular subspace. This depends upon a kernel function.

A kernel function is a positive semidefinite function

$$\begin{aligned} K(\boldsymbol{x}, \boldsymbol{x}^*) &= < \boldsymbol{h}(\boldsymbol{x}), \boldsymbol{h}(\boldsymbol{x}^*) > \\ &= \sum_{j=1}^{q} h_j(\boldsymbol{x}) h_j(\boldsymbol{x}^*). \end{aligned}$$

This is related to reproducing kernel Hilbert spaces.

Three common choices for kernel functions in SVM applications are:

- $K(\boldsymbol{x}, \boldsymbol{x}^*) = \exp(-\|\boldsymbol{x} - \boldsymbol{x}^*\|^2/c)$, known as the radial basis;
- $K(\boldsymbol{x}, \boldsymbol{x}^*) = \tanh(\alpha_1 < \boldsymbol{x}, \boldsymbol{x}^* > +\alpha_2)$, or the neural network basis; and
- $K(\boldsymbol{x}, \boldsymbol{x}^*) = (1+ < \boldsymbol{x}, \boldsymbol{x}^* >)^r$, the rth degree polynomials.

To see how this works, suppose $p = 2$ and use the rth degree polynomial basis with $r = 2$. Then

$$\begin{aligned} K(\boldsymbol{x}, \boldsymbol{x}^*) &= (1+ < \boldsymbol{x}, \boldsymbol{x}^* >)^2 \\ &= (1 + x_1 x_1^* + x_2 x_2^*)^2 \\ &= 1 + 2x_1 x_1^* + 2x_2 x_2^* + (x_1 x_1^*)^2 + (x_2 x_2^*)^2 + \\ &\quad 2x_1 x_2 x_1^* x_2^*. \end{aligned}$$

Thus $q = 6$, and with a little algebra one can show that

$$\begin{aligned} h_1(\boldsymbol{x}) &= 1 & h_2(\boldsymbol{x}) &= \sqrt{2}x_1 \\ h_3(\boldsymbol{x}) &= \sqrt{2}x_2 & h_4(\boldsymbol{x}) &= x_1^2 \\ h_5(\boldsymbol{x}) &= x_2^2 & h_6(\boldsymbol{x}) &= \sqrt{2}x_1 x_2. \end{aligned}$$

So the SVM is looking for quadratic discriminant rules, and the programming problem finds the best quadratic surface (in terms of maximizing the margin in $\mathbb{R}^6$) that separates the classes.

As another example, consider the radial basis kernel

$$K(\boldsymbol{x}, \boldsymbol{x}^*) = \exp(-\|\boldsymbol{x} - \boldsymbol{x}^*\|^2/c).$$

In this case $q = \infty$; $\mathcal{H}$ is an infinite-dimensional Hilbert space.

The Nadaraya-Watson estimate for regression with a Gaussian kernel can be represented as a linear combination of radial basis functions:

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{n} y_i h_i(\boldsymbol{x}).$$

Here

$$h_i(\boldsymbol{x}) = a \exp[-b(\boldsymbol{x} - \boldsymbol{x}_i)'(\boldsymbol{x} - \boldsymbol{x}_i)]$$

where $a = \sum_{i=1}^{n} h_i(\boldsymbol{x})$ is the normalization constant and b is the bandwidth.

For SVMs, the separating surface is $g(\boldsymbol{x}) = \boldsymbol{\beta}^T \boldsymbol{h}(\boldsymbol{x}) + \beta_0$.

In the context of classification, the corresponding optimization problem that must be solved is to find $\boldsymbol{\beta}$ and β_0 is

$$\min_{\boldsymbol{\beta},\beta_0} \sum_{i=1}^{n} [1 - y_i g(\boldsymbol{x}_i)]_+ + \lambda \|\boldsymbol{\beta}\|^2$$

where $[\,\cdot\,]_+$ denotes the positive part of the argument.

This has the form of minimizing a loss function plus a penalty. The penalty is on the length of $\boldsymbol{\beta}$, and the loss is a weighted sum of the misclassifications.

SVMs do not automatically avoid the Curse of Dimensionality. For example, as p gets large, the 2nd degree polynomial basis becomes very much like multiple polynomial regression, and we already know that has problems.

7.2 Trees and Random Forests

Recursive partitioning is used for classification as well as regression. CART (Breiman, Friedman, Olshen, and Stone, 1984, *Classification and Regression Trees*, Wadsworth) is the most famous of these methods.

A classification tree starts with a training sample of n cases with known categories. Case i has a vector of covariates $\boldsymbol{x}_i$, and those are used to build a tree-structured classification rule.

Recursive partitioning is one of the most popular data mining tools, in large part because the tree-structured decision rule is easy to represent and often easy to interpret.

Formally, recursive partitioning splits the training sample into increasingly homogeneous groups, thus inducing a partition on the space of explanatory variables.

At each step, the algorithm considers three possible kinds of splits using the vector of explanatory values $\boldsymbol{x}$:

1. Is $x_i \leq t$? (univariate split)
2. Is $\sum_{i=1}^{p} w_i x_i \leq t$? (linear combination split)
3. Does $x_i \epsilon S$? (categorical split, used if x_i is a categorical variable).

The algorithm searches over all possible values of t, all coefficients $\{w_i\}$, and all possible subsets S of the category values to find the split that best separates the cases in the training sample into two groups with maximum increase in overall homogeneity.

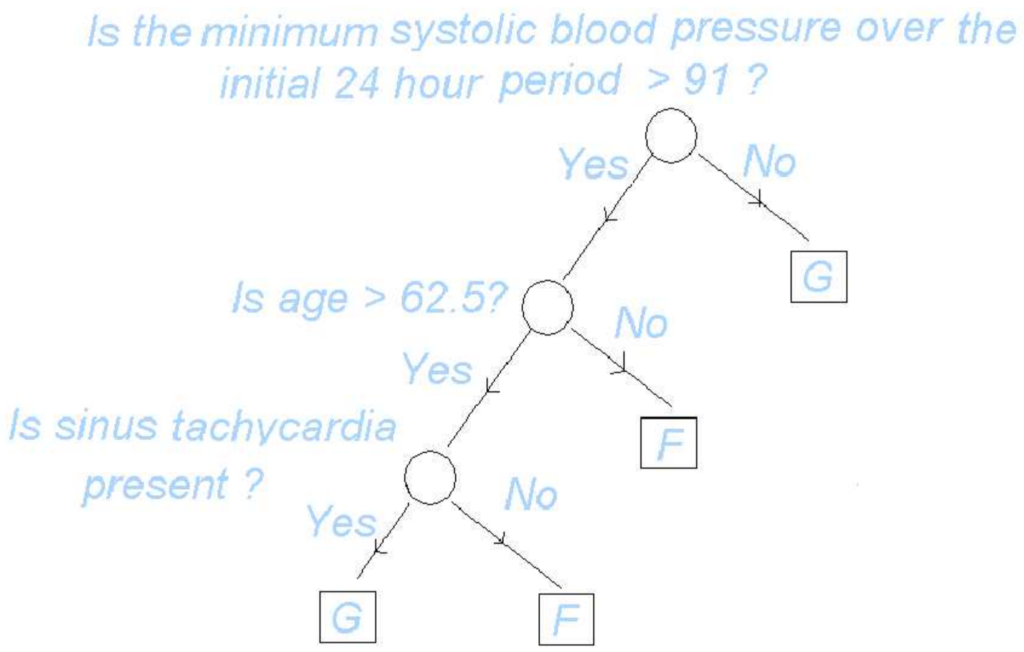
intentional blank

Different partitioning algorithms use different methods for assessing improvement in homogeneity. Classical methods seek to minimize Gini's index of diversity, or use a "twoing rule." Hybrid methods can switch criteria as they move down the decision tree.

Similarly, some methods seek to find the greatest improvement on both sides of the split, whereas other methods choose the split that achieves maximum homogeneity on one side or the other.

Some methods grow elaborate trees, and then prune back to improve predictive accuracy outside the training sample (this is a partial response to the kinds of overfit concerns that arise from the Curse of Dimensionality).

The result is a decision tree. The following figure shows a famous application that assesses whether or not an emergency room patient is at risk for a second heart attack.



Is the minimum systolic blood pressure over the initial 24 hour period > 91 ?
Yes
No
G
Is age > 62.5?
Yes
No
F
Is sinus tachycardia present ?
Yes
No
G
F

A patient whose minimum systolic blood pressure is less than 91 is classified as high risk; otherwise, the tree asks whether the patient is older than 62.5. If not, then the patient is low risk; if so, then the tree asks about sinus tachycardia. If there is no tachycardia, the patient is low risk.

intentional blank

intentional blank

intentional blank

A random forest is a collection of identically distributed trees. Each tree is constructed by some tree classification algorithm such as CART.

The random forest is formed by taking bootstrap samples from the training set. For each bootstrap sample, a classification tree is formed, and there is no pruning—the tree grows until all terminal nodes are pure.

After the tree is grown, one drops a new case down each of the trees. The classification that receives the majority vote is the one that is assigned.

Random forests were invented by Breiman (2001; *Machine Learning*, **45**, 5-32).

The key features of random forests include:

- It is a good classifier—fully competitive with SVMs.
- It generates an internal unbiased estimate of the generalization error.
- It handles missing data very well, and can maintain high levels of accuracy when up to 80% of the data are missing at random.
- It provides estimates of the relative importance of each of the covariates in the classification rule.
- It computes proximities between pairs of cases that can be used in clustering, identifying outliers, and multidimensional scaling.
- It can be applied with unlabelled data, leading to unsupervised clustering.
- It can rebalance the weights when the category proportions in the training data do not reflect the category proportions in the true population.

Random forests are an extension of the idea of bagging trees. But bagging is a very general idea.

Bagging refers to *bootstrap aggregation*. The strategy refers to a way of improving the accuracy of a model. Suppose one has a training sample, fits a model $\hat{f}(\boldsymbol{x})$, and now wants to predict a the response for a new explanatory vector $\boldsymbol{x}^*$.

A bagged estimate for $\boldsymbol{x}^*$ is found by drawing B bootstrap samples from the training data and using these to fit the models $\hat{f}_1(\boldsymbol{x}), \ldots, \hat{f}_B(\boldsymbol{x})$. Then the bagged prediction is:

$$\hat{f}_{\mathbf{bag}}(\boldsymbol{x}^*) = \frac{1}{B}\sum_{i=1}^{B} \hat{f}_i(\boldsymbol{x}^*).$$

intentional blank

The bagging estimate is a simple average of models. Stacking builds on this idea by peforming a weighted average, where the weights take account of the complexity of the model that is fit.

Stacking can be viewed as a version of Bayesian model averaging, where the estimated weights correspond to Bayesian priors that downweight complex models.

Suppose there are K possible models, of different complexities. The stacking prediction at a point $\boldsymbol{x}$ has the form

$$\hat{f}_{\mathbf{stack}}(\boldsymbol{x}) = \sum_{k=1}^{K} \hat{w}_k \hat{f}_k(\boldsymbol{x})$$

where each $\hat{f}_k$ corresponds to one of the K models, as fitted from the training data, and the weights are estimated by stacking.

Let $f_k^{(-i)}(\boldsymbol{x})$ be the prediction at $\boldsymbol{x}$ using model k, as estimated from training data with the ith observation removed. Then the estimated weight vector $\hat{\boldsymbol{w}}$ solves

$$\hat{\boldsymbol{w}} = \mathbf{argmin}_{\boldsymbol{w}} \sum_{i=1}^{n} \left[y_i - \sum_{k=1}^{K} w_k \hat{f}_k^{(-i)}(\boldsymbol{x}_i) \right]^2 .$$

This puts low weight on models that have poor leave-one-out accuracy in the training sample (but beware of twin problem).

Note that one can use other methods than multiple linear regression to combine the models when finding the weights above. Also, one might use different weights for different values of $\boldsymbol{x}$.

7.3 Boosting

Boosting is a method invented by computer scientists to improve weak classification rules.

If one has a classification procedure that does only slightly better than chance at predicting the true categories, then one can apply the procedure to just the portions of the training sample that are misclassified to produce new rules, and then weight all the rules together to achieve better predictive accuracy. Essentially, each rule has a weighted vote on the final classification of a case.

The procedure was proposed by Schapire (1990; *Machine Learning*, **5**, 197-227) and subsequently improved under the name AdaBoost. There have been many refinements since.

intentional blank

The core algorithm for binary classification assumes one has a weak rule $g_1(\boldsymbol{x})$ that takes values in the set $\{1, -1\}$ according to the category.

AdaBoost starts by putting equal weight $w_i = n^{-1}$ on each of the n cases in the training sample. Next, the algorithm repeats the following steps K times:

1. Apply the procedure g_k to the training sample with weights $w_1, \ldots, w_n$.
2. Find the empirical probability p_w of misclassification under these weightings.
3. Calculate $c_k = \ln \frac{(1-p_w)}{p_w}$.
4. If case i is misclassified, replace w_i by $w_i \exp c_k$. Then renormalize so that $\sum_{i=1}^{n} w_i = 1$ and go to step 1.

The final inference is the sign of $\sum_{k=1}^{K} c_k g_k(\boldsymbol{X})$, which is a weighted sum of the determinations made by each of the K rules formed from the original rule g_1.

This boosting rule has several impressive properties:

- It provably improves classification.
- It is resistant to overfit, which arises when K is large.
- The procedure allows quick computation, and thus can be made practical even for very large datasets.
- It can be generalized to handle more than two categories.

Boosting provides an automatic and effective way to increase the capability of almost any classification technique.

As a new method, it is the object of active research; Friedman, Hastie, and Tibshirani (2000; *Annals of Statistics*, **28**, 337-407) link it to formal statistical models such as logistic regression and generalized additive models.

To illustrate the power of boosting, the following graph was constructed by Hastie, Tibshirani, and Friedman (2001; *The Elements of Statistical Learning*, Springer-Verlag).

The graph shows the peformance of a "stump" (i.e., a two-node classfication tree), a 400-node classification tree, and a boosted version of the stump.

The training data are 2000 cases in $\boldsymbol{X}_i \in \mathbb{R}^{10}$; each component is drawn independently from a standard normal distribution. The labels are deterministically assigned, according to:

$$Y = \begin{cases} 1 & \textbf{if } \sum X_i^2 > 9.34 \\ -1 & \textbf{if } \sum X_i^2 \leq 9.34. \end{cases}$$

Note that the χ^2_{10} distribution has median 9.34. The test data consist of 10,000 such values.

intentional blank

intentional blank

To analyze boosting, the first thing to note is that it is fitting an additive model. The form of the classifier is

$$g(\boldsymbol{x}) = \textbf{signum}\{\sum_{k=1}^{K} c_k g_k(\boldsymbol{x})\}$$

and apart from the signum function, this is like the additive models in section 4.

The second thing to note is that boosting chooses the next term in the model in order to maximize the improvement in fitting the training sample. This is called forward stagewise modeling.

From this perspective, boosting constitutes an ensemble method (others include Bayesian model averaging and Random Forests).

Forward stagewise modeling requires a loss function. Boosting uses the loss function

$$L[y, f(\boldsymbol{x})] = \exp[-yf(\boldsymbol{x})]$$

which puts a small penalty on correct classification but a larger one on mistakes.

The forward stagewise additive modeling algorithm works as follows:

1. Initialize. Start with the weak classifier G_1.
2. Cycle. For $k = 2, \dots, K$:
   i. Solve
   $$(g_k, c_k) = \mathbf{argmin}_{c,g} \sum_{i=1}^{n} \exp\{-y_i[G_{k-1}(\boldsymbol{x}_i) + cg(\boldsymbol{x}_i)]\}$$
   ii. Set $G_k(\boldsymbol{x}) = G_{k-1}(\boldsymbol{x}) + c_k g_k(\boldsymbol{x})$.

To sovle the minimization problem in step 2(i), note that:

$$\sum_{i=1}^{n} \exp\{-y_i[G_{k-1}(\boldsymbol{x}_i) + cg(\boldsymbol{x}_i)]\} = \sum_{i=1}^{n} w_{ik} \exp[-y_i cg(\boldsymbol{x}_i)]$$

where $w_{ik} = \exp[-y_i G_{k-1}(\boldsymbol{x})]$.

For any $c > 0$, the solution for g_k is

$$g_k = \mathbf{argmin}_g \sum_{i=1}^{n} w_{ik} I[y_i \neq g(\boldsymbol{x}_i].$$

This is the classifier that minimizes the weighted error rate for predicting y; to see this, note that:

$$\sum_{i=1}^{n} w_{ik} \exp[-y_i cg(\boldsymbol{x}_i)] = e^{-c} \sum_{y_i = g(\boldsymbol{x}_i)} w_{ik} + e^{c} \sum_{y_i \neq g(\boldsymbol{x}_i)} w_{ik}.$$

This solution for g_k did not depend on c, so we can plug it back into the original argmin problem and solve to find:

$$c_k = \frac{1}{2} \ln \frac{1 - \mathbf{err}_k}{\mathbf{err}_k}$$

for err_k the weighted error given by

$$\mathbf{err}_k = \frac{\sum_{i=1}^n w_{ik} I[y_i \neq g_k(\boldsymbol{x}_i)]}{\sum_{i=1}^n w_{ik}}.$$

This stage is the updated so that

$$G_k(\boldsymbol{x}) = G_{k-1}(\boldsymbol{x}) + c_k g_k(\boldsymbol{x})$$

and thus the weights in the next stage are

$$w_{i,k+1} = w_i \exp[-c_k y_i g_k(\boldsymbol{x}_i)].$$

A bit more manipulation shows that these reweights are those used in the boosting algorithm.

One advantage of the exponential loss function used in boosting is that it leads to simple reweighting.

Friedman, Hastie, and Tibshirani (2000) show that the exponential loss function has population maximizer:

$$\begin{aligned} f^*(\boldsymbol{x} &= \mathbf{argmin}_{f(\boldsymbol{x})} \mathbb{E}[\exp[(-Y f(\boldsymbol{x})] \\ &= \frac{1}{2} \frac{\mathbb{P}[Y = 1 \mid \boldsymbol{x}]}{\mathbb{P}[Y = -1 \mid \boldsymbol{x}]} \end{aligned}$$

which imples that boosting is estimating half the log-odds of $\mathbb{P}[Y = 1 \mid \boldsymbol{x}]$. This seems like a reasonable criterion.

Boosting was not invented with exponential loss in mind. The connection was discovered later.

8. Cluster Analysis

In data mining, cluster analysis is often called **structure discovery**. Traditional data methods attempt to cluster all of the cases, but in data mining is is often sufficient to just find some of the clusters.

Recent applications include:

- market segmentation (e.g., Claritas, Inc.)
- syndromic surveillance
- text retrieval
- microarray data

Classical statistics invented three kinds of cluster analysis, many of which were independently reinvented by computer scientists. These three families are:

- hierarchical agglomerative clustering, which developed in the biological sciences and is the most widely used strategy;
- k-means clustering, which is an algorithmic technique invented by MacQueen (1967; *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, University of California Press, 281-297);
- mixture models, which were recently proposed by Banfield and Raftery (1993; *Biometrics*, **49**, 803-821) and are gaining wide currency.

Of these three methods mixture models make the strongest assumptions about the data being a random sample from a population, but this is also the method which best supports inference.

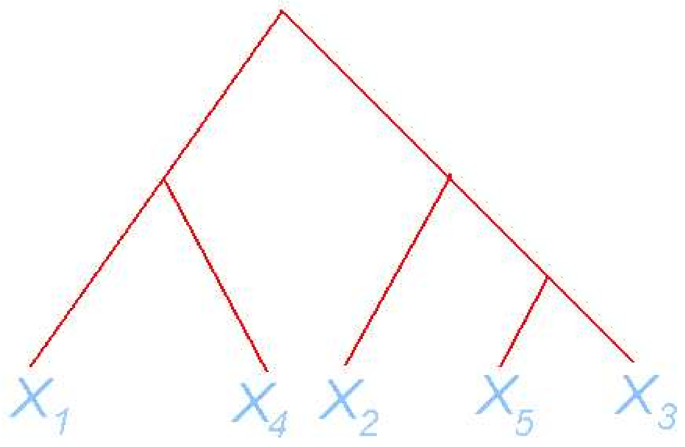
8.1 Hierarchical Clustering

Hierarchical agglomerative clustering joins cases together according to a fixed set of rules.

0. One starts with a sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ of cases and a metric d on all possible sets of cases (including the singleton sets).
1. At the first step, one joins the two cases $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ that have the minimum distances among all possible pairs of cases.
2. At the second step, one joins either two singleton cases or a case $\boldsymbol{x}_k$ to $\{\boldsymbol{x}_i, \boldsymbol{x}_j\}$, according to whichever situation achieves minimum distance.
3. At subsequent steps one may be joining either pairs of cases, pairs of sets of cases, or a case to a set.

Sibson and Jardine (1971; *Mathematical Taxonomy*, Wiley) show that this approach uniquely satisfies certain theoretically desirable properties.

The result of a hierarchical agglomerative cluster analysis is often displayed as a tree.



The lengths of the edges in the binary tree shows the order in which cases or sets were joined together and the magnitude of the distance between them.

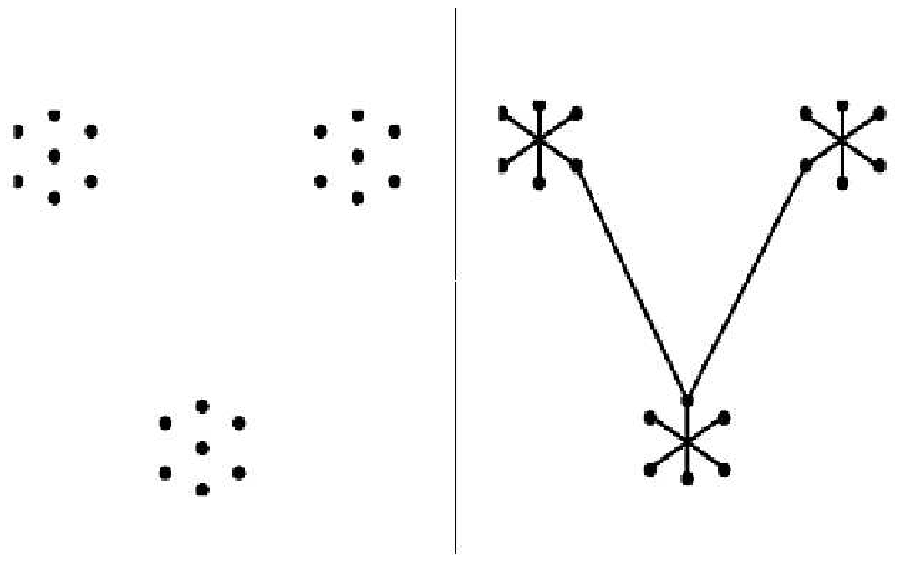
But it is hard to know when to stop growing a tree and declare the clusters. Milligan and Cooper (1985; *Psychometrika*, **50**, 159-179) like the cubic clustering criterion.

Some of the classic metrics for linking sets of cases are:

- Nearest-neighbor or single linkage. Here one has a metric d^* on $\mathbb{R}^p$ and defines
$$d(\mathcal{A}, \mathcal{B}) = \min_{a \in \mathcal{A}, b \in \mathcal{B}} d^*(a, b).$$
- Complete linkage. Here
$$d(\mathcal{A}, \mathcal{B}) = \max_{a \in \mathcal{A}, b \in \mathcal{B}} d^*(a, b).$$
- Centroid linkage. Here
$$d(\mathcal{A}, \mathcal{B}) = d^*(\bar{a}, \bar{b})$$
 where $\bar{a}$ is the centroid (or average) of $\mathcal{A}$ and $\bar{b}$ is the centroid of $\mathcal{B}$.
- Many others have been used. Ward's metric looks at the ratio of between-set and within-set sums of squares, others use various kinds of weighting, etc.

Fisher and Van Ness (1971; *Biometrika*, **58**, 91-104) compare the properties of these metrics in a series of papers.

When the sample size is very large, and or p is large, then the fastest way to cluster cases uses single linkage and creates a minimal spanning tree. Then one just removes the longest edges to create clusters, as shown below.



There is an $\mathcal{O}(n^2)$ algorithm for this that was developed by Prim (1957; *Bell System Technical Journal*, **36**, 1389-1401). It can even be accelerated a little bit.

When p is large and most of the measurements are noise, then it is very difficult to discover true cluster structure. The signal that distinguishes the clusters gets lost in the chance variation among the spurious variables.

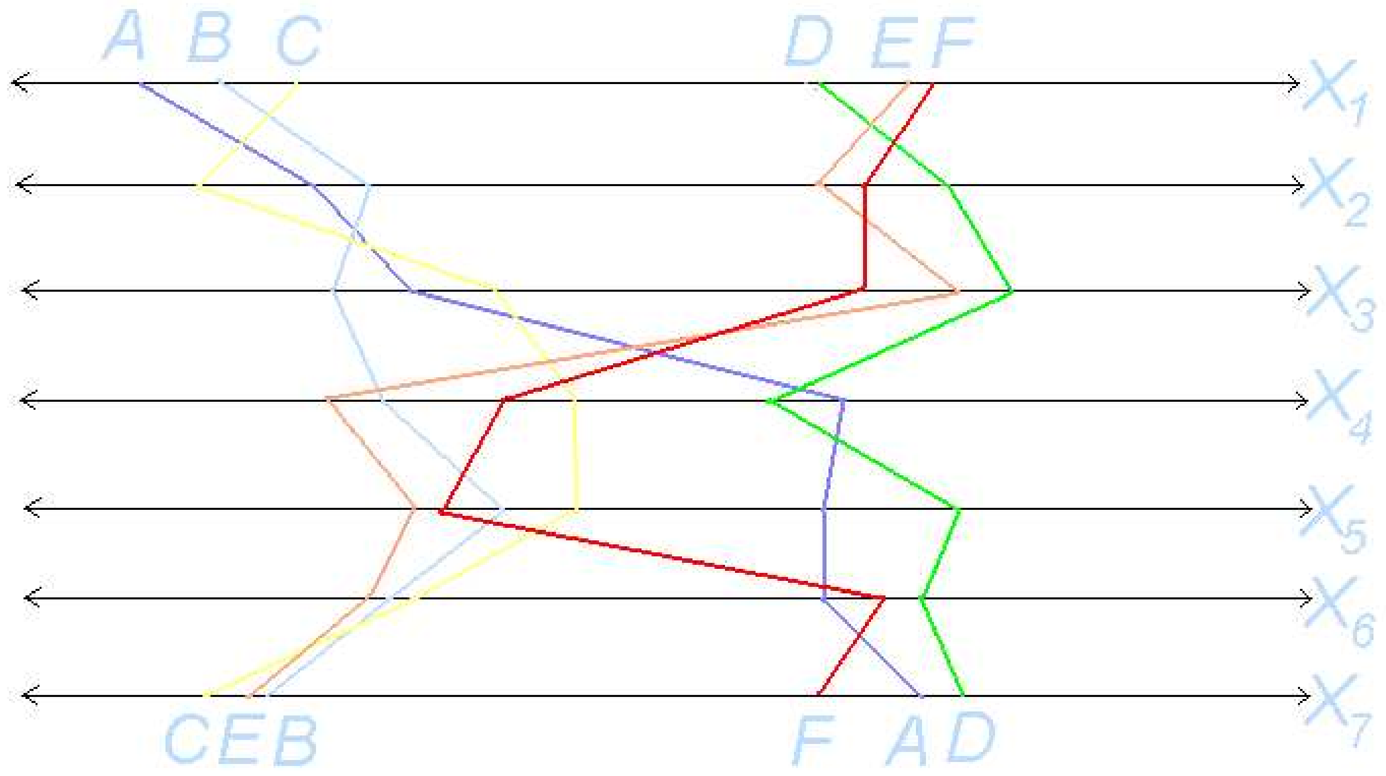
Graphical methods such as G-Gobi take too long to find interesting projections (via a Hermite polynomial function that measures "interestingness" according to the "gappiness" of the projection). See Swayne, Cook, and Buja (1998; *Journal of Computational and Graphical Statistics*, **7**, 113-130).

Visualization techniques probably become infeasible for $p > 10$ or so.

Another thing that can happen with large p is that there are multiple cluster structures. Here cases show strong clustering with respect to one subset of variables, and comparably strong, but distinct, clusters with respect to a different subset of the variables.

This situation gives rise to product structure in the clustering, which quickly becomes unmanageable.

Friedman and Meulman (2004, *Journal of the Royal Statistical Society, Series B*, to appear) discuss this problem and related issues. One strategy for visualizing multiple cluster structure is to use parallel coordinate plots, invented by Inselberg (1985; *The Visual Computer*, **1**, 69-91).



This shows that cases A, B, C and D, E, F have distinct cluster structure with respect to variables x_1, x_2, and x_3, while cases C, E, B and F, A, D Cluster on variables x_6 and x_7. There is no cluster structure on x_4 and x_5.

8.2 k-Means Clustering

In k-means clustering the analyst picks the number of clusters k and makes initial guesses about the cluster centers.

The algorithm starts at those k centers and absorbs nearby cases. Then it calculates a new cluster center (usually the mean) based upon the absorbed cases, and often a covariance matrix, and then absorbs more cases that are nearby, usually in terms of Mahalanobis distance, to the current center:

$$d(\boldsymbol{x}_i, \bar{\boldsymbol{x}}_j) = [(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_j)'\boldsymbol{S}^{-1}(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_j)]^{1/2}$$

where $\boldsymbol{S}$ is the within-cluster covariance matrix and $\bar{\boldsymbol{x}}_j$ is the center of the current cluster j.

The process continues until all of the cases are assigned to one of the k clusters.

Smart computer scientists can do k-means clustering (or approximately this) very quickly. Andrew Moore presented methods at the NSA conference on streaming data in December 2002.

As in hierarchical agglomerative clustering, it is hard to know k. But one can do univariate search—try many values of k, and pick the one at the knee of some lack-of-fit curve, e.g., the ratio of the average within-cluster to between-cluster sum of squares.

The cluster centers move over time. If they move too much, this suggests that the clusters are unstable.

No unique solution is guaranteed for k-means clustering. In particular, if two starting points fall within the same true cluster, then it can be hard to discover new clusters.

8.2.1 Self-Organizing Maps

Kohonen (1989; *Self-Organization and Associative Memory*, Springer-Verlag) developed a procedure called Self-Organizing Maps or SOMs. They quickly become popular for visualizing complex data.

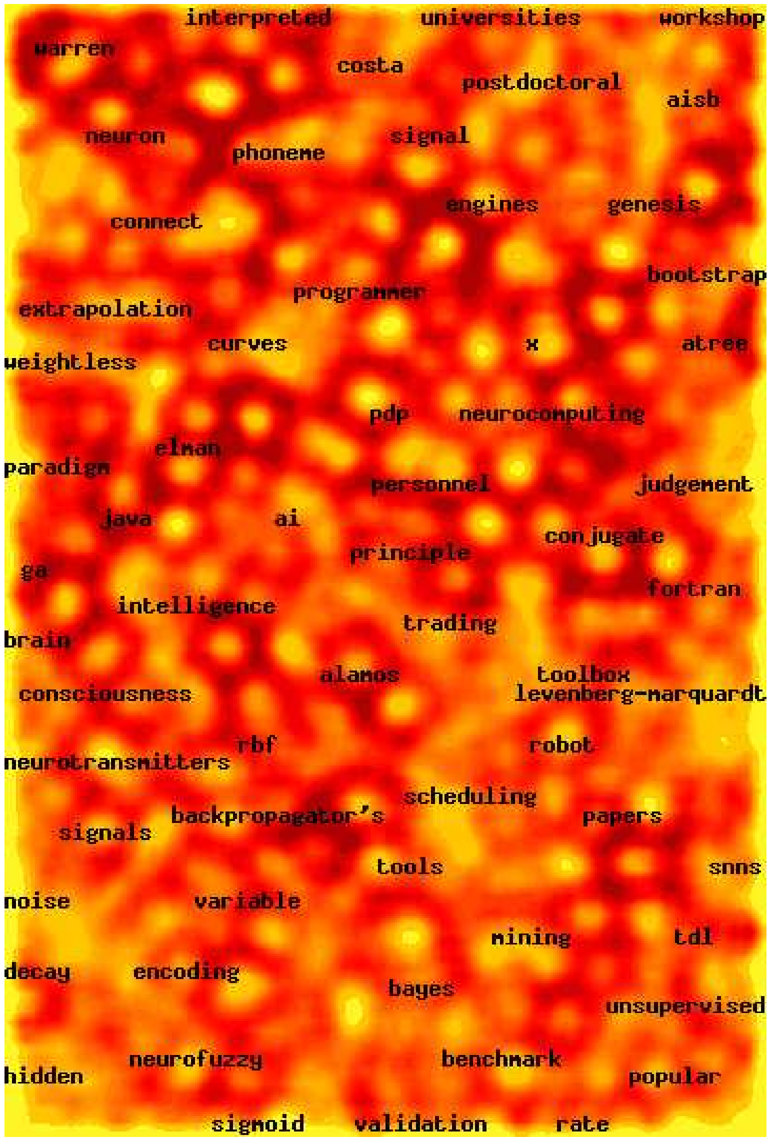
SOMs have become widely used in some aspects of business and information technology, but their underlying theory may be weak. There is no real way to specify uncertainty, and the method is highly susceptible to outliers.

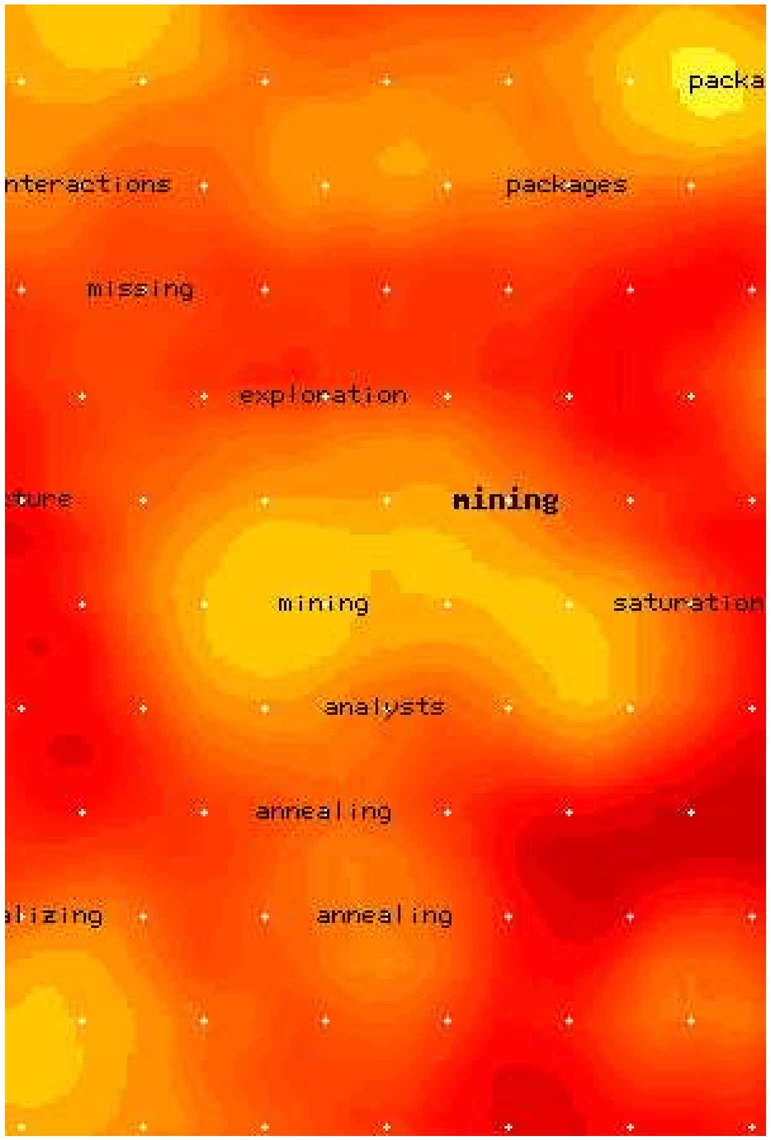
It turns out that SOMs can be viewed as k-means clustering with the constraint that the cluster centers have to lie in a plane.

SOMs are rather like multidimensional scaling. The intention is to produce a two-dimensional (sometimes three-dimensional) picture of high-dimensional data, one that puts similar observations close together in the visualization.

One starts with a set of prototypes, which are just the integer coordinates in the plane formed by the first two principal components of the data. This plane is then distorted, so as to pull the prototypes near to the data. Finally, the data are identified with their nearest point on the distorted surface.

The following two examples of SOMs are heatmaps showing the number of documents in the newsgroup corpus comp.ai.neural-nets with different kinds of subject matter, taken from the WEBSOM homepage. The first is the entire corpus (with well-populated nodes tagged with their keyword), and the second is a blow-up of the heatmap focused on the region around “mining”.


interpreted
universities
workshop
warren
costa
postdoctoral
aisb
neuron
phoneme
signal
engines
genesis
connect
bootstrap
programmer
extrapolation
curves
x
atree
weightless
pdp
neurocomputing
elman
paradigm
personnel
judgement
java
ai
principle
conjugate
ga
fortran
intelligence
trading
brain
alamos
toolbox
levenberg-marquardt
consciousness
rbf
robot
neurotransmitters
scheduling
backpropagator's
papers
signals
tools
snns
noise
variable
mining
tdl
decay
encoding
bayes
unsupervised
hidden
neurofuzzy
benchmark
popular
sigmoid
validation
rate


packa
nteractions
packages
missing
exploration
ture
mining
mining
saturation
analysts
annealing
alizing
annealing

8.3 Mixture Models

Mixture models fit data by a weighted sum of pdfs:

$$f(\boldsymbol{x}) = \sum_{j=1}^{k} \pi_j g_j(\boldsymbol{x} \,|\, \boldsymbol{\theta}_j)$$

where the π_j are positive and sum to one and the g_j are density functions in a family indexed by $\boldsymbol{\theta}$.

The main technical difficulty with mixture models is identifiability. Both

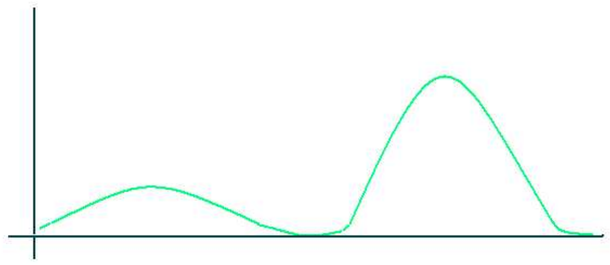
$$\begin{aligned} \textbf{Model 1} &: \quad \{\pi_1 = .5, g_1 = N(0,1);\ \pi_2 = .5, g_2 = N(0,1)\} \\ \textbf{Model 2} &: \quad \{\pi_1 = 1, g_1 = N(0,1);\ \pi_2 = 0, g_2 = N(-10,7)\} \end{aligned}$$

describe exactly the same situation. But identifiability is only an issue at the "edges" of the model space—in most regions it is not a problem.

Bruce Lindsay (1995; *Mixture Models: Geometry, Theory, and Applications*, IMS) gives a convexity argument for solving the identifiability problem. But this has not yet worked its way into data mining practice.

Traditional mixture modeling ducks the identifiability question and uses the EM algorithm (cf. Dempster, Laird, and Rubin; 1977, *Journal of the Royal Statistical Society, Series B*, **39**, 1-22) for model fitting.

To illustrate this, consider how to fit a two-component Gaussian model to the following smoothed histogram.



The model for an observation is

$$f(x) = \pi\phi(x \,|\, \mu_1, \sigma_1^2) + (1-\pi)\phi_2(x \,|\, \mu_2, \sigma_2^2)$$

where ϕ_j is the normal density with parameters $\boldsymbol{\theta}_j = (\mu_j, \sigma_j^2)$.

Let $\boldsymbol{\theta} = (\pi, \mu_1, \mu_2, \sigma_1^2, \sigma_2^2)'$. A direct effort to maximize the log-likelihood leads to

$$\ell(\boldsymbol{\theta}, \boldsymbol{x}) = \sum_{i=1}^{n} \ln[\pi\phi_1(x_i \,|\, \mu_1, \sigma_1^2) + (1-\pi)\phi_2(x_i \,|\, \mu_2, \sigma_2^2)]$$

which turns out to be difficult to solve.

The EM algorithm alternates between an expectation step and a maximization step in order to converge on a solution for the maximum likelihood estimates in this problem.

Instead, assume that there are latent variables (unobserved variables) Δ_i that determine from which mixture component observation x_i came.

$$\Delta_i = \begin{cases} 0 & \textbf{if } x_i \sim \phi_2 \\ x_2^2 & \textbf{if } x_i \sim \phi_1 \end{cases}$$

Then we can write the following generative model:

$$\begin{aligned} Y_{1i} &= N(\mu_1, \sigma_1^2) \\ Y_{2i} &= N(\mu_2, \sigma_2^2) \\ X_i &= \Delta_i Y_{1i} + (1 - \Delta_i) Y_{2i} \end{aligned}$$

where $\mathbb{P}[\Delta_i = 1] = \pi$.

If we knew the $\{\Delta_i\}$ then we could get the mles for (μ_j, σ_j^2) separately for $j = 1, 2$, and this would be easy. But since we do not, we use the EM algorithm to obtain estimates by iterative alternation.

1. Make initial guesses for $\hat{\pi}$, $\hat{\mu}_1$, $\hat{\mu}_2$, $\hat{\sigma}_1^2$, and $\hat{\sigma}_2^2$. Usually we take $\hat{\pi} = .5$, the $\hat{\mu}_j$ are well-separated sample values, and the variance estimates are both set to the sample variance of the full dataset.

2. Expectation Step. Compute the expected values of the Δ_i terms (these are sometimes called "responsibilities"). The responsibility γ_i is:

$$\gamma_i = \frac{\hat{\pi}\phi_1(x_i \,|\, \hat{\mu}_1, \hat{\sigma}_1^2)}{\hat{\pi}\phi_1(x_i \,|\, \hat{\mu}_1, \hat{\sigma}_1^2) + (1-\hat{\pi})\phi_2(x_i \,|\, \hat{\mu}_2, \hat{\sigma}_2^2)}.$$

   Note that this is an approximation to the posterior probability the x_i comes from component ϕ_1.

3. Maximization Step. Compute the new estimates by weighting the sample:

$$\hat{\mu}_1 = \sum_{i=1}^n \gamma_i x_i \Big/ \sum_{i=1}^n \gamma_i \qquad \hat{\mu}_2 = \sum_{i=1}^n (1-\gamma_i) x_i \Big/ \sum_{i=1}^n (1-\gamma_i)$$

$$\hat{\sigma}_1^2 = \frac{\sum_{i=1}^n \gamma_i (x_i - \hat{\mu}_1)^2}{\sum_{i=1}^n \gamma_i} \qquad \hat{\sigma}_2^2 = \frac{\sum_{i=1}^n (1-\gamma_i)(x_i - \hat{\mu}_2)^2}{\sum_{i=1}^n (1-\gamma_i)}.$$

   Also, the new $\hat{\pi}$ is $n^{-1}\sum_{i=1}^n \gamma_i$.

4. Repeat steps 2 and 3 until convergence.

intentional blank

The EM algorithm climbs hills, so in general it converges to a local (but possibly not global) maximum. In part, it is fast because it increases the criterion function on *both* steps, not just the maximization step.

The EM algorithm does not solve the identifiability problem that can arise in mixture models. But it does find a local mle.

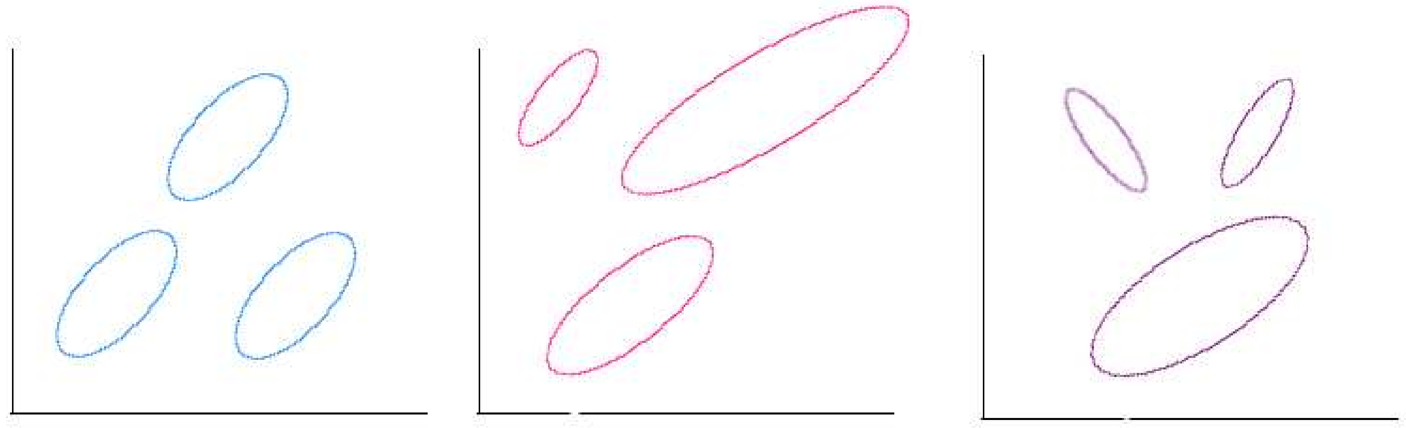
The EM algorithm is considerably more flexible than this example indicates, and it applies in a wide variety of cases (e.g., imputation in missing data problems, inference on causation through counterfactual data). The review given here follows the application in Hastie, Tibshirani, and Friedman (2001; *The Elements of Statistical Learning*, Wiley, chap. 8.5).

8.3.1 Model-Based Cluster Analysis

This is a parametric family of mixture models (usually Gaussian mixtures) that are becoming widely used in both cluster analysis and data mining.

Essentially, this uses a nested sequence of normal mixtures for cluster analysis. The nesting enables one to use likelihood ratio tests to determine which level of modeling is appropriate.

1. At the most nested level, the model assumes that the data come from a k-component mixture of normal distributions, each with a common covariance matrix but different means.
2. At the next-most nested level, the covariance matrices are allowed to differ by an unknown scalar multiple.
3. At the highest level, the covariance matrices for different components are completely unrelated.



The nesting problem does not avoid the problem of identifiability or choice of k. The EM algorithm is used for estimation.

9. Issues with Bases

Nonparametric regression often tries to fit a model of the form

$$f(\boldsymbol{x}) = \sum_{j=1}^{M} \beta_j h_j(\boldsymbol{x}) + \epsilon$$

where the h_j functions may pick out specific components of $\boldsymbol{x}$ (multiple linear regression), or be powers of components of $\boldsymbol{x}$ (polynomial regression), or be prespecified transformations of components of $\boldsymbol{x}$ (nonlinear regression).

Many functions f cannot be represented by the kinds of h_j listed above. But if the set $\{h_j\}$ is an orthogonal basis for a space of functions that contains f, then we can exploit many standard strategies from linear regression.

9.1 Hilbert Spaces

Usually we are concerned with spaces of functions such as $\mathcal{L}^2[a,b]$, the Hilbert space of all real-valued functions defined on the interval $[a,b]$ that are square-integrable:

$$\int_a^b f(x)\,dx < \infty.$$

This definition extends to functions on $\mathbb{R}^p$.

Hilbert spaces have an inner product. For $\mathcal{L}^2[a,b]$ it is

$$< f, g >= \int_a^b f(x)g(x)\,dx.$$

The inner product defines a norm $\|f\|$, given by $< f, g >^{1/2}$, which is essentially a metric on the space of functions.

There are additional issues for a Hilbert space, such as completeness (i.e., the space contains the limit of all Cauchy sequences), but we can ignore those.

A set of functions $\{h_j\}$ in a Hilbert space is mutually orthogonal if for all $j \neq k$, $< h_j, h_k >= 0$.

Additionally, if $\|h_j\| = 1$ for all j, then the set is orthonormal.

If $\{h_j\}$ is an orthonormal basis for a space $\mathcal{H}$ then every function $f \in \mathcal{H}$ can be uniquely written as:

$$f(x) = \sum_{j=1}^{\infty} \beta_j h_j(x)$$

where

$$\beta_j =< f, h_j > .$$

Some famous orthogonal bases include the Fourier bases, consisting of $\{\cos nx, \sin nx\}$, wavelets, Legendre polynomials, and Hermite polynomials.

If one has an orthonormal basis for the space in which f lives, then several nice things happen:

1. Since f is a linear function of the basis elements, then simple regression methods allow us to estimate the coefficients β_j.
2. Since the basis is orthogonal, the estimates of different β_j coefficients are independent.
3. Since the set is a basis, there is no identifiability problem; each function in $\mathcal{H}$ is uniquely expressed as a weighted sum of basis elements.

But not all orthonormal bases are equally good. If one can find a basis that includes f itself, that would be ideal. Second best is a basis in which only a few elements in the representation of f have non-zero coefficients. But this problem is tautological since we do not know f.

One approach to choosing a basis is to use Gram-Schmidt orthogonalization to construct a basis set such that the first few elements correspond to the kinds of functions that statisticians expect to encounter.

This approach is often practical if the right kind of domain knowledge is available. This is often the case in audio signal processing; Fourier series are natural ways to represent vibration.

But in general, one must pick basis set without regard to f. A common criterion is that the influence of the basis elements be local. From this standpoint, polynomials and trigonometric functions are bad, because their support is the whole line, but splines and wavelets are good, because their support is essentially compact.

Given an orthonormal basis, one can try estimating all of the $\{\beta_j\}$. But one quickly runs out of data—even if f is exactly equal to one of the basis elements, noise ensures that all of the other elements will make some contribution to the fit.

To address this problem one can:

- Restrict the set of functions, so it is no longer a basis (e.g., linear regression);
- select only those basis elements that seem to contribute significantly to the fit of the model (e.g., variable selection methods, or greedy fitters like MARS, CART, and boosting);
- regularize, to restrict the values of the coefficients (e.g., through ridge regression or shrinkage or thesholding).

The first technique is problematic, especially since it prevents flexibility in fitting nonlinear functions.

The second technique is important, especially when one has large p but small n. But it is often insufficient, and the theory for aggressive variable selection is not well-developed yet.

9.2 Ridge Regression

Ridge regression is an old idea, used originally to protect against multicollinearity. It shrinks the coefficients in a regression towards zero (and each other) by imposing a penalty on the sum of the squares of the coefficients.

$$\hat{\beta} = \mathbf{argmin}_{\boldsymbol{\beta}} \{ \sum_{I=1}^{n} (y_I - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \}$$

where $\lambda \geq 0$ is a penalty parameter that controls the amount of shrinkage.

Recall Stein's result that when estimating a multivariate normal mean with squared error loss, the sample average is inadmissible and can be improved by shrinking the estimator towards $\mathbf{0}$ (or any other value).

Neural net methods now often do similar shrinkage on the weights at each node, Thereby improving predictive squared accuracy.

intentional blank

Ridge regression methods are not equivariant under scaling, so one normally standardizes the x_{ij} sample values wrt j so that each has unit variance.

Traditional ridge regression solves

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}'\boldsymbol{y}$$

so the name derives from the stabilization of the inverse matrix obtained by adding a constant to the diagonal.

Note that this increases the trace of the hat matrix, which corresponds to the degrees of freedom used in fitting the model.

Ridge regression (and shrinkage methods in general) can be obtained as the Bayesian posterior mode for a suitable prior. In the multivariate normal case, the prior assumes independent normal distributions for each β_j, with common variance.

intentional blank

9.3 The Lasso

The Lasso method is analogous to ridge regression. The Lasso estimate is given by

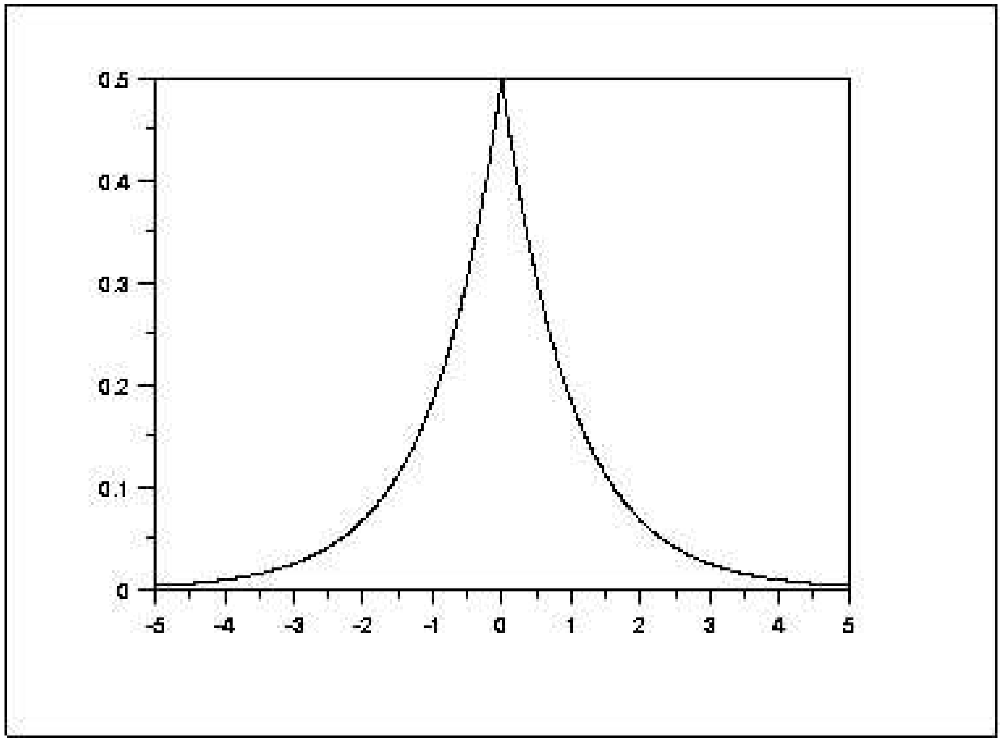
$$\hat{\beta} = \mathbf{argmin}_{\beta}\{\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2\}$$

subject to the constraint that $\sum_{j=1}^{p}|\beta_j| \leq s$.

The Lasso replaces the quadratic penalty in ridge regression by a penalty on the sum of the absolute values of the β_j terms.

If s is larger than the sum of the absolute values of the least squares estimators, then the Lasso agrees with OLS. If s is small, then many of the β_j terms are driven to 0, so it is performing variable selection.

The Lasso corresponds to a Bayesian method in which the prior on each parameter is Laplacian.



Unlike ridge regression, the Lasso is apt to drive coefficients to zero (i.e., perform variable selection) when the penalty is tight. To see this, the left graph shows how the contours for the OLS estimates intersect with the constrained space for the Lasso, versus the right graph which shows the same situation for ridge regression.

This property of the lasso results from the geometry of the set of $\boldsymbol{\beta}$ values that have constant penalty. The procedure has become extremely popular and seems to perform well.

See Tibshirani (1996; *Journal of the Royal Statistical Society, Series B*, **58**, 267-288) for more details and many examples.

intentional blank

intentional blank

intentional blank

intentional blank

9.4 Overcompleteness

Statisticians traditionally have used an orthogonal basis of functions $\{h_j\}$ for estimating functions f:

$$\hat{f}(\boldsymbol{x}) = \sum_{j=1}^{M} \hat{\beta}_j h_j(\boldsymbol{x}).$$

This choice is largely motivated by the independence of the coefficient estimates and the ability to do asymptotic theory.

But computer scientists have found that using larger sets of functions than are available in the orthogonal basis can improve performance. See Wolfe, Godsill, and Ng (2004; *Journal of the Royal Statistical Society, Series B*, **66**, 1-15). This expansion is called overcompleteness and it proving surprisingly successful in both regression and classification contexts.

This larger set is called a frame. A frame contains a basis for the space of interest (e.g., $\mathcal{L}^2[a, b]$), but may also other functions. Formally, a frame is a set of functions $\{h_j : j \in J\}$ with the property that there are constants $A, B > 0$ such that

$$A\|f\|^2 \leq \sum_{j \in J} | < f, h_j > |^2 \leq B\|f\|^2 \quad \forall f \in \mathcal{H}.$$

A frame that is just an ordinary basis has $A = B = 1$. A frame that is the union of two ordinary bases has $A = B = 2$. Some frames have uncountable cardinality.

Nontrivial frames contain at least one non-zero element which can be written as a linear combination of the other elements in the frame. This contrasts with the situation for basis sets.

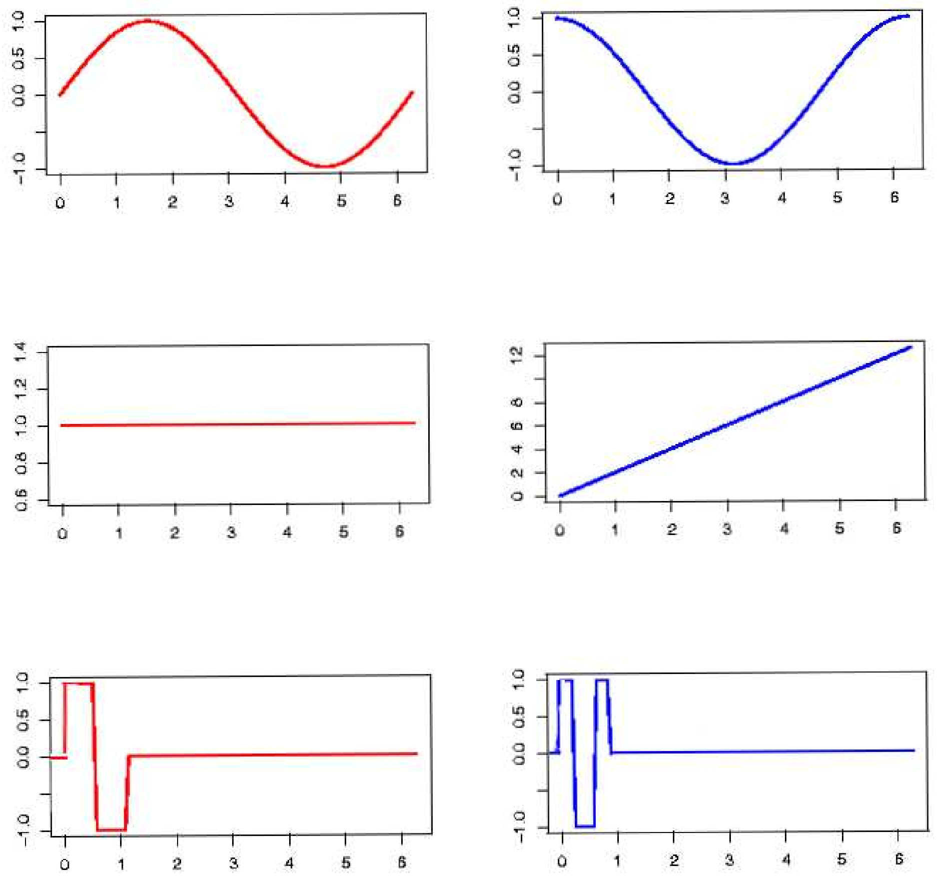
The advantage of frames is that their greater size allows the possibility of finding a *very* parsimonious representation for f.

One wants a criterion for comparing the performance of different frames used for function estimation.

Consider estimating functions in $\mathcal{L}^2[a, b]$. If one forms an overcomplete frame as the union of two basis sets, is it better to take the union of a Fourier basis and a Hermite polynomial basis, or the union of a Fourier basis and the Haar basis?

One imagines that it would be desirable to combine bases that contain very different functions. From this perspective, the Fourier basis, which is smooth, might be more effectively combined with the rough Haar basis than the smooth Hermite polynomial basis. Thus the resulting frame would contain elements that allow parsimonious representation of both smooth and rough functions.

The following figure shows the first two elements of these three bases.



One proposal for a criterion that compares two frames depends upon the parsimony of the approximating functions.

For a given frame $F = \{\, h_j : j \in J \,\}$, a function f, and a tolerance ϵ^*, consider the set of all approximating functions

$$S(f, \epsilon^*) = \{\, \hat{f} : \hat{f} = \sum_{j=1}^{k} \beta_j h_j \textbf{ and } \|f - \hat{f}\| < \epsilon^* \,\}.$$

Each $\hat{f} \in S(f, \epsilon^*)$ has a certain number of terms in the sum. The most parsimonious approximation is the one that uses the fewest terms h_j. Let

$$k(f, \epsilon^*) = \inf\{j : \hat{f} \in S(f, \epsilon^*)\}.$$

Some functions $f \in L^2[0,1]$ will be hard to approximate with elements in the frame F, and for these functions $k(f, \epsilon^*)$ will be large (and maybe infinite).

To handle the worst case for estimation, let

$$k(\epsilon^*) = \sup_{f \in L^2[0,1]} \{ k(f, \epsilon^*) \}.$$

This is the number of terms needed to approximate the most difficult function in $L^2[0,1]$ to within ϵ^* using only frame elements.

For some frames and values ϵ^*, $k(\epsilon^*)$ will be infinite. But there are many cases in which it is finite. For example, if F contains an ϵ^*-net, this trivially ensures that $k(\epsilon^*) = 1$.

For non-trivial cases in which frames F and G are unions of bases, one would like to say that frame F is better than frame G at level ϵ^* if $k_F(\epsilon^*) < k_G(\epsilon^*)$, where the subscript indicates the frame.

If there exists some γ such that the inequality holds for all $\epsilon^* < \gamma$, then one could broadly claim that F is better than G.

10. Wavelets

A wavelet is a function that looks like a localized wiggle. Special collections of wavelets can be used to obtain approximations of functions by separating then information at different scales.

The stunning success of wavelets has spurred explosive growth in research. Donoho and Johnstone (1994; *Biometrika*, **81**, 425-455) led the way in statistics, showing that wavelets can achieve local asymptotic minimaxity in approximating thick classes of functions.

Local asymptotic minimaxity is a technical property, but it ensures that the estimates are asymptotically minimax with respect to a large family of loss functions in a large class of functions. Such estimates probably evade the COD, in the same technical sense that neural nets do.

In its simplest form, a wavelet representation begins with a single function ψ, often called th mother wavelet.

We define a collection of wavelets—called a wavelet basis—by dilation and translation of the mother wavelet ψ:

$$\psi_{jk}(t) = 2^{j/2}\psi(2^j t - k)$$

for $j, k \in \boldsymbol{Z}$, the set of integers.

One obtains an overcomplete frame if one allows j, k to take values in $\mathbb{R}$ rather than $\boldsymbol{Z}$. This is called an *undecimated* wavelet basis.

Each ψ_{jk} has a characteristic resolution scale (determined by j) and is roughly centered on the location $k/2^j$. It turns out that if ψ is properly chosen, any "reasonable" function can be represented by an infinite linear combination of the ψ_{jk} functions.

Wavelets have three key features:

- Wavelets provide *sparse* representations of a broad class of functions and signals,
- Wavelets can achieve very good *localization* in both time and frequency,
- There are *fast algorithms* for computing wavelet representations in practice.

Sparsity means that most of coefficients in the linear combination are nearly zero. A single wavelet basis can provide sparse representations of many spaces of functions at the same time.

Good localization means that both the wavelet function ψ *and* its Fourier transform have small support; i.e., the functions are essentially zero outside some compact domain.

Regarding speed, wavelet representations can be computed in $\mathcal{O}(n \log n)$ and sometimes $\mathcal{O}(n)$ operations (the FFT is $\mathcal{O}(n \log n)$).

10.1 Constructing Wavelets

We begin with a mathematical description of a smooth localized wiggle.

Let $\mathcal{L}^p(\mathbb{R})$ denote the space of measurable complex-valued functions f on the real numbers $\mathbb{R}$ such that

$$\|f\|_p = \left[\int |f(x)|^p\right]^{1/p} dx < \infty.$$

Here $\mathcal{L}^2(\mathbb{R})$ is a Hilbert space with inner product defined by

$$< f, g >= \int f(x)\bar{g}(x)\, dx,$$

where $\bar{g}$ is the complex conjugate of g.

Recall that a complex function $g(x) = u(x) + iv(x)$ has conjugate $\bar{g}(x) = u(x) - iv(x)$ where $u(x)$ and $v(x)$ are real-valued functions and i is $\sqrt{-1}$. The modulus of $g(x)$ is $\sqrt{u^2(x) + v^2(x)}$.

For integers $D, M \geq 0$, suppose that $\psi \in \mathcal{L}^2(\mathbb{R})$ satisfies the following for $d = 0, \ldots, D$ and $m = 0, \ldots, M$:

1. The derivative $\psi^{(d)}$ exists and is in $\mathcal{L}^\infty(\mathbb{R})$,
2. The modulus $|\psi^{(d)}|$ is rapidly decreasing as $|x| \to \infty$, and
3. The mth moment of ψ vanishes, i.e., $\int x^m \, \psi(x) \, dt = 0$.

Then we say that ψ is a basic wavelet of regularity (D, M).

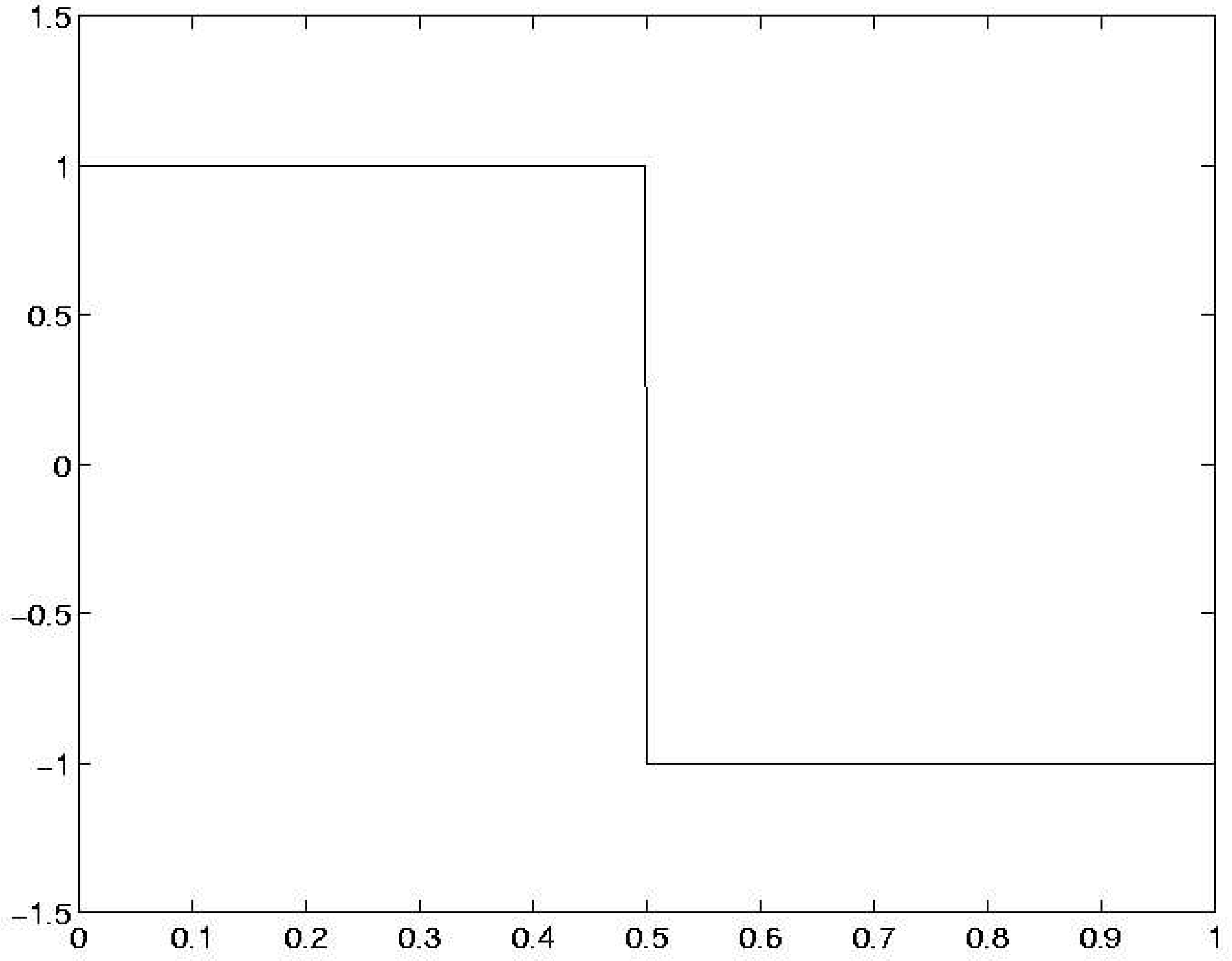
Condition 1 implies that ψ is smooth, and conditions 1 and 2 together imply that ψ is localized in time and frequency.

Condition 2 implies that ψ is highly concentrated in the x domain by requiring that it decrease faster than any inverse polynomial outside some compact set.

Condition 1 implies that the Fourier transform of ψ is quite concentrated as well, decreasing faster than the reciprocal of a D-degree polynomial for high frequency.

Condition 3 forces ψ to be "wiggly" since the integral of ψ with any polynomial up to degree M must be zero.

The standard example of a wavelet basis is the Haar Basis. Let $\psi = 1_{[0,1/2)} - 1_{[1/2,1)}$ where 1_A is the indicator function of the set A. Then the mother wavelet is:



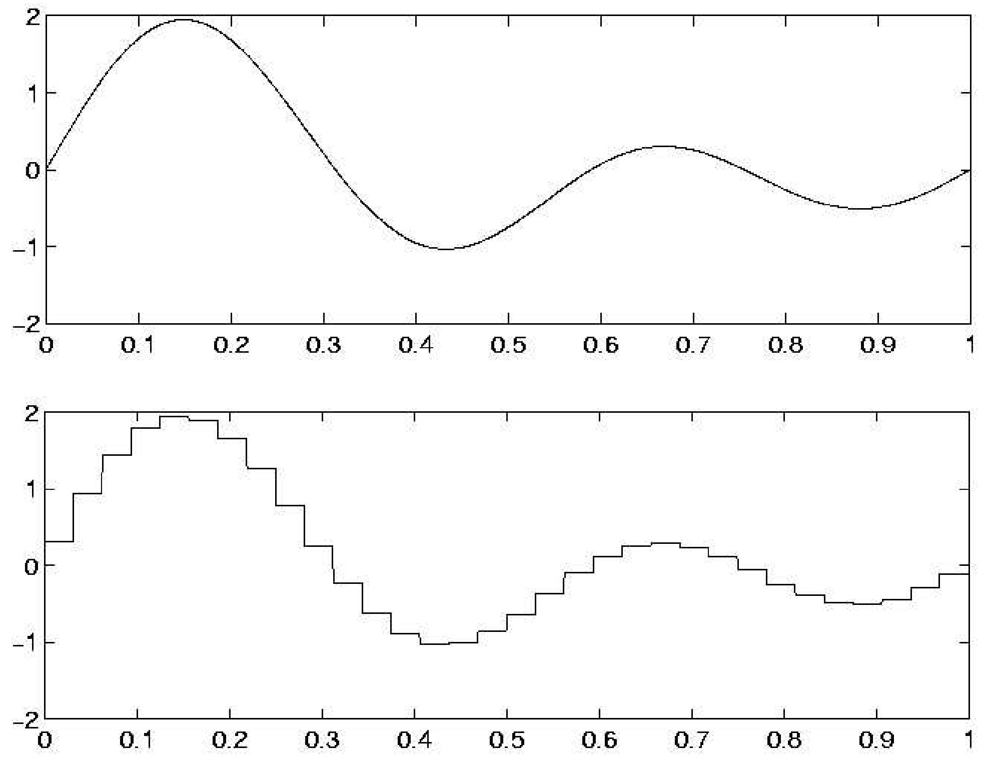
This function is not differentiable, but it has compact support and integrates any constant to zero. Thus ψ is a basic wavelet of regularity $(0, 0)$.

If we define ψ_{jk} from ψ by translation and dilation, we obtain a doubly-infinite set of functions with the same properties as ψ.

Moreover, any two ψ_{jk} are orthonormal in the sense that $< \psi_{j,k}, \psi_{j',k'} >= \delta_{jj'}\delta_{kk'}$ where δ is a Kronecker delta.

These ψ_{jk} form an orthonormal basis for $\mathcal{L}^2(\mathbb{R})$. any function in $\mathcal{L}^2(\mathbb{R})$ can be well-approximated by $\sum_{j,k} \beta_{j,k}\psi_{jk}$. (As a first step in seeing this, note that it is sufficient to be able to approximate any function that is piecewise constant on dyadic intervals.)

Since the Haar basis consists of step functions, an approximation of a smooth function by a finite number of Haar wavelets is necessarily ragged.



The construction of smoother variants of the Haar basis leads to more general wavelets.

A wavelet basis is especially useful if it can extract interpretable information about $f \in \mathcal{L}^2(\mathbb{R})$ through the inner product $\beta =< f, \psi >$.

If ψ is well-localized in $\mathbb{R}$, then a β coefficient essentially reflects the behavior of f on a particular part of its domain (unlike the case in linear regression).

Similarly, if ψ is well-localized in frequency, then β essentially reflects particular frequency components of f as well.

Suppose that ψ is concentrated in a small interval and that in that interval f can be approximated by a Taylor series; then, since the inner product with ψ cancels all polynomials up to degree M, β reflects only the higher-order behavior in f. (Another way to state this is that if two functions differ only by a polynomial of degree M or smaller, their β coefficient will be the same.)

For a basis $\{\psi_{jk}\}$, the index j represents resolution scale; as j increases, ψ_{jk} becomes concentrated in a decreasing set.

The index k represents location; as k changes, the function is shifted along the line. Hence, the map $(j,k) \mapsto \beta_{j,k} =< f, \psi_{jk} >$ provides information about f at every dyadic scale and location.

The wavelet representation

$$f(x) = \sum_{j \in Z} \sum_{k \in Z} \beta_{jk} \psi_{jk}(x)$$

is called the homogeneous equation. It expresses f in terms of functions that all integrate to 0. (There is no paradox here, since convergence in $\mathcal{L}^2(\mathbb{R})$ and convergence in $\mathcal{L}^1(\mathbb{R})$ are not the same thing.)

10.2 The Father Wavelet

It is often useful to re-express the homogeneous equation in an equivalent but inhomogeneous form. This form separates "coarse" structure from "fine" structure.

To do this, select a convenient resolution level J_0. Resolution levels $j \le J_0$ capture the coarse structure of f and levels $j > J_0$ capture the fine structure.

One approximates the fine-resolution structure of f by a linear combination of the ψ_{jk} at $j \geq J_0$, and one approximates the coarse-resolution structure by a combination of the functions $\phi_k(t) = \phi(t-k)$ for $k \in \boldsymbol{Z}$, where ϕ is called the father wavelet(or scaling function).

The inhomogeneous representation for $f \in \mathcal{L}^2(\mathbb{R})$ is then

$$f = \sum_{k \in \boldsymbol{Z}} < f, \phi_k > \phi_k \; + \sum_{j \geq J_0} \sum_{k \in \boldsymbol{Z}} < f, \psi_{jk} > \psi_{jk}.$$

This father wavelet ϕ is closely related to the mother wavelet ψ:

- one can choose ϕ so that $< \phi, \psi >= 0$,
- the father wavelet satisfies the first two requirements of a basic wavelet with the same indices of regularity as ψ.

The difference between the inhomogeneous and homogeneous representations lie in the first sum. This aggregates the information in the low resolution levels through the specially constructed function ϕ rather than distributing it among the ψ_{jk} terms.

The prime advantage of this inhomogeneous representation is that the coarse/fine dichotomy is intuitively appealing and useful. This is especially the case if the noise in the function estimation can be thought of as high-frequency, while the signal in the problem is low-frequency.

For the Haar Basis, the father wavelet is the indicator of the unit interval, $\phi = 1_{[0,1)}$.

When $J_0 = 0$, the inhomogeneous representation takes the form

$$f = \sum_{k \in \boldsymbol{Z}} \alpha_k \, \phi_k \; + \sum_{j \geq 0} \sum_{k \in \boldsymbol{Z}} \beta_{j,k} \, \psi_{jk},$$

where $\alpha_k =< f, \phi_k >$ and $\beta_{j,k} =< f, \psi_{jk} >$.

The coefficients α_k are just integrals of f over intervals of the form $[k, k+1)$; the remaining structure in f is represented as fluctuations within those intervals.

Note that ϕ is in $\mathcal{L}^2(\mathbb{R})$, and consequently can be written in terms of the ψ_{jk}:

$$\phi = \frac{1}{2} \psi_{-1,0} + \frac{1}{2} \sum_{j=2}^{\infty} 2^{-j/2} \psi_{-j,0}.$$

This uses only the coarsest ψ_{jk} terms, as expected.

The relationship between ψ and ϕ goes the other way as well: $\psi = 1_{[0,1/2)} - 1_{[1/2,1)}$ by definition, which is a linear combination of translated and dilated ϕ terms.

10.3 Multiresolution Analysis

A key idea in wavelets is that of successive refinement. If one can approximate at several levels of accuracy, then the differences between successive approximations characterize the refinements needed to move from one level to another.

Multiresolution analysis (MRA) formalizes this notion for approximations that are related by a translation and dilation. An MRA is a sequence of nested approximation spaces for a containing class of functions.

For the containing class $\mathcal{L}^2(\mathbb{R})$, MRA is a nested sequence of closed subspaces

$$\cdots \subset V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots$$

such that

1. $\textbf{clos}\left\{\bigcup_{j\,\in\,\boldsymbol{Z}} V_j\right\} = \mathcal{L}^2(\mathbb{R})$

2. $\bigcap_{j\,\in\,\boldsymbol{Z}} V_j = \{0\}$

3. $f \in V_j \Leftrightarrow f(2^{-j}\cdot) \in V_0 \quad \forall j \in \boldsymbol{Z}$

4. $f \in V_0 \Leftrightarrow f(\cdot - k) \in V_0 \quad \forall k \in \boldsymbol{Z}.$

Condition 1 ensures that the approximation spaces (V_j) are sufficient to approximate any function in $\mathcal{L}^2(\mathbb{R})$.

There are many sequences of spaces satisfying Conditions 1, 2 and 4; the name "multiresolution" is derived from Condition 3 which implies that all the V_j are dyadically scaled versions of a common space V_0. By Condition 4, V_0 is invariant under integer translations.

When generating orthonormal wavelet bases, we also require that the space V_0 of the MRA contains a function ϕ such that the integer translations $\{\phi_{0,n}\}$ form an orthonormal basis for V_0. For simplicity, we will focus almost exclusively on orthonormal bases here.

Since $V_0 \subset V_1$, we can define its orthogonal complement W_0 in V_1, so $V_1 = V_0 \oplus W_0$. We can do likewise for every j, where $V_{j+1} = V_j \oplus W_j$.

The sequence of spaces $\{W_j\}_{j \in \boldsymbol{Z}}$ are mutually orthogonal and they are inherit Condition 3 from the V_j; i.e., $f \in W_j$ if and only if $f(2^{-j} \cdot) \in W_0$.

By themselves, these spaces provide a homogeneous representation of $\mathcal{L}^2(\mathbb{R})$, since $\mathcal{L}^2(\mathbb{R}) = \text{clos}\{\cup_{j \in \boldsymbol{Z}} W_j\}$. The W_j are the building blocks for successive refinement from one approximating space to another.

Given $f \in \mathcal{L}^2(\mathbb{R})$, the best approximation to f in any V_j is given by $P_j f$, where P_j is the orthogonal projection onto V_j. It follows that $Q_j = P_j - P_{j-1}$ is the orthogonal projection onto W_j.

Given a coarse approximation $P_0 f$, one can refine to the finer approximation $P_J f$ for any $J > 0$ by adding details from successive W_j spaces:

$$\begin{aligned} P_J f &= P_0 f + \sum_{j=1}^{J} P_j f - P_{j-1} f \\ &= P_0 f + \sum_{j=0}^{J-1} Q_j f, \end{aligned}$$

and as $J \to \infty$,

$$f = P_0 f + \sum_{j=0}^{\infty} Q_j f.$$

Successive refinement exactly mimics the inhomogeneous wavelet representation. The coarse approximation $P_0 f$ corresponds to a linear combination of the $\phi_{0,k}$, and each $Q_j f$ for $j \geq 0$ corresponds to a linear combination of the span of the ψ_{jk}.

As an example, suppose V_j is the set of piecewise constant functions on intervals of the form $[k2^{-j}, (k+1)2^{-j})$. So V_0 is generated by integer translations of the function $\phi = 1_{[0,1)}$, the father wavelet for the Haar basis.

For $f \in \mathcal{L}^2(\mathbb{R})$, let $\alpha_{j,k} =< \phi_{j,k}, f >$. Then $P_0 f = \sum_k \alpha_{0,k}\phi_{0,k}$ and $P_1 f = \sum_k \alpha_{1,k}\phi_{1,k}$ are approximations to f that are piecewise constant on unit and half-unit intervals, respectively.

How does one refine the coarse approximation $P_0 f$ to the next higher resolution level?

We know $\alpha_{0,k} = \frac{1}{\sqrt{2}}(\alpha_{1,2k} + \alpha_{1,2k+1})$, but we also need to know the difference $\beta_{0,k} = \frac{1}{\sqrt{2}}(\alpha_{1,2k} - \alpha_{1,2k+1})$ between the integrals of f over the half-intervals $[k, k+1/2)$ and $[k+1/2, k+1)$. This $\beta_{0,k}$ is just the coefficient $< \psi_{0,k}, f >$ of the $(0,k)$ Haar wavelet $\psi_{0,k}$. The translations of the Haar mother wavelet form an orthonormal basis for the space W_0.

For a general MRA, how does one find the corresponding ψ?

It turns out that ψ and ϕ determine each other through the refinement relations among the spaces. So one can construct a function ψ whose integer translations (i.e., $\psi_{0,k}(t) = \psi(t-k)$) yield an orthonormal basis of W_0.

By construction, both ψ and ϕ are in V_1, and the $\psi_{0,k}$ and $\phi_{0,n}$ are orthogonal. It follows that both ϕ and ψ can be expressed as a linear combination of the $\phi_{1,n}$ with some constraints on the coefficients.

This reasoning leads to the following two-scale identities:

$$\begin{aligned} \phi(t) &= \sqrt{2}\sum_n g_n\, \phi(2t-n) \\ \psi(t) &= \sqrt{2}\sum_n h_n\, \phi(2t-n). \end{aligned}$$

The $\{h_n\}$ and $\{g_n\}$ satisfy $\sum_n |h_n|^2 = 1$ and $\sum_n |g_n|^2 = 1$. Orthogonality of the $\phi_k(t)$ and $\psi_{0,k}$ and the fact that V_1 is a direct sum of V_0 and W_0 force relationships among $\{g_n\}$ and $\{h_n\}$.

For a specific MRA, these relationships provide enough conditions to uniquely identify mother and father wavelets ϕ and ψ.

In the Haar case, the sequences are

$$\begin{aligned}\{g_n\} &= (\ldots, 0, 1/\sqrt{2}, 1/\sqrt{2}, 0, \ldots)\\ \{h_n\} &= (\ldots, 0, 1/\sqrt{2}, -1/\sqrt{2}, 0, \ldots).\end{aligned}$$

In general, it is more convenient to work with the two-scale identities in the Fourier domain so as to characterize the Fourier transforms ϕ^* and ψ^*.

The value of the two-scale identities is in the connections they impose on the wavelet coefficients. By the two-scale identities,

$$\begin{aligned}\phi_{jk}(t) &= 2^{(j+1)/2} \sum_n g_n \phi(2^{j+1}t - 2k - n)\\ \psi_{jk}(t) &= 2^{(j+1)/2} \sum_n h_n \phi(2^{j+1}t - 2k - n).\end{aligned}$$

It follows that

$$\begin{aligned} < \phi_{jk}, f > &= \sum_n g_n < \phi_{j+1,2k+n}, f > \\ &= \sum_n g_{n-2k} < \phi_{j+1,n}, f > \end{aligned}$$

and

$$\begin{aligned} < \psi_{jk}, f > &= \sum_n h_n < \phi_{j+1,2k+n}, f > \\ &= \sum_n h_{n-2k} < \phi_{j+1,n}, f >. \end{aligned}$$

Each results from convolving the sequence of higher-resolution inner products with the reversed sequences $\{g_n\}$ and $\{h_n\}$ and then retaining only the even numbered components of the convolution.

So given the coefficients $< \phi_{J,k}, f >$ for $k \in \boldsymbol{Z}$ at some fixed resolution level J, one can compute the coefficients at all coarser levels by successively filtering and decimating the sequences. This is the heart of the Discrete Wavelet Transform (DWT).

11. Model Complexity

Our concern is the old bias-variance tradeoff (or overfitting, or capacity control). One gets the best predictive performance if the family of models strikes the right balance between the capacity of the family and the accuracy of the performance on the training set.

Capacity refers to the ability of the model (or machine to perfectly fit the training sample. If the model is so flexible that it can perform without error on the training sample, then it is fitting the noise as well as the signal (i.e., overfitting).

A model with too much capacity is like a biologist with perfect memory, who when shown a dog, cannot identify its species because it has a different number of hairs than any dog previously studied. A family with too little capacity is like a lazy biologist who classifies everything with four legs as a dog.

11.1 VC Classes

Machine learning theorists have developed clever ways to set bounds on the performance capability of data mining procedures.

Suppose one has a training sample $\{(y_i, \boldsymbol{x}_i)\}$ and wants to predict a future Y value from measured $\boldsymbol{X}$ values. To do this, one chooses a family of models $\mathcal{F} = \{f(\boldsymbol{x}, \boldsymbol{\theta})\}$ and uses the training sample to find a value of $\boldsymbol{\theta}$ that gives "good" performance.

This structure holds in both regression or classification. Here $f(\cdot, \boldsymbol{\theta}) : \mathbb{R}^p \rightarrow \mathbb{R}$, and the family of functions (the model $\mathcal{F}$) is indexed by $\boldsymbol{\theta} \in \boldsymbol{\Theta}$.

In multiple linear regression, $\mathcal{F}$ consists of all p-dimensional hyperflats and $\boldsymbol{\theta}$ are the regression coefficients. In two-class linear discriminant analysis, $\mathcal{F}$ consists of all p-dimensional hyperflats and the $\boldsymbol{\theta}$ are the values in the Mahalanobis-distance rule.

The performance of any model in $\mathcal{F}$ is assessed through a loss function $L[f(\boldsymbol{x}, \boldsymbol{\theta}), y]$. This measures the deviation between the true value y and a predicted value $f(\boldsymbol{x}, \boldsymbol{\theta})$.

For regression, standard loss functions include:

- absolute loss: $L[f(\boldsymbol{x}, \boldsymbol{\theta}), y] = |f(\boldsymbol{x}, \boldsymbol{\theta}) - y|$
- squared error loss: $L[f(\boldsymbol{x}, \boldsymbol{\theta}), y] = [f(\boldsymbol{x}, \boldsymbol{\theta}) - y]^2$.

For binary classification, labeled so that $y \in \{-1, 1\}$, a standard loss function is:

- $L[f(\boldsymbol{x}, \boldsymbol{\theta}], y) = I[-yf(\boldsymbol{x}, \boldsymbol{\theta}) > 0]$.

This last is an indicator function that is 1 iff the sign of y is different from the sign of $f(\boldsymbol{x}, \boldsymbol{\theta})$. Recall that the usual rule in classification is to predict 1 or -1 according to the sign of the classification rule $f(\boldsymbol{x}, \boldsymbol{\theta})$.

Assume that the joint distribution of future values of $(Y, \boldsymbol{X})$ is $P(y, \boldsymbol{x})$. Then the risk, or expected loss in a future prediction, is

$$R(\boldsymbol{\theta}) = \int_{\mathbb{R}^p \times \mathbb{R}} L[f(\boldsymbol{x}, \boldsymbol{\theta}), y] \, dP(y, \boldsymbol{x}).$$

But this cannot be calculated without knowing $P(y, \boldsymbol{x})$.

Consequently, people use the training sample to calculate the empirical risk:

$$R_e(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} L[f(\boldsymbol{x}_i, \boldsymbol{\theta}), y_i].$$

We want to find a bound on the risk such that with high probability,

$$R(\boldsymbol{\theta}) \leq R_e(\boldsymbol{\theta}) + B.$$

That will tell us how bad our empirical risk is as an estimate of the true risk.

In the context of two-class classification, Vapnik (1995, *Statistical Learning Theory*, Wiley) showed that with probability q,

$$R(\boldsymbol{\theta}) \leq R_e(\boldsymbol{\theta}) + \sqrt{\frac{1}{n}[v + \ln(v/h) - \ln(q/4)]}$$

where v is a non-negative integer called the Vapnik-Červonenkis (VC) dimension.

Note that the bound:

- does not depend on the $P(y, \boldsymbol{x})$;
- assumes that the training sample is a random sample from $P(y, \boldsymbol{x})$;
- is simple to compute, if v is known;
- can exceed 1, in which case it is useless.

The VC dimension depends upon the class $\mathcal{F}$. To start, we consider only the two-class discrimination problem, so $f(\boldsymbol{x}, \boldsymbol{\theta}) \in \{-1, 1\}$.

A given set of n points can be labeled in 2^n possible ways. If for any such labeling, there is a member of $\mathcal{F}$ that can correctly assign those labels, then we say that the set of points is shattered by $\mathcal{F}$.

The VC dimension for $\mathcal{F}$ is defined as the maximum number of points (i.e., training data) that can be shattered by the elements of $\mathcal{F}$.

Note: If the VC dimension is v, then there exists at least one set of v points that can be shattered. In general, it is not true that *every* set of v points can be shattered.

intentional blank

For the two-class linear discrimination problem in $\mathbb{R}^p$, one can prove that the VC dimension is $v = p + 1$. Thus the bound on the risk gets large as p increases.

One expects that as the elements of $\mathcal{F}$ get more flexible, then the VC dimension should increase. But the situation is complex. Consider $\mathcal{F} = \{f(x, \theta)\}$ where

$$f(x,\theta) = \begin{cases} 1 & \textbf{if } \sin(\theta x) > 0 \\ -1 & \textbf{if } \sin(\theta x) \leq 0. \end{cases}$$

Select the points $\{x_i = 10^{-i}\}$ for $i = 1, \ldots, n$. Let y_i be the label of x_i. The one can show that for any choice of labels,

$$\theta = \pi \left[1 + \sum_{i=1}^{n} \frac{1}{2}(1 - y_i)10^i \right]$$

gives the correct classification (example due to Levin and Denker).

Thus a one-parameter family can have infinite VC dimension.

intentional blank

Consider the 1-nearest neighbor classifier. This $\mathcal{F}$ has infinite v, since any number of arbitrarily labeled points can be shattered. And the empirical risk is zero (unless two observations with opposite labels coincide). So in this case the VC bound gives no useful information.

Burges (1998; *Data Mining and Knowledge Discovery*, **2**, 121-167) considers a "notebook" classifier for a 50/50 mix of population types A and B. The notebook has m pages; one writes down the labels of the first m training observations, for $m < n$. With all subsequent data, predict type A.

The empirical risk (under standard classification loss) for the first m values is 0; the empirical risk for the next $n - m$ is .5. The true risk on future samples is .5. And the VC dimension is $v = m$.

The VC-dimension approach to bounding error grew up in the context of support vector machines. So it is worth noting that for nonlinear machines, the VC dimension is $v = \dim(\mathcal{H}) + 1$, where $\mathcal{H}$ is the higher-dimensional space into which the data are mapped. (Recall section 7.1.3.)

Thus for some kernels, the VC dimension is infinite. To see this in the case the radial basis function kernel, note that one can shatter a set of points by putting a small normal distribution on top of each, and labeling it as needed. (This is like the 1-nearest-neighbor classifier.)

Nonetheless, SVMs peform very well. A good choice of kernel enables parsimonious solutions.

Similarly to classification, one can find a VC-bound on the risk in regression:

$$R(\boldsymbol{\theta}) \leq \frac{R_e(\boldsymbol{\theta})}{(1 - c\sqrt{\delta})_+}$$

where

$$\delta = \frac{a}{n}\left[v + v \ln \frac{bn}{v} - \ln \frac{q}{4}\right].$$

As before, the probability that this bound holds is $1 - q$.

The bound was developed by Cherkassky and Mulier (1998; *Learning From Data*, 108-11). One can tune the constants a, b, c to the application. Cherkassky and Mulier recommend taking $a = b = c = 1$ for regression applications.

The bound tends to be loose.

To use the VC dimension v for regression problems, one needs to extend its definition from classes of dyadic (i.e., +1/-1) functions $\mathcal{F}$ to classes of real-valued functions.

The strategy is to take any class of real-valued functions $\{f(\boldsymbol{x}, \boldsymbol{\theta})\}$ and create a set of dyadic functions

$$f_y(\boldsymbol{x}, \boldsymbol{\theta}) \begin{cases} 1 & \textbf{if } f(\boldsymbol{x}, \boldsymbol{\theta}) - y > 0 \\ -1 & \textbf{if } f(\boldsymbol{x}, \boldsymbol{\theta}) - y \leq 0 \end{cases}$$

where y is in the range R of f. The VC-dimension v_y for $\{f_y(\boldsymbol{x}, \boldsymbol{\theta})\}$ is now defined in the same way as before.

The VC dimension for the regression class $\mathcal{F}$ is $v = \sup_R\{v_y\}$.

Vapnik advocates Structural Risk Minimization to exploit the VC bound.

Consider a nested sequence of function classes:

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \ldots \subset \mathcal{F}_k = \mathcal{F}$$

where k may go to infinity. Let class $\mathcal{F}_i$ have VC dimension v_i, and this sequence is strictly increasing: $v_1 < v_2 < \ldots < v_k$.

For example, in classification $\mathcal{F}_1$ might be linear discrimainators, $\mathcal{F}_2$ might be quadratic discriminant functions, and so forth. It is known that the VC dimension for a polynomial discriminator of degree d in $\mathbb{R}^2$ is $(d^2 + 3d + 2)/2$ (Vapnik, 1995).

The SRM strategy is to start at $\mathcal{F}_1$ and increase. The empirical risk for the best function in $\mathcal{F}_i$ decreases monotonically in i, and the second term in the risk bound increases monotonically in i. One stops increasing the class at the i^* for which the risk bound is minimized.

Suppose one has a large, complex dataset that contains multiple kinds of structures and/or noise, e.g.:

- 40% of the data follow $Y = \alpha_0 + \sum_{i=1}^{p} \alpha_i X_i + \epsilon$
- 30% of the data follow $Y = \beta_0 + \sum_{i=1}^{p} \beta_i X_i + \epsilon$
- 30% of the data are noise.

What can one do to analyze cases like this? One should assume that the data miner has little prior knowledge of the kinds of structure that might be present.

The standard approach in linear regression is to use S-estimators, which look for the thinnest strip (think of a transparent ruler) which covers some prespecified (but larger than 50%) fraction of the data.

This strategy breaks down in high dimensions, or when the structures of interest contain less than 50% of the sample, or when fitting complex nonlinear models.

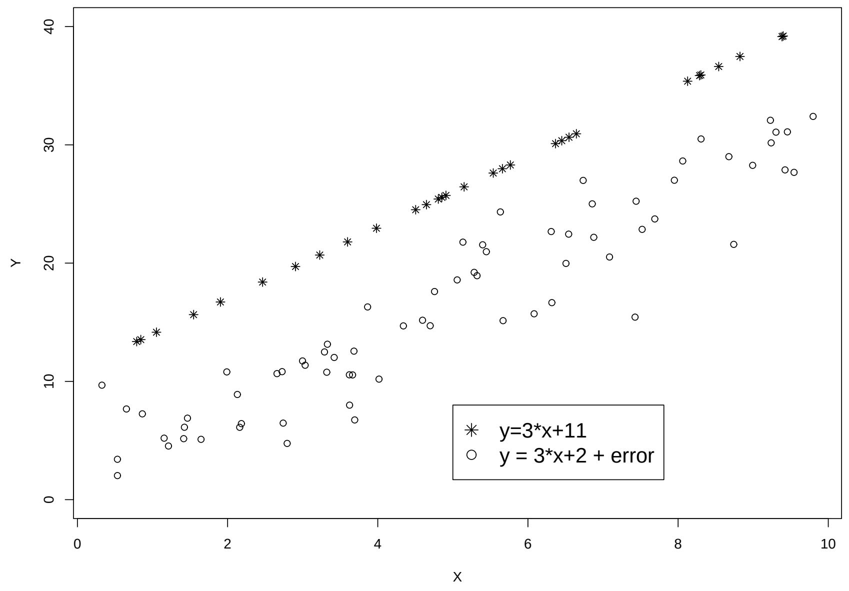
One wants a solution strategy that applies to more general cases, including:

- linear and non-linear regression in high dimensions,
- multidimensional scaling,
- cluster analysis.

The approach described in this section can be extended to other cases as well.

11.1 Hidden Structure in Regression

Consider the graph below, for a simple regression problem. It is clear that the data come from two different models, and that any naive attempt at regression will miss both of the interesting structures and find some kind of average solution.



In the linear regression scenario, assume the observations are $\{Y_i, \mathbf{X}_i\}$ for $i = 1, n$ and that Q percent of these follow the model

$$Y_i = \beta_0 + \beta_1 X_{i1} + \ldots + \beta_p X_{ip} + \epsilon_i \quad \text{where} \quad \epsilon_i \sim N(0, \sigma)$$

where Q, $\boldsymbol{\beta}$, and σ are unknown.

One can refer to the Q% of the data as "good" and the rest as "bad".

Simple Idea:

Start small, with a subsample of only **good** observations
$\Rightarrow$ add only **good** observations
$\Rightarrow$ end with a large subsample of **good** observations.

General procedure:

1. Strategically choose an initial set of d starting subsamples S_j, each of size m
2. Grow the subsamples by adding consistent data
3. Select the largest subsample.

The algorithm for choosing the starting subsamples and growing them efficiently is important in practice.

One starts with a guess about Q, the fraction of good data. In general, this is unknown, so one might pick a value that is reasonable given

- domain knowledge about the data collection
- scientific interest in a fraction of the data.

From the full dataset $\{Y_i, \mathbf{X}_i\}$ one selects, without replacement, d subsamples S_j of size m.

One needs to choose d and m to ensure that at least one of the starting subsamples S_j has a very high probability C of consisting entirely of good data (i.e., data that come from the model).

Preset a probability C that determines the chance that the algorithm will work.

The value m, which is the size of the starting-point random subsamples, should be the smallest possible value that allows one to calculate a goodness-of-fit measure. In the case of multiple linear regression, that value is $p+2$, and the natural goodness-of-fit measure is R^2.

One solves the following equation for d:

$$C = \mathbb{P}[\ \textbf{at least one of } S_1, \ldots, S_d \textbf{ is all good}\ ] = 1 - (1 - Q^{p+2})^d.$$

Example: $Q = .8$, $c = .95$, $m = 3$ $(p = 1)$:

$$.95 = 1 - [1 - (.8)^{p+2}]^d \qquad \longrightarrow \quad d = 5$$

Given the d starting-point subsamples S_j, one grows each one of them by adding observations that do not lower the goodness-of-fit statistic (R^2).

Conceptually, for a particular S_j, one could cycle through all of the observations, and on each cycle augment S_j by adding the observation that provided the largest value of R^2. This cycling would continue until no observation can be added to S_j without decreasing the R^2.

One does this for all of the subsamples S_j. At the end of the process, each augmented S_j would have size m_j and goodness-of-fit R_j^2. The augmented subsample that achieves a large value of m_j and a large value of R_j^2 is the one that captures the most important structure in the data.

Then one can remove the data in S_j and iterate to find the next-most important structure in the dataset.

In practice, the conceptual algorithm which adds one observation per cycle is expensive when the dataset is large or when one is fitting a complex model (e.g., doing MARS fits rather than multiple regression). For this reason, one might use a two-step procedure to add observations.

Fast Search

- Sequentially sweep through all observations not in S_i.
- If the observation improves the fitness measure (or perhaps only lowers it by a small amount η), then
  $\rightarrow$ add observation to S_j
  $\rightarrow$ set $m_j = m_j + 1$.

If $m_j < kn$ then implement slow search.

Slow Search

- Add the observation that improves the FM the most or decreases the fitness measure the least (regardless of η).
- Repeat until $m_j = kn$.

The analyst may pick a value for η that seems appropriately small, and a value for k that seems appropriately large. These choices determine the runtime of the algorithm and should reflect practical constraints.

The fast search is greedy, and the order of observations in the cycling matters. The slow search is less greedy; order does not matter, but it adds myopically. The fast search can add many observations per cycle through the data, but the slow search always adds exactly one.

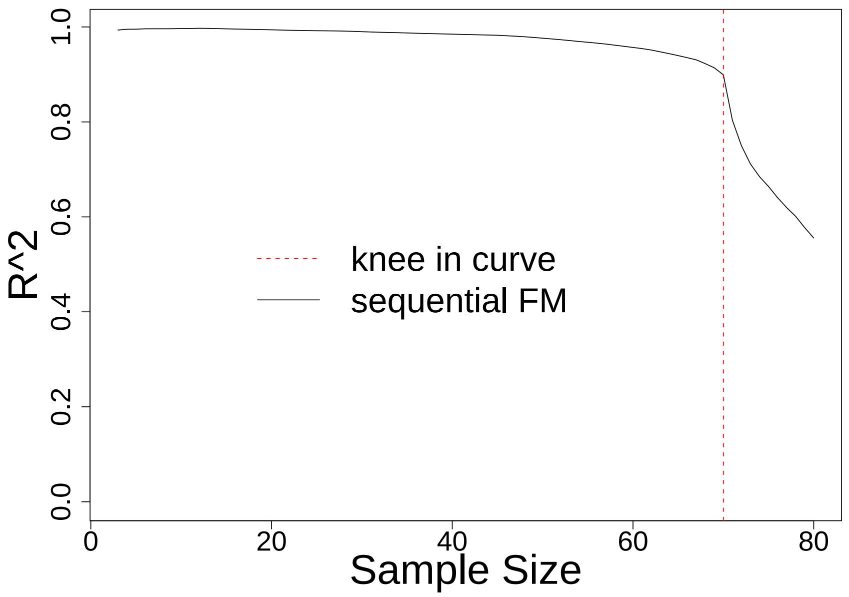
If speed is truly important, then there are other ways to accelerate the algorithm. A standard strategy would be to increase the number of starting-point subsamples and combine those that provide similar models and fits as they grow.

The main concern is not to enumerate all $\begin{pmatrix} n \\ Qn/100 \end{pmatrix}$ possible subsamples.

Some further comments are useful.

1. One does not need to terminate the search at some preset value kn; one can just grow until the goodness-of-fit measure deteriorates too much.
2. The goodness-of-fit measure should not depend upon the sample size. For R^2 this is easy, since it is just the proportion of variation in Y explained by X. For larger p, if one is doing stepwise regression to select variables, then one wants to use an AIC or Mallows' C_p statistic to adjust the tradeoff in fit between the number of variables and the sample size.
3. Other measures of fit are appropriate for nonparametric regression, such as cross-validated within-subsample squared error. But this adds to the computational burden.
4. One can and should monitor the fit as new observations are added. When one starts to add bad data, this is quickly visible, and there is a clear "slippery-slope" effect.

To see how the slippery-slope occurs, and the value of monitoring fit as a function of order of selection, consider the plot below. This plot is based on using R^2 for fitness with the double-line data shown previously. The total sample size is 80, and 70 observations were generated exactly on a line, as indicated by the knee in the curve.



11.2 Hidden Structure in Multidimensional Scaling

Multidimensional scaling (MDS) starts with a proximity matrix that gives approximate distances between all pairs in a set of objects. These distances are often close to a true metric.

The purpose of MDS is to find a low-dimensional plot of the objects such that the inter-object distances are as close as possible to the values given in the proximity matrix. That representation automatically puts similar objects near each other. This is done in terms of a least squares fit to the values in the proximity matrix, by minimizing the stress function:

$$\mathbf{Stress}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n) = \left[\sum_{i \neq i'} (d_{ii'} - \|\boldsymbol{z}_i - \boldsymbol{z}_{i'}\|) \right]^{1/2}$$

where $\boldsymbol{z}_i$ is the location assigned to pseudo-object i in the low-dimensional space and $d_{ii'}$ is the entry in proximity matrix.

The classic example is to take the entries in the proximity matrix to be the drive-time between pairs of cities. This is not a perfect metric, since roads curve, but it is approximately correct. MDS finds a plot in which the relative position of the cities looks like it would on a map (except the map can be in any orientation; north and south are not relevant).

MDS seeks is extremely susceptible to bad data. For example, if one had a flight tire while driving from Rockville to DC, this would create a seemingly large distance. The MDS algorithm would distort the entire map in an effort to put Rockville far from DC and still respect other inter-city drive times.

A very small proportion of outliers, or objects that do not fit well in a low-dimensional representation, can completely wreck the interpretability of an MDS plot. In many applications, such as text retrieval, this is a serious problem.

11.2.1 MDS Example

To test cherry-picking for MDS, consider the latitudes and longitudes of 99 eastern U.S. cities. The Euclidean distances between these cities gave the proximity matrix; the only stress in the MDS map is due to the curvature of the earth.

Perturb the proximity matrix by inflating a random proportion $1 - Q$ of the entries:

Bad Data	Distortion (%)	Stress
2	150	1.028
	500	2.394
10	150	1.791
	500	28.196
30	150	3.345
	500	9.351

To make things more interesting, we use not the traditional MDS using the stress measure defined previously, but rather Kruskal-Shephard non-metric scaling, in which one finds $\{\boldsymbol{z}_i\}$ to minimize

$$\textbf{Stress}_{KS}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n) = \frac{\sum_{i \neq i'} [\theta((\|\boldsymbol{z}_i - \boldsymbol{z}_{i'}\|) - d_{ii'}]^2}{\sum_{i \neq i'} d_{ii'}^2}$$

where $\theta(\cdot)$ is an arbitrary increasing function fit during the minimization. The result is invariant to monotonic transformations of the data, which is why it is nonparametric.

This minimization uses an alternating algorithm that first fixes $\theta(\cdot)$ and finds the $\{\boldsymbol{z}_i\}$, and then fixes the $\{\boldsymbol{z}_i\}$ and uses isotonic regression to find $\theta(\cdot)$. This shows that the algorithm can be used in complex fits.

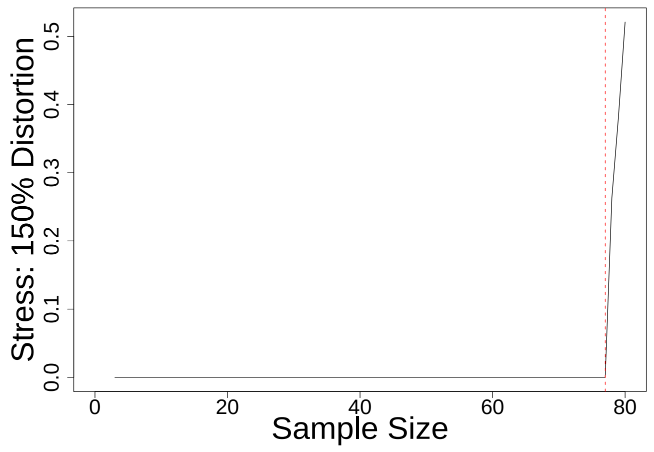
The goal is to cherry-pick the largest subset of cities whose intercity distances can be represented with little stress.

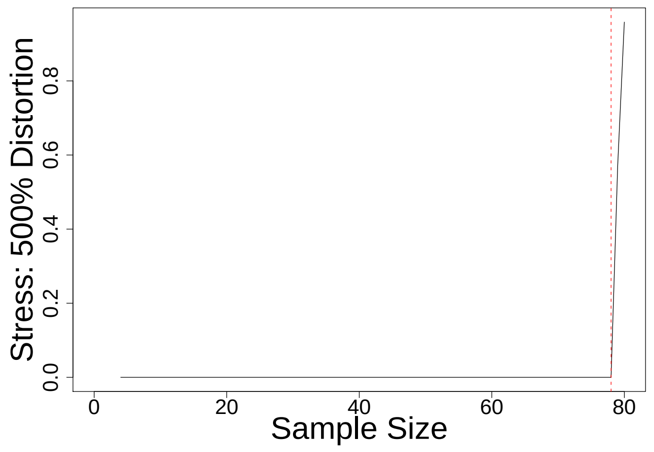
In MDS, the size m of the initial subsamples is 4 (since three points are always coplanar). We took $C = .99$ as the prespecified chance of getting at least one good subsample, and the table below shows the results.

True $1 - Q$ (%)	Distance Distortion (%)	Original Stress	n^a	n^*	Final Stress
2	150	1.028	80	80	4.78e-12
	500	2.394	80	80	4.84e-12
10	150	1.791	80	80	4.86e-12
	500	28.196	80	80	4.81e-12
30	150	3.345	80	77	4.86e-12
	500	9.351	80	78	4.78e-12

- Note: The stress of the undistorted dataset was 8.42×10^{-12}.

As before, one should inspect order-of-entry plots that display the stress against the cities chosen for inclusion. The following two plots are typical, and show the knee in the curve that occurs when one begins to add bad cities.





- This is a simple strategy for identifying primary stucture in complex datasets.
- The calculations are practical in computer-intensive applications, but one needs to use relatively greedy search algorithms to select observations for inclusion.
- The main computational problem is to scale the algorithm to accommodate very large samples, but there are obvious ways to address this.
- One can make probabilistic statements about the chance of having a good starting-point subsample, and this almost leads to a probabilistic guarantee on the result, but not quite.
- Simulation indicates this works well across a range of problems and situations.
- Once structure is discovered in the data, it can be removed and the process repeated to find second-order structure.
- The same approach can be used in cluster analysis, where fit is measured by the ratio of within-cluster variation to between-cluster variation.

References

- Breiman, Friedman, Olshen, and Stone, 1984, *Classification and Regression Trees*, Wadsworth.
- Green and Silverman, 1994, *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*, Chapman & Hall.
- Hall, 1992, *The Bootstrap and Edgeworth Expansion*, Springer-Verlag.
- Hastie and Tibshirani, 1990, *Generalized Additive Models*, Chapman & Hall.
- Hastie, Tibshirani, and Friedman, 2001, *The Elements of Statistical Learning*, Springer-Verlag.
- Mitchell, 1997, *Machine Learning*, McGraw Hill.
- Scott, 1992, *Multivariate Density Estimation: Theory, Practice, and Visualization*, Wiley.
- Vapnik, 1996, *The Nature of Statistical Learning Theory*, Springer-Verlag.
- Vidakovic, 1999, *Statistical Modeling by Wavelets*, Wiley.