Chapter 7

High Dimensional Problems

In observational studies we usually have observed predictors or covariates $X_1, X_2, \ldots, X_p$ and a response variable Y. A scientist is interested in the relation between the covariates and the response, a statistician summarizes the relationship with

$$E(Y|X_1, \ldots, X_p) = f(X_1, \ldots, X_n) \tag{7.1}$$

Knowing the above expectation helps us

- understand the process producing Y
- assess the relative contribution of each of the predictors
- predict the Y for some set of values $X_1, \ldots, X_n$.

One example is the air pollution and mortality data. The response variable Y is daily mortality counts. Covariates that are measured are daily measurements of particulate air pollution X_1, temperature X_2, humidity X_3, and other pollutants X_4,...,X_p.

Note: In this particular example we can consider the past as covariates. GAM is more appropriate for data for which this doesn't happen.

In this section we will be looking at a diabetes data set which comes from a study of the factors affecting patterns in insulin-dependent diabetes mellitus in children. The objective was to investigate the dependence of the level of serum C-peptide on various other factors in order to understand the patterns of residual insulin secretion. The response measurements Y is the logarithm of C-peptide concentration (mol/ml) at diagnosis, and the predictor measurements are age and base deficit, a measurement of acidity.

A model that has the form of (7.1) and is often used is

$$Y = f(X_1, \ldots, X_n) + \varepsilon \tag{7.2}$$

with ε a random error with mean 0 and variance σ^2 independent from all the $X_1, \ldots, X_p$.

Usually we make a further assumption, that ε is normally distributed. Now we are not only saying something about the relationship between the response and covariates but also about the distribution of Y.

Given some data, "estimating" $f(x_1, \ldots, x_n)$ can be "hard". Statisticians like to make it easy assuming a linear regression model

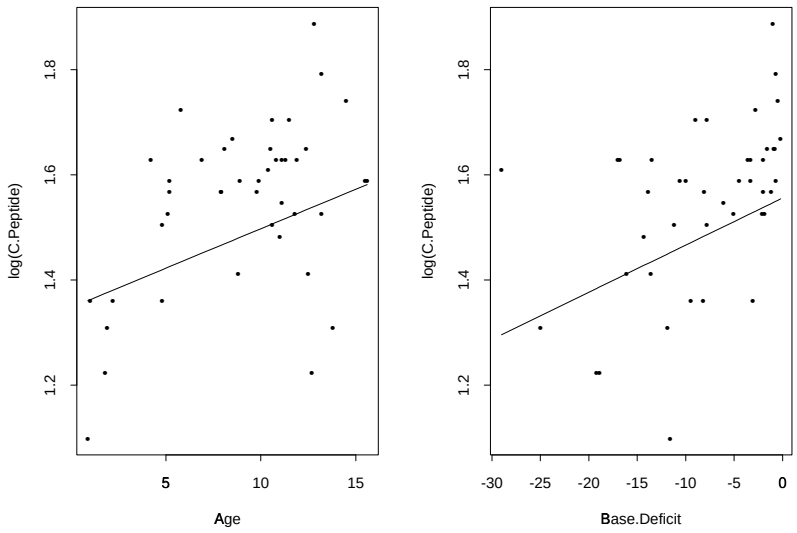
$$f(X_1, \ldots, X_n) = \alpha + \beta_1 X_1 + \ldots + \beta_p X_p$$

This is useful because it

- is very simple
- summarizes the contribution of each predictor with one coefficient
- provides an easy way to predict Y for a set of covariates $X_1, \ldots, X_n$.

It is not common to have an observational study with continuous predictors where there is "science" justifying this model. In many situations it is more useful to let the data "say" what the regression function is like. We may want to stay away from linear regression because it forces linearity and we may never see what $f(x_1, \ldots, x_n)$ is really like.

In the diabetes example we get the following result:



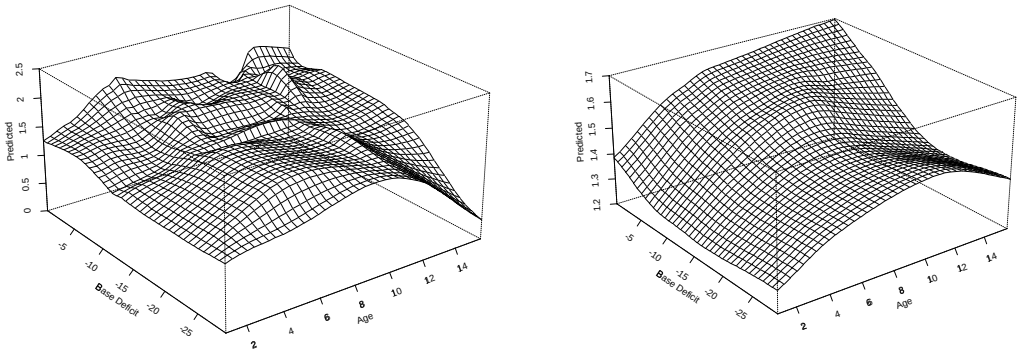
So does the data agree with the fits? Let's see if a "smoothed" version of the data agrees with this result.

But how do we smooth? Some of the smoothing procedures we have discussed may be generalized to cases where we have multiple covariates.

There are ways to define splines so that $g : I \subset \mathbb{R}^p \to \mathbb{R}$. We need to define knots in $I \subset \mathbb{R}^p$ and restrictions on the multiple partial derivative which is difficult but can be done.

It is much easier to generalize loess. The only difference is that there are many more polynomials to choose from: β_0, $\beta_0 + \beta_1 x$, $\beta_0 + \beta_1 x + \beta_2 y$, $\beta_0 + \beta_1 x + \beta_2 y + \beta_3 xy$, $\beta_0 + \beta_1 x + \beta_2 y + \beta_3 xy + \beta_4 x^2$, etc...

This is what we get when we fit local planes and use 15% and 66% of the data.



However, when the number of covariates is larger than 2 looking at small "balls" around the target points becomes difficult.

Imagine we have equally spaced data and that each covariate is in $[0, 1]$. We want to fit loess using $\lambda \times 100\%$ of the data in the local fitting. If we have p covariates and we are forming $p - dimensional$ cubes, then each side of the cube must have size l determined by $l^p = \lambda$. If $\lambda = .10$ (so its supposed to be very local) and $p = 10$ then $l = .1^{1/10} = .8$. So it really isn't local! This is known as *the curse of dimensionality*.

7.1 Projection Pursuit

One solution is projection-pursuit. It assumes a model of the form

$$f(X_1, \ldots, X_n) = \sum_{j=1}^{p} f_j\{\boldsymbol{\alpha}_j' \mathbf{X}\}$$

where $\boldsymbol{\alpha}_j' \mathbf{X}$ denotes a one dimensional projection of the vector $(X_1, \ldots, X_p)'$ and f_j is an arbitrary function of this projection.

The model builds up the regression surface by estimating these univariate regressions along carefully chosen projections defined by the $\boldsymbol{\alpha}_k$. Thus for $K = 1$ and

$p = 2$ the regression surface looks like a corrugated sheet and is constant in the directions orthogonal to $\alpha_{\mathbf{k}}$.

If you don't see how this solves the problem of dimensionality, the next section will help you understand.

7.2 Additive Models

Additive models are specific application of projection pursuit. They are more useful in scientific applications.

In additive models we assume that the response is linear in the predictors effects and that there is an additive error. This allows us to study the effect of each predictor separately. The model is like (7.2) with

$$f(X_1, \ldots, X_p) = \sum_{j=1}^{p} f_j(X_j).$$

Notice that this is projection pursuit with the projection

$$\boldsymbol{\alpha}_j' \mathbf{X} = X_j.$$

The assumption made here is not as strong as in linear regression, but its still quite strong. It's saying that the effect of each covariate is additive. In practice this may not make sense.

Example: In the diabetes example consider an additive model that models log(C-peptide) in terms of age X_1 and base deficit X_2. The additive model assumes that for two different ages x_1 and x_1' the conditional expectation of Y (seen as a random variable depending on base deficit):

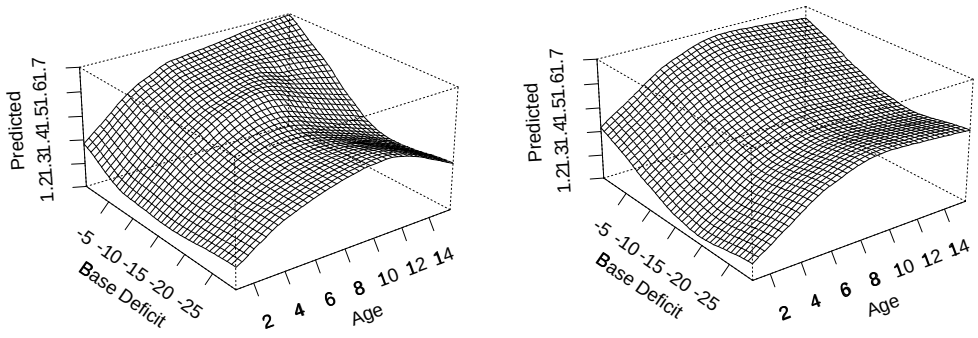
$$\mathrm{E}(Y|X_1 = x_1, X_2) = f_1(x_1) + f_2(X_2)$$

and

$$\mathrm{E}(Y|X_1 = x_1', X_2) = f_1(x_1') + f_2(X_2).$$

This say that the way C-peptide depends on base deficit only varies by a constant for different ages. It is not easy to disregard the possibility that this dependence changes. For example, at older ages the effect of high base deficit can be dramatically bigger. However, in practice we have too make assumptions like these in order to get some kind of useful description of the data.

Comparing the non-additive smooth (seen above) and the additive model smooth shows that it is not completely crazy to assume additivity.

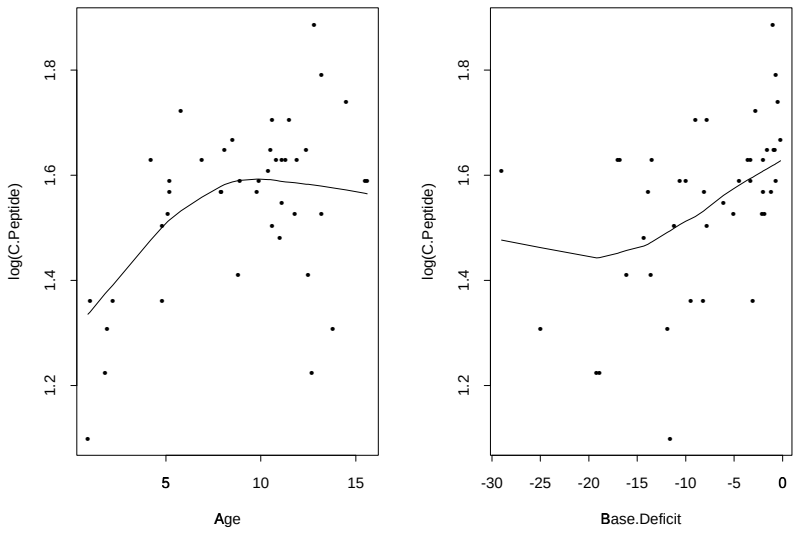


Notice that in the first plots the curves defined for the different ages are different. In the second plot they are all the same.

How did we create this last plot? How did we fit the additive surface. We need to estimate f_1 and f_2. We will see this in the next section.

Notice that one of the advantages of additive model is that no matter the dimension

of the covariates we know what the surface $f(X_1, \ldots, X_p)$ is like by drawing each $f_j(X_j)$ separately.



7.2.1 Fitting Additive Models: The Backfitting Algorithm

Conditional expectations provide a simple intuitive motivation for the backfitting algorithm.

If the additive model is correct then for any k

$$\mathrm{E}\left(Y - \alpha - \sum_{j \neq k} f_j(X_j) \,\middle|\, X_k\right) = f_k(X_k)$$

This suggest an iterative algorithm for computing all the f_j.

Why? Let's say we have estimates $\hat{f}_1, \ldots, \hat{f}_{p-1}$ and we think they are "good"

estimates in the sense that $\mathrm{E}\{f_j(X_j) - f_j(X_j)\}$ is "close to 0". Then we have that

$$\mathrm{E}\left(Y - \hat{\alpha} - \sum_{j=1}^{p-1} \hat{f}_j(X_j) \,\middle|\, X_p\right) \approx f_p(X_p).$$

This means that the partial residuals $\hat{\epsilon} = Y - \hat{\alpha} - \sum_{j=1}^{p-1} \hat{f}_j(X_j)$

$$\hat{\epsilon}_i \approx f_p(X_{ip}) + \delta_i$$

with the δ_i approximately IID mean 0 independent of the X_p's. We have already discussed various "smoothing" techniques for estimating f_p in a model as the above.

Once we choose what type of smoothing technique we are using for each covariate, say its defined by $S_j(\cdot)$, we obtain an estimate for our additive model following these steps

1. Define $\mathbf{f}_j = \{f_j(x_{1j}), \ldots, f_j(x_{nj})\}'$ for all j.
2. Initialize: $\alpha^{(0)} = \text{ave}(y_i)$, $\mathbf{f}_j^{(0)} =$ linear estimate.
3. Cycle over $j = 1, \ldots, p$

$$\mathbf{f}_j^{(1)} = S_j\left(\mathbf{y} - \alpha^{(0)} - \sum_{k \neq j} \mathbf{f}_k^{(0)} \,\middle|\, \mathbf{x}_j\right)$$

4. Continue previous step until functions "don't change", for example until

$$\max_j \left|\left|\mathbf{f}_j^{(n)} - \mathbf{f}_j^{(n-1)}\right|\right| < \delta$$

with δ is the smallest number recognized by your computer. In my computer using S-Plus its:

```
.Machine$double.eps = 2.220446e-16
```

Things to think about:

Why is this algorithm valid? Is it the solution to some minimization criterion? Its not MLE or LS.

7.2.2 Justifying the backfitting algorithm

The backfitting algorithm seems to make sense. We can say that we have given an intuitive justification.

However statisticians usually like to have more than this. In most cases we can find a "rigorous" justification. In many cases the assumptions made for the "rigorous" justifications too work are carefully chosen so that we get the answer we want, in this case that the back-fitting algorithm "converges" to the "correct" answer.

In the GAM book, H&T find three ways to justify it: Finding projections in L^2 function spaces, minimizing certain criterion with solutions from reproducing-kernel Hilbert spaces, and as the solution to penalized least squares. We will look at this last one.

We extend the idea of penalized least squares by considering the following criterion

$$\sum_{i=1}^{n}\left\{y_i - \sum_{j=1}^{p} f_j(x_{ij})\right\}^2 + \sum_{j=1}^{p}\lambda_j \int \{f_j''(t)\}^2\, dt$$

over all p-tuples of functions $(f_1, \ldots, f_p)$ that are twice differentiable.

As before we can show that the solution to this problem is a p-tuple of cubic splines with knots "at the data", thus we may rewrite the criterion as

$$\left(\mathbf{y} - \sum_{j=1}^{p}\mathbf{f}_j\right)'\left(\mathbf{y} - \sum_{j=1}^{p}\mathbf{f}_j\right) + \sum_{j=1}^{p}\lambda_j \mathbf{f}_j \mathbf{K}_j \mathbf{f}_j$$

where the $\mathbf{K}_j$s are penalty matrices for each predictor defined analogously to the $\mathbf{K}$ of section 3.3.

If we differentiate the above equation with respect to the function $\mathbf{f}_j$ we obtain $-2(\mathbf{y} - \sum_k \mathbf{f}_k) + 2\lambda_j \mathbf{K}_j \mathbf{f}_j = 0$. The $\hat{\mathbf{f}}_j$'s that solve the above equation must

satisfy:

$$\hat{\mathbf{f}}_j = (\mathbf{I} + \lambda_j \mathbf{K}_j)^{-1} \left(\mathbf{y} - \sum_{k \neq j} \hat{\mathbf{f}}_k \right), j = 1, \ldots, p$$

If we define the smoother operator $\mathbf{S}_j = (\mathbf{I} + \lambda_j \mathbf{K}_j)^{-1}$ we can write out this equation in matrix notation as

$$\begin{pmatrix} \mathbf{I} & \mathbf{S}_1 & \ldots & \mathbf{S}_1 \\ \mathbf{S}_2 & \mathbf{I} & \ldots & \mathbf{S}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{S}_p & \mathbf{S}_p & \ldots & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_p \end{pmatrix} = \begin{pmatrix} \mathbf{S}_1 \mathbf{y} \\ \mathbf{S}_2 \mathbf{y} \\ \vdots \\ \mathbf{S}_p \mathbf{y} \end{pmatrix}$$

One way to solve this equation is to use the Gauss-Seidel algorithm which in turn is equivalent to solving the back-fitting algorithm. See Buja, Hastie & Tibshirani (1989) Ann. Stat. 17, 435–555 for details.

Remember that that for any set of linear smoother

$$\hat{\mathbf{f}}_j = \mathbf{S}_j \mathbf{y}$$

we can argue in reverse that it minimizes some penalized least squares criteria of the form

$$(\mathbf{y} - \sum_j \mathbf{f}_j)'(\mathbf{y} - \sum_j \mathbf{f}_j) + \sum_j \mathbf{f}_j'(\mathbf{S}_j^- - I)\mathbf{f}_j$$

and conclude that it is the solution to some penalized least squared problem.

7.2.3 Standard Error

When using `gam()` in S-Plus we get point-wise standard errors. How are these obtained?

Notice that our estimates $\hat{\mathbf{f}}_j$ are no longer of the form $\mathbf{S}_j \mathbf{y}$ since we have used a complicated backfitting algorithm. However, at convergence we can express $\hat{\mathbf{f}}_j$ as $\mathbf{R}_j \mathbf{y}$ for some $n \times n$ matrix $\mathbf{R}_j$. In practice this $\mathbf{R}_j$ is obtained from the last calculation of the $\hat{\mathbf{f}}_j$'s but finding a closed form is rarely possible.

Ways of constructing confidence sets is not straight forward, and (to the best of my knowledge) is an open area of research.

7.3 Classification Algorithms and Regression Trees

This is from the book by Breiman et. al.

At the university of California, San Diego Medical Center, when a heart attack patient is admitted, 19 variables are measured during the first 24 hours. They include BP, age and 17 other binary covariates summarizing the medical symptoms considered as important indicators of the patient's condition.

The goal of a medical study can be to develop a method to identify high risk patients on the basis of the initial 24-hour data.

The next figure shows a picture of a tree structured classification rule that was produced in the study. The letter F means no high and the letter G means high risk.

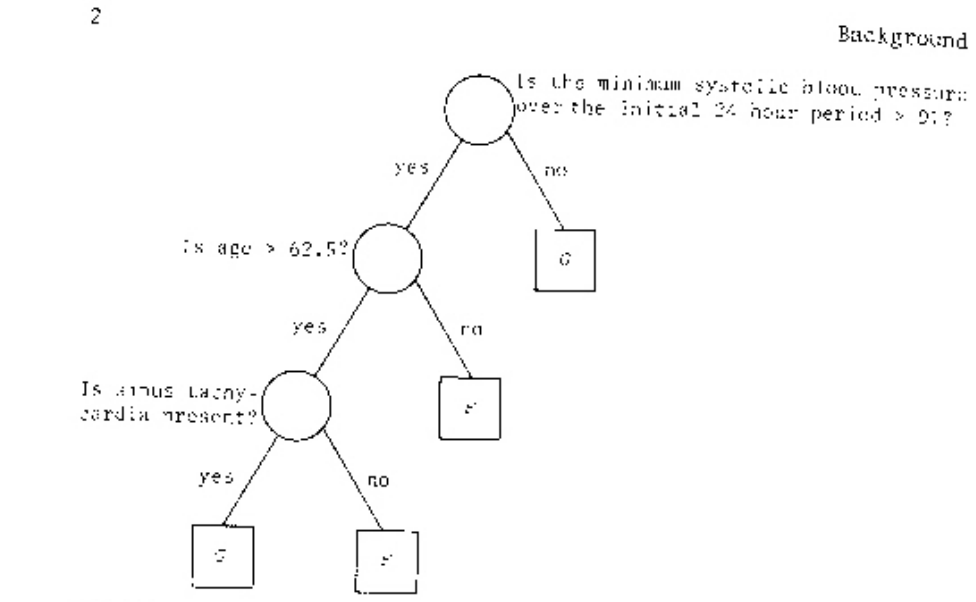


FIGURE 1.1

How can we use data to construct trees that give us useful answers. There is a large amount of work done in this type of problem. We will give a brief description in this section.

7.3.1 Classifiers as Partitions

Suppose we have a categorical outcome $y \in \mathcal{C} = \{1, 2, \ldots, J\}$. We call $\mathcal{C}$ the set of classes. Denote with $\mathcal{X}$ the space of all possible covariates.

We can define a classification rule as a function $d(\mathbf{x})$ defined on $\mathcal{X}$ so that for every $\mathbf{x}$, $d(\mathbf{x})$ is equal to one of the numbers $1, \ldots, J$.

This could be considered a systematic way of predicting class membership from the covariates.

Another way to define classifiers is to partition $\mathcal{X}$ into disjoint sets $A_1, \ldots, A_j$ with $d(\mathbf{x}) = j$ for all $\mathbf{x} \in A_j$.

But how do we construct these classifiers from data?

7.3.2 What is truth?

We are now going to describe how to construct classification rules from data. The data we use to construct the tree is called the training set $\mathcal{L}$ which is simply $\{(\mathbf{x}_1, j_1), \ldots, (\mathbf{x}_n, j_n)\}$.

Once a classification rule $d(X)$ is constructed how do we define it's accuracy?

In this section we will define the *true misclassification rate* $R^*(d)$.

One way to estimate $R^*(d)$ is to draw another very large subset (virtually infinite)

from the same population as $\mathcal{L}$ and observe the rate of correct classification in that set. The proportion misclassified by d is our estimate of $R^*(d)$.

To make this definition more precise, define the space $\mathcal{X} \times \mathcal{C}$ as the set of all couples $(\mathbf{x}, j)$ where $\mathbf{x} \in \mathcal{X}$ and $j \in \mathcal{C}$. Let $\Pr(A, j)$ be a probability distribution on $\mathcal{X} \times \mathcal{C}$. Assume each element of $\mathcal{L}$ is an iid outcome from this distribution.

We define the misclassification rate as

$$R^*(d) = \Pr[d(\mathbf{x}) \neq j | \mathcal{L}] \tag{7.3}$$

with $(\mathbf{x}, j)$ an outcome independent of $\mathcal{L}$.

How do we obtain an estimate of this?

The *substitution estimate* simply counts how many times we are right with the data we have, i.e.

$$R(d) = \frac{1}{N} \sum_{n=1}^{N} 1_{d(\mathbf{x}_n) \neq j_n}.$$

The problem with this estimate is that most classification algorithms construct d trying to minimize the above equation. If we have enough covariates we can define a rule that always has $d(\mathbf{x}_n) = j_n$ and randomly allocates any other $\mathbf{x}$. This has an $R(d) = 0$ but one can see that, in general, $R^*(d)$ will be much bigger.

Another popular approach is the *test sample* estimate. Here we divide the data $\mathcal{L}$ into two groups $\mathcal{L}_1$ and $\mathcal{L}_2$. We then use $\mathcal{L}_1$ to define d and $\mathcal{L}_2$ to estimate $R^*(d)$ with

$$R(d) = \frac{1}{N_2} \sum_{\mathbf{x}_n \in \mathcal{L}_2} 1_{d(\mathbf{x}_n) \neq j_n}$$

with N_2 the size of $\mathcal{L}_2$. A popular choice for N_2 is 1/3 of N, the size of $\mathcal{L}$.

A problem with this procedure is that we don't use 1/3 of the data when constructing d. In situations where N is very large this may not be such a big problem.

The third approach is cross validation. We divide the data into many subsets of equal (or approximately equal) size $\mathcal{L}_1, \ldots \mathcal{L}_V$, define a d_v for each of these

groups, and use the estimate

$$R(d) = \frac{1}{V}\sum_{v=1}^{V}\frac{1}{N_v}\sum_{\mathbf{x}_n\in\mathcal{L}_v} 1_{d(\mathbf{x}_n)\neq j_n}.$$

7.3.3 Bayes Rule

The major guide that has been used in the construction of classifiers is the concept of the Bayes rule. A Bayes rule is the d_B for which

$$\Pr[d_B(\mathbf{x}) \neq j) \leq \Pr[d(\mathbf{x}) \neq y].$$

for all classification rules $d(\mathbf{x})$.

If we assume that $\Pr(A|j)$ has a probability density $f_j(\mathbf{x})$ such that

$$\Pr(A|j) = \int_A f_j(\mathbf{x})\, dx$$

then we can show that

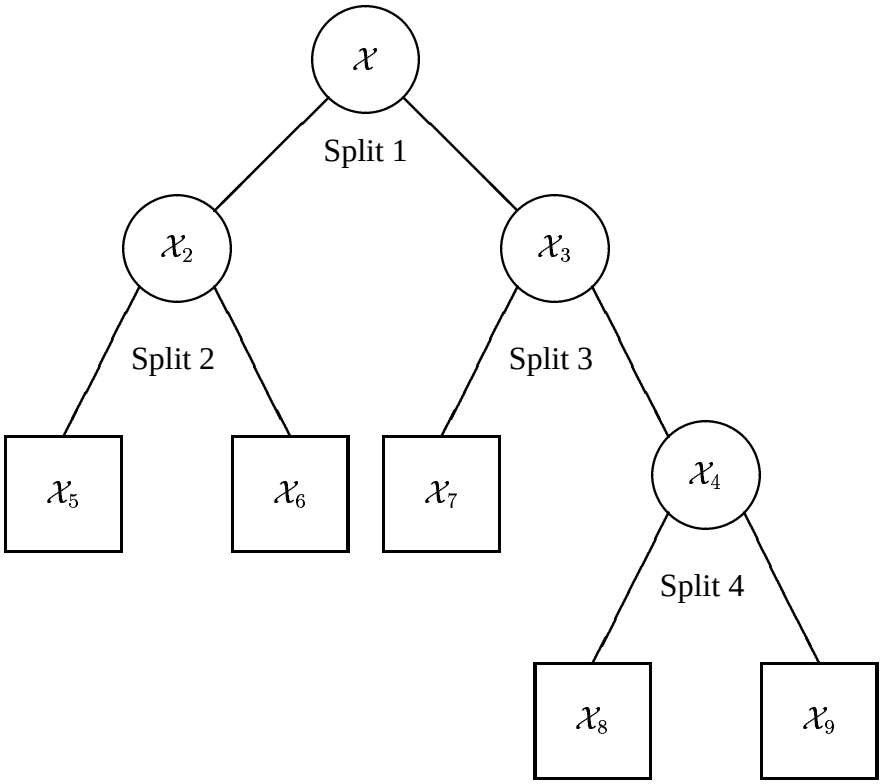
$$d_B(\mathbf{x}) = j \text{ on } A_j = \{\mathbf{x}; f_j(\mathbf{x})\Pr(j) = \max_i f_i(\mathbf{x})\Pr(j)\}.$$

Discriminant analysis, kernel density estimation, and k-th nearest neighbor smoothing attempt to estimate $f_j(\mathbf{x})$ and $\Pr(j)$ in order to estimate the Bayes rule. They make many assumptions so the methods are not always useful.

7.3.4 Constructing tree classifiers

Notice how big the space of all possible classifiers is. In the simple case where $\mathcal{X} = \{0,1\}^p$ this space has 2^p elements.

Binary trees are a special case of this partition. Binary trees are constructed by repeated splits of the subsets of $\mathcal{X}$ into two descendant subsets, beginning with $\mathcal{X}$ itself.



The subsets created by the splits are called *nodes*. The subsets which are not split are called terminal nodes.

Each terminal nodes gets assigned to one of the classes. So if we had 3 classes we could get $A_1 = \mathcal{X}_5 \cup \mathcal{X}_9$, $A_2 = \mathcal{X}_6$ and $A_3 = \mathcal{X}_7 \cup \mathcal{X}_8$. If we are using the data we assign the class most frequently found in that subset of $\mathcal{X}$. We call these classification tress.

Various question still remain to be answered

- How do we define truth?
- How do we construct the trees from data?

- How do we assess trees, i.e. what makes a good tree?

The first problem in tree construction is how to use $\mathcal{L}$ to determine the binary splits of $\mathcal{X}$ into smaller and smaller pieces. The fundamental idea is to select each split of a subset so that the data in each of the descendant subsets are "purer" than the data in the parent subset.

This can be implemented in the following way

- Define the node proportions $p(j|t)$ to be the proportion of cases $\mathbf{x}_n \in t$ belonging to class j so that $\sum_j p(j|t) = 1$.
- Define a measure of impurity $i(t)$ as a nonnegative function ϕ such that it reaches its maximum at $\phi(1/n, \ldots, 1/n)$, $\phi(1, 0, \ldots, 0) = 0$, and is symmetric with respect to its entries.

  A popular example is the entropy

  $$i(t) = -\sum_{j=1}^{J} p(j|t) \log p(j|t),$$

  but there are many other choices.
- Define a set $\mathcal{S}$ of binary splits s at each node. Then we chose the split that minimize the impurity of the new left and right nodes

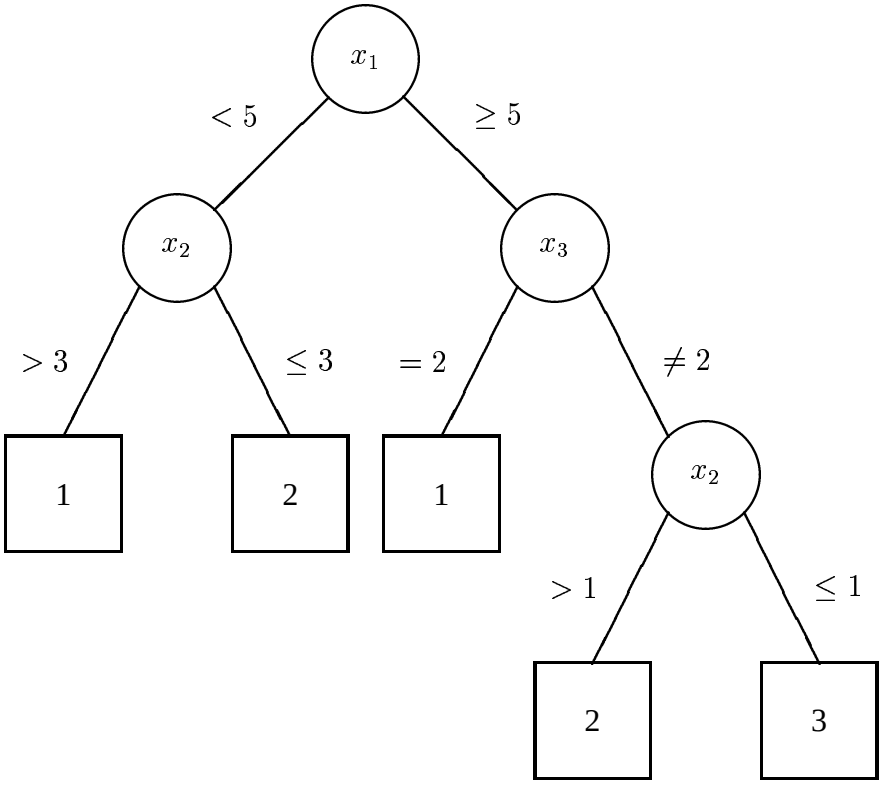
  $$\Delta i(s, t) = i(t) - p_L i(t_l) + p_R i(t_R)$$

There are many different possible splits. For continuous variables there are an infinite amount. We need to define the set of splits $\mathcal{S}$ that we consider.

Most implementations require that the the splits are defined by only one covariate, but fancier versions permit the use of linear combinations.

If the covariate is continuous or ordered then the split must be defined by $x < c$ and $x \geq c$.

If the covariate is categorical then we simply consider all splits that divide original set into two.



Now all we need is a stopping rule and we are ready to create trees. A simple stopping rule is that $\Delta i(s,t) < \delta$, but this does not work well in practice.

What is usually done is that we let the trees grow to a size that is bigger than what we think makes sense and then prune. We remove node by node and compare the trees using estimates of $R^*(d)$.

Sometimes to save time and/or choose smaller trees we define a penalized criterion based on $R^*(d)$.

The big issue here is *model selection*. The model selection problem consists of four orthogonal components.

1. Select a space of models
2. Search through model space
3. Compare models
    - of the same size
    - of different sizes (penalize complexity)
4. Assess the performance of a procedure

Important points:

- Components 2 and 3 are often confused (e.g., in stepwise regression). That's bad.
- People often forget component 1.
- People almost always ignore component 4; it can be the hardest.

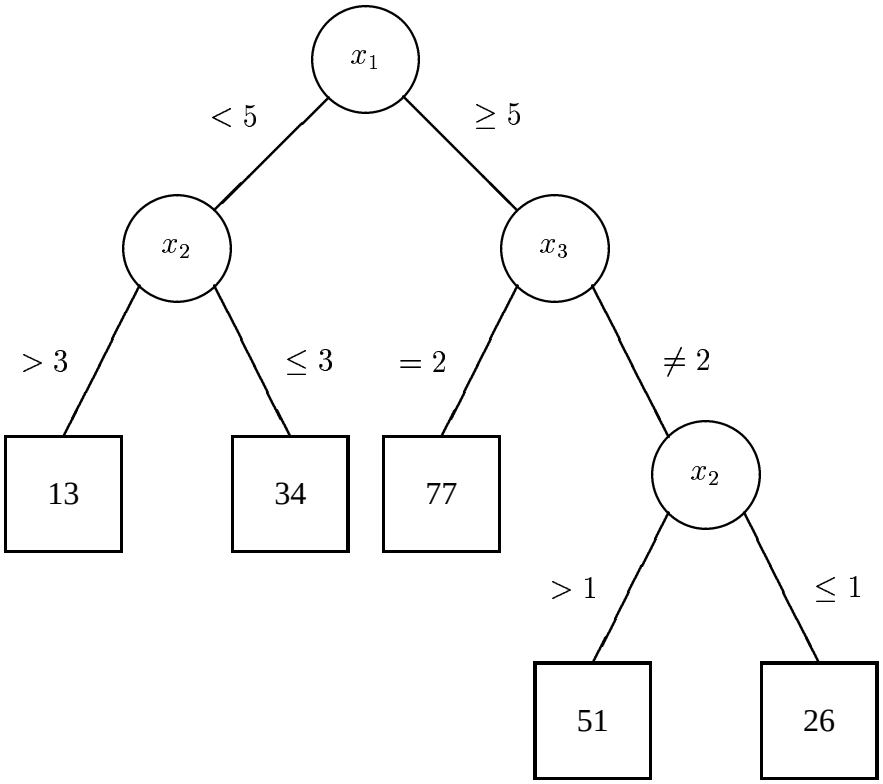
Better trees may be found by doing a one-step "look ahead," but this comes with the cost of a great increase in computation.

7.3.5 Regression Trees

If instead of classification we are interested in predicting we can assign a predictive value to each of the terminal nodes. Notice that this defines an estimate for the regression function $\mathrm{E}(Y|X_1, \ldots, X_n)$ that is like a multidimensional bin smoother. We call these regression trees.

Regression trees are constructed in a similar way to classification trees. They are used for the case where Y is a continuous random variable.

A regression tree partitions x-space into disjoint regions A_k and provides a fitted value $\mathrm{E}(y|x \in A_k)$ within each region.



In other words, this is a decision tree where the outcome is a fitted value for y.

We need a new definition $d(\mathbf{x})$ and $R^*(d)$.

Now $d(\mathbf{x}_j)$ will simply be the average of the terminal node where $\mathbf{x}_j$ lies. So $d(\mathbf{x})$ defines a step-function $\mathbb{R}^p \to \mathbb{R}$.

Instead of misclassification rate, we can define mean squared error

$$R^*(d) = \mathrm{E}[Y - d(\mathbf{x})]^2$$

The rest is pretty much the same.

7.3.6 General points

From Karl Broman's notes.

- This is most natural when the explanatory variables are categorical (and it is especially nice when they are *binary*).
- There is nothing special about the tree structure...the tree just partitions x-space, with a fitted value in each region.
- **Advantage**: These models go after *interactions* immediately, rather than as an afterthought.
- **Advantage**: Trees can be easy to explain to non-statisticians.
- **Disadvantage**: Tree-space is huge, so we may need *a lot* of data.
- **Disadvantage**: It can be hard to assess uncertainty in inference about trees.
- **Disadvantage**: The results can be quite variable. (Tree selection is not very *stable*.)
- **Disadvantage**: Actual *additivity* becomes a mess in a binary tree. This problem is somewhat alleviated by allowing splits of the form $x_1 + bx_2 < (\geq)\ d$.

Computing with trees

R: `library(tree)`; `library(rpart)` [MASS, ch 10]

An important issue: Storing trees

Binary trees are composed of nodes (root node, internal nodes and terminal nodes).

Root and internal nodes:

- Splitting rule (variable + what goes to right)
- Link to left and right daughter nodes
- Possibly a link to the parent node (null if this is the root node)

Terminal nodes:

- Fitted value
- Possibly a link to the parent node

C: Use pointers and structures (`struct`)

R: It beats me. Take a look.

Ref: Breiman et al (1984) Classification and regression trees. Wadsworth.