1 Chapter 6. Strategies and Methods for Prediction

Greg Ridgeway, The RAND Corporation, Statistics Group

Running title: Strategies and Methods for Prediction

Keywords:
Boosting
AdaBoost
Prediction
Naïve Bayes
Neural networks
Support vector machines
Logistic regression
Probit regression
Generalized linear models
Cox model
Survival models
Proportional hazard model
Poisson regression
Least squares
Nearest neighbor classifier
Loss function
Calibration
Brier score
iteratively reweighted least squares
linear discriminant analysis
kernel methods
splines
additive models
natural spline
basis functions
R project
Murphy decomposition
trees
CART

2 Introduction to the prediction problem

Many data mining problems depend on the construction of models, equations, or machines that are able to predict future outcomes. Although prediction is an important component of data mining, the abundance of methods such as linear models, neural networks, decision trees, and support vector machines can make a seemingly simple prediction problem rather confusing. Other chapters in this volume focus on particular methods. In this chapter we will briefly assemble these ideas into a common framework within which we can begin to understand how all these methods relate to one another.

In many applications we are not only interested in having accurate predictions in the future but also in learning the relationship between the features of an observation and the outcomes. For example, we will consider an example of predicting at age 12 which students are likely to drop out of high school before age 18. Certainly we wish to have an accurate assessment of dropout risk, but we also wish to enumerate those factors that place certain students at greater risk. Understanding the mechanism relating the student features to the outcome helps formulate rules-of-thumb for identifying at-risk students and interventions targeting those students. Whether our goal is prediction alone or understanding the underlying mechanism or both, a well-structured and well-estimated prediction model is the first step in the process.

In this chapter, we introduce the basics of prediction problems. We offer strategies on how to relate the choice of prediction method to applications and describe several of the frequently used prediction methods. This is far from a complete catalog of available tools and strategies but rather, like the rest of this handbook, it represents a starting place from which the reader could springboard into other, more technical developments of the methods. After reading this chapter the data miner will know the fundamental prediction concepts and be able to think critically and creatively about solving specific prediction problems.

Throughout this chapter we will use the following notation. Generically we have a dataset containing N observations $(y_i, \mathbf{x}_i)$ where $i = 1, \ldots, N$. These observations come from some unknown process, generating mechanism, or probability distribution. The features, $\mathbf{x}_i$, often referred to as covariates, independent variables, or inputs, may be a vector containing a mix of continuous, ordinal, and nominal values. We wish to construct our prediction model, denoted as $f(\mathbf{x})$, so that it predicts the outcome, y_i, also called the response, the output, or the target, from the features.

2.1 Guiding examples

This chapter will focus on three examples to help explain the concepts and techniques. The applications generally focus on public policy issues but the same methods apply to problems spanning the scientific domains.

2.1.1 Regression: Cost of stroke rehabilitation

Regression problems (some engineering literature refers to these as estimation problems) involve the prediction of continuous outcomes. In 1997 the Balanced Budget Act mandated that the Centers for Medicare and Medicare Services (CMS) develop a prospective payment system for inpatient rehabilitation facilities. This system would determine how to allocate a $4.3 billion budget to facilities that provide care for individuals covered under Medicare, the United States' federal healthcare system for the elderly and disabled. Part of the development involved building a cost model that predicts cost of rehabilitation from patient features. From CMS billing records we obtained each patient's age, reason for the stay (stroke, hip fracture, etc.), and cost of care. From secondary sources we obtained functional ability scores that measured the patients' motor and cognitive abilities. The prediction problem involves developing a model so that for any future patient we can accurately predict cost of rehabilitation from the patient's features for accurate hospital reimbursement.

2.1.2 Classification: Detecting high school dropouts

The National Education and Longitudinal Study of 1988 (NELS:88) is an extensive data source surveying the attitudes and behaviors of a nationally representative sample of American adolescents. NELS:88 first surveyed its respondents as eighth graders and has conducted three waves of follow-up measures in 1990, 1992, and 1994. Student, family, and school level measures are included in each wave of the survey. Offering multiple item indicators of student's goals, ability, past achievement, and involvement in school, NELS:88 also includes detailed data on parenting style, parent/child behavior and interactions, religion, race/ethnicity, parents' occupation(s) and income, along with numerous other measures of family background. The strengths of this data set result from its large number of cases (over 15,000 students in this chapter's analyses), its comprehensiveness (measuring over 6000 variables), and its longitudinal design (allowing temporal as well as cross-sectional analyses). The examples will utilize data from the first three survey waves, analyzing information from the baseline grade 8 data to predict failure to complete high school.

2.1.3 Survival: Survival time of PBC patients

To show the breadth of loss functions available for consideration and the flexibility of prediction methods we include a clinical trial example. In this example the problem is to estimate survival time of patients suffering from primary biliary cirrhosis of the liver (PBC). Although analyses using survival models predict time until death or time in remission for medical treatment studies, the models are applicable outside the domain of medicine. Applications also include time until failure of a machine or part, time until a customer churns by abandoning their current telecommunication provider for a competitor, and time between when a gun is purchased and when it is confiscated or recovered at a crime scene.

2.2 Prediction model components

Prediction methods may differ in three main ways: the *loss function* or performance measure that it seeks to optimize, the *structural form* of the model, and the manner of obtaining model *parameter estimates* from training data. When considering a new or unfamiliar method, understanding these three basic components can go a long way toward realizing its limitations and advantages. Several methods may have the same structural form but differ on performance measures or scalability due to differences in how we estimate or learn the model from data. We will see that three established classification methods, naïve Bayes, logistic regression, and linear discriminant analysis, all have exactly the same structural form but differ on the loss function and parameter estimation method. In addition, the chapter on tree models in this handbook discusses models that all share the tree structure but may have different methods of forming splits and estimating the number of terminal nodes from the available data.

The next section discusses the most popular loss functions in use in statistics, machine learning, and data mining practice. Following that we give a concise catalog of some of the structural forms used in practice. Even after selecting a loss function and a structural form for our predictor, the main problem facing data miners today is getting those models fit to massive datasets. We will comment on the scalability issue as it arises but it continues to be an active area of research and progress. Some accurate methods that were assumed to be not scalable to large datasets now have been tuned and optimized for practical use.

3 Loss functions – what we are trying to accomplish

When developing a prediction model we usually have some performance measure that we want our model to optimize. The *loss function* is a function that takes as input a prediction model and produces a single number that indicates how well

that prediction model performs. This section reviews some of the most commonly used loss functions. The notation $J(f)$ indicates the loss function J evaluated for prediction model f.

3.1.1 Common regression loss functions

For regression problems the most widely used loss function is squared prediction error, which is the expected squared difference between the true value and the predicted value

$$J(f) = E_{y,\mathbf{x}}(y - f(\mathbf{x}))^2 \tag{1}$$

where the $E_{y,\mathbf{x}}$ represents the expectation operator that averages over all (y, $\mathbf{x}$) pairs drawn from some common distribution. By minimizing (1) we assure ourselves that, on average, new predictions will not be too far from the true outcome. The properties of expectation indicate that the $f(\mathbf{x})$ that minimizes (1) is $f(\mathbf{x}) = E(y|\mathbf{x})$, the average outcome at each value of $\mathbf{x}$. The probability distribution that generates the (y, $\mathbf{x}$) pairs is unknown and so we cannot compute (1) directly. Instead we rely on a sample based estimate

$$\hat{J}(f) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i))^2 \,. \tag{2}$$

Remember that it is almost *always* (1) that we really want to minimize but resort to (2) to guide us to a solution. There are many $f(\mathbf{x})$ that can make (2) arbitrarily small but usually only one that minimizes (1). The model fitting process will find an f that minimizes (2) subject to some constraints on its structural form. An unbiased evaluation of the performance of a particular choice for f requires a separate test dataset or some form of cross-validation. The chapter on performance analysis and evaluation in this volume describes this process in more detail.

Although squared error loss is the dominant loss function in most applied regression work, decades of work on robustness have demonstrated that squared error is highly sensitive to outliers, unusually large outcomes potentially from data contamination and spurious measurements. Absolute prediction error

$$J(f) = E_{y,\mathbf{x}} \mid y - f(\mathbf{x}) \mid \tag{3}$$

has its minimum when $f(\mathbf{x}) = \text{median}(y|\mathbf{x})$, the median outcome at each value of $\mathbf{x}$. For data mining applications prone to contamination the absolute prediction error may be preferable.

Other regression loss functions use variations on the above theme. For example, Huber (1964) proposed a loss function that behaves like squared-error near 0 and like absolute error when $y - f(\mathbf{x})$ exceeds some cutoff, providing some protection against extreme y values. Support vector machine regression methods commonly use a loss function that is zero when $y - f(\mathbf{x})$ is less than some cutoff and then behaves like absolute error for deviations greater than the cutoff. At this point, simply note that there is considerable flexibility in terms of specifying what it means to have a prediction be "close" to the true value and that the different choices result in different prediction models. The implications will follow shortly in some examples.

3.1.2 Common classification loss functions

While regression problems focus on predicting continuous outcomes, classification methods attempt to label observations as one of k categories. The most common loss functions used in classification problems include misclassification rate, expected cost, and log-likelihood. Misclassification rates are generally the primary measure on which methods are compared. In fact, it is generally what problem solvers are aiming to minimize when considering a particular classification problem. In almost all problems, however, a false positive has a different cost than a false negative. High school dropouts are thought to cost twenty times more than graduates in terms of societal costs. The false negatives in this case are the more expensive mistake. Let c_0 be the cost of misclassifying a true 0 case and c_1 be the cost of misclassifying a 1 case. For a two class classification problem our classifier, $f(\mathbf{x})$, predicts values 0 or 1. Then the expected cost is

$$\begin{aligned} J(f) &= E_{\mathbf{x}} E_{y|\mathbf{x}} c_0 I(y=0) f(\mathbf{x}) + c_1 I(y=1)(1 - f(\mathbf{x})) \\ &= E_{\mathbf{x}} c_0 (1 - P(y=1|\mathbf{x})) f(\mathbf{x}) + c_1 P(y=1|\mathbf{x})(1 - f(\mathbf{x})) \end{aligned} \quad (4)$$

where I(·) is the indicator function that is 1 if the expression is true and 0 otherwise. Minimizing the expression pointwise at each $\mathbf{x}$ we see that ideally $f(\mathbf{x})$ should equal 0 whenever $c_0(1 - P(y = 1|\mathbf{x})) > c_1 P(y = 1|\mathbf{x})$. Equivalently, the best classifier is

$$f(x) = \begin{cases} 0 & \text{if } P(y=1|\mathbf{x}) < c_0/(c_0 + c_1) \\ 1 & \text{otherwise} \end{cases} \quad (5)$$

We actually do not need an estimate of $P(y = 1|\mathbf{x})$ in order to obtain a good decision rule. It is sufficient to have a method that determines which side of $c_0/(c_0+c_1)$ the probability would fall. In fact, some excellent classifiers produce poor estimates of $P(y = 1|\mathbf{x})$. Note that if $c_0 = 1$ and $c_1 = 20$, as in the high school

dropout problem, then any student with a dropout probability exceeding 0.047 needs special attention.

Although many classification methods advertise their ability to obtain low misclassification costs, many classification procedures minimize cost indirectly. Classification trees are among the few procedures that directly aim to minimize cost. Many other procedures aim for good estimates of $P(y = 1|\mathbf{x})$, which (5) as previously mentioned shows is sufficient but not necessary for developing a decision rule for any choice for c_0 and c_1. In some settings a probabilistic prediction itself is necessary to have a complete risk assessment.

The likelihood principle, studied in detail in Berger and Wolpert (1984), implies that any inference about parameters of interest should depend on the data only through the likelihood function, the probability that the model would generate the observed data. So while (4) is the loss function for minimizing misclassification cost, when seeking good probability estimates for the two-class classification problem we should look to the *Bernoulli likelihood*,

$$L(p) = \prod_{i=1}^{N} p(\mathbf{x}_i)^{y_i} (1 - p(\mathbf{x}_i))^{1-y_i} \tag{6}$$

where $p(\mathbf{x}) = P(y = 1|\mathbf{x})$ and is what we want to estimate and study. While up to this point we have used $f(\mathbf{x})$ to denote the prediction model that we are trying to estimate, here we use $p(\mathbf{x})$ to remind ourselves that it is a probability and must be on the interval [0,1]. Many statistical procedures are based on estimates of $p(\mathbf{x})$ that maximize the likelihood, intuitively the $p(\mathbf{x})$ that makes the observed data most likely. While before we discussed finding $f(\mathbf{x})$ to minimize a loss function, here the goal is to find a $p(\mathbf{x})$ to maximize a likelihood. The log-likelihood is the more commonly used form of this loss function and, again, we are not simply interested in maximizing it for our finite sample but in expectation over a new observation drawn from the same distribution that generated our dataset.

$$\begin{aligned} J(p) &= E_{y,\mathbf{x}} \log L(p) \\ &= E_{y,\mathbf{x}} \left[y \log p(\mathbf{x}) + (1 - y) \log(1 - p(\mathbf{x})) \right] \end{aligned} \tag{7}$$

We will see in section 4.2 that logistic regression procedures are based on *maximum likelihood* estimates of $p(\mathbf{x})$.

The Bernoulli log-likelihood naturally extends to multiclass classification via the *multinomial log-likelihood*,

$$E_{y,\mathbf{x}} L(p_1, p_2, \ldots, p_K) = E_{y,\mathbf{x}} I(y_i = k) \log p_k(\mathbf{x}_i), \tag{8}$$

where $p_k(\mathbf{x}) = P(y = k \mid \mathbf{x})$ and the $p_k(\mathbf{x})$ sum to 1. With this loss function we will seek k functions, each of which estimates one of the class probabilities. Also, *ordinal regression* methods are available when the K classes are ordered as in preference rating scales.

Using the Bernoulli log-likelihood as a loss function focuses on obtaining good probability estimates but it is unclear what "good" means in this context. Meteorologists especially have been studying accurate probability assessments, decomposing prediction accuracy into discrimination and calibration components. Discrimination, which generally gets the most attention, is the ability to separate the classes while calibration is the ability to assign meaningful probabilities to events. When we are dealing with a binary outcome (y is either 0 or 1) the Brier score (Brier 1950) shown in (9) offers an interesting assessment of probabilistic assignments.

$$J(p) = E_{y,\mathbf{x}} (y - p(\mathbf{x}))^2 \tag{9}$$

The Brier score is small when our probability estimate is small and $y = 0$ and when our probability estimate is near 1 when in fact $y = 1$. Clearly the Brier score is minimized when we have perfect forecasts. Yates (1982) discusses the *Murphy decomposition* of the empirical Brier score exposing different aspects of prediction quality.

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - p(\mathbf{x}_i))^2 = \bar{y}(1-\bar{y}) - \frac{1}{N}\sum_{k=1}^{K} n_k(\bar{y}_k - \bar{y})^2 + \frac{1}{N}\sum_{k=1}^{K} n_k(p_k - \bar{y}_k)^2 \tag{10}$$

= uncontrollable variation + resolution + calibration

The first term is the variance of the outcome. It is only small when the y_i's are all 0 or all 1, something over which we have no control. The variance term also represents the best we can do. In the second term the n_k represents the number of observations that are given the probabilistic prediction p_k and $\bar{y}_k$ is the average of the outcomes given the prediction p_k. This *resolution* term is large (very negative) when we are able to discriminate the 0 outcomes from the 1s. In that situation the average outcome given prediction p_k is far from the baseline rate, near 0 or 1. The last term measures *calibration*, the ability to assign meaningful probabilities to the outcomes.

To understand calibration let us again turn to the high school dropout problem. Assume we have a probabilistic prediction model. A new set of students arrives

on which we assess the dropout risk. If the model is well calibrated then among the collection of students to which the model assigned a dropout probability of, for example, 0.3, 30% would actually dropout. Figure 1 shows a smoothed calibration plot for a boosted classification model (section 5.6) for the high school dropout example. The 45º line is the perfectly calibrated predictor. The tick marks along the bottom of the figure mark the deciles of the estimated probabilities, most of which are below 0.1. Note that for all the students with an estimated dropout probability around 0.3, 36% of them actually dropped out.

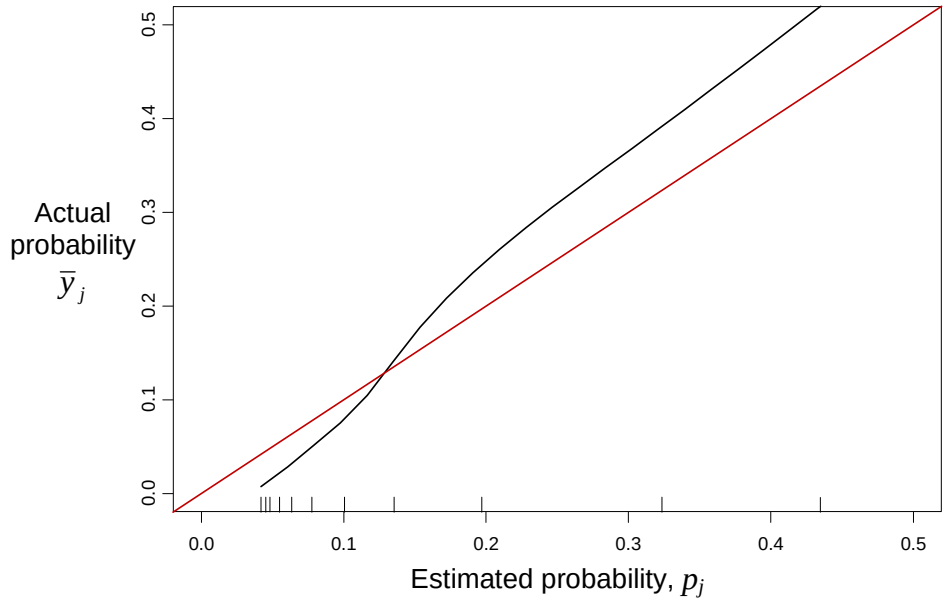
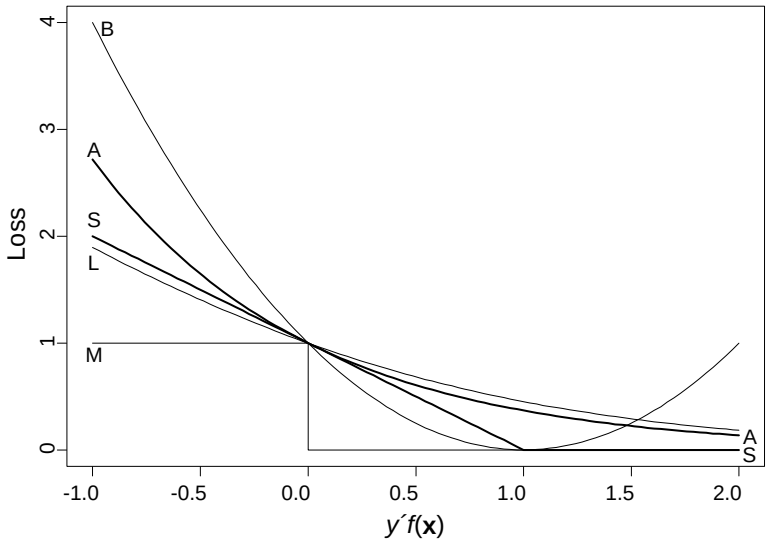


Figure 1: Calibration plot for the high school dropout example

We learn from the Murphy decomposition that when measuring classification performance with the Brier score that both discrimination *and* calibration are important. In practice we will have to decide which attributes are most important for the problem at hand. Good classification accuracy alone at times may be insufficient.

Classification is conceptually quite simple. We want to put the right labels on observations. But that conceptual simplicity is not easily translated into a loss function without additional information on the goals we seek. Choosing to minimize cost or opting to go for the best probability estimates leads to different choices for loss functions and, therefore, prediction methods. One of the early strategic steps in a classification problem is defining the goals and usually that translates into a natural choice for the loss function.

So far we have presented three candidates for the loss function but others have been proposed. Figure 2 shows the similarities and differences among the various classification loss functions. First consider relabeling the two classes as –1 and 1, $y' = \frac{1}{2}(y+1)$, which simplifies the comparison. Therefore, when $y'f(\mathbf{x})$ is positive the observation is correctly classified. The curve labeled M in Figure 2 reflects this. Note that all of the loss functions are bounds for the misclassification loss function. Also shown in bolder lines are the loss functions for support vector machines (section 5.5) and AdaBoost (section 5.6). The different loss functions determine how much we penalize our predictor for certain mistakes. The Brier score strongly penalizes mistakes but is the only loss function that also penalizes overconfidence in predictions, indicated by the upswing in the Brier loss for $y'f(\mathbf{x}) > 1$.



Brier score

$E(yf(\mathbf{x})-1)^2$

AdaBoost

$E\exp(-yf(\mathbf{x}))$

SVM

$E\max(0,1-yf(\mathbf{x}))$

Logistic

$E\log(1+e^{-yf(\mathbf{x})})$

Misclassification

$E\ I(yf(\mathbf{x})<0)$

Figure 2: Classification loss functions

3.1.3 Cox loss function for survival data

The last specific loss function discussed here is often used for survival data. We include it here partially for those interested in lifetime data but also to demonstrate creative developments of loss functions for specific problems. Survival analysis continues to be a thriving area in biostatistics and this section focuses on a small part, the Cox model for *proportional hazards regression.*

For each observation we observe the subject features, $\mathbf{x}_i$, the time the subject was last observed, t_i, and a 0/1 indicator whether the subject failed at time t_i, δ_i. If we know the exact failure time for all observations, that is $\delta_i = 1$ for all i, then we can

turn to some of the standard loss functions from section 3.1.1. When we have many observations that have not failed, such as customers who have not switched their long distance carrier *yet*, we would lose information if we did not include these observations in the estimation process. To make this more concrete, customer A in South Dakota with no previously filed complaints has used long distance company X for 100 days and remains a customer. The observation for this customer would be

($\mathbf{x}$ = {State=SD, Complaint=0}, $\delta = 0$, $t = 100$).

Since customer A remains a customer at day 100 they have not yet "failed" and so $\delta = 0$. Customer B, on the other hand, is from North Dakota, has filed 3 complaints, and on day 65 of service switches long distance carriers. The observation record for this customer would be

($\mathbf{x}$ = {State=ND, Complaint=3}, $\delta = 1$, $t = 65$).

So from a dataset of observations ($\mathbf{x}_i$, δ_i, t_i), for i in 1, …, N, we want to construct a model to predict *actual* failure time t_i from $\mathbf{x}_i$.

The proportional hazards model assumes that the *hazard function*, the instantaneous probability that a subject with feature $\mathbf{x}$ fails in the next small interval of time given survival up to time t, is

$$h(t, \mathbf{x}) = \lambda(t) \exp(f(\mathbf{x})) \tag{11}$$

where $\lambda(t)$ is the baseline hazard. This model assumes that the relevance of a particular feature does not change over time. If we can estimate $f(\mathbf{x})$ then we can determine those indicators that accelerate the rate of failure and those observations that have an accelerated risk of failure. Given that an observation in the dataset with N observations failed at time t' the probability that it was observation i is

$$\frac{I(t_i \geq t')\lambda(t')\exp(f(\mathbf{x}_i))}{\sum_{j=1}^{N} I(t_j \geq t')\lambda(t')\exp(f(\mathbf{x}_j))}. \tag{12}$$

Conveniently, the baseline hazard, $\lambda(t)$, cancels in (12). We can then write down the likelihood that the first observed failure would have failed at its failure time and the second observed failure would have failed at its failure time and so on.

$$\prod_{i=1}^{N}\left[\frac{\exp(f(\mathbf{x}_i))}{\sum_{j=1}^{N} I(t_j \geq t_i)\exp(f(\mathbf{x}_j))}\right]^{\delta_i} \tag{13}$$

If we can find an $f(\mathbf{x})$ that makes the *Cox partial likelihood* (13) large, this indicates that $f(\mathbf{x})$ can put the N observations approximately in order of when they will fail. Subsequently we can use this $f(\mathbf{x})$ to determine which subjects that have not failed yet are most at risk of failure in the near future. Even though we do not know the exact form of the baseline hazard $\lambda(t)$ we are still able to estimate $f(\mathbf{x})$ using only the order of failure times, rather than the actual failure times themselves. We can then estimate $\lambda(t)$ although $f(\mathbf{x})$ is sufficient for identifying observations that are prone to shorter times to failure.

Now that we have a handful of loss functions for measuring predictive performance, the next section begins a discussion of finding the ideal f to optimize our selected loss function.

4 Linear models

In this section we will begin looking at prediction models that have a linear structure. Although the structure at first may seem naïve, the development is an important one. First, these methods have a long history and are still in widespread use. Second, although apparently simplistic these methods can perform particularly well when the data is sparse. And third, several of the most modern methods build upon the basic linear structure and share some of the model building tools.

4.1 Linear Regression

Section 3.1.1 introduced the squared error loss function. Now let us restrict the model to have the form

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_d x_d = \beta'\mathbf{x} \tag{14}$$

where d+1 is the dimension of the feature vector $\mathbf{x}$ and the first element of $\mathbf{x}$ is 1. Rather than having a completely unspecified $f(\mathbf{x})$ we now only have d+1 model parameters to estimate. The coefficient, β_j, represents a difference between two subjects that have the same feature vector except differ by 1 unit on feature x_j. When fitting linear regression models with a categorical feature with k levels we can create k – 1 0/1 *dummy variables*. If variable x_1 has k=3 levels then we can fit the model

$$f(\mathbf{x}) = \beta_0 + \beta_{11} I(x_1 = 1) + \beta_{12} I(x_1 = 2) + \beta_2 x_2 + \ldots + \beta_d x_d \,. \tag{15}$$

Now β_{11} represents a difference between category 1 subjects and category 3 subjects.

To fit the model we simply select the vector of coefficients to minimize the empirical squared error loss

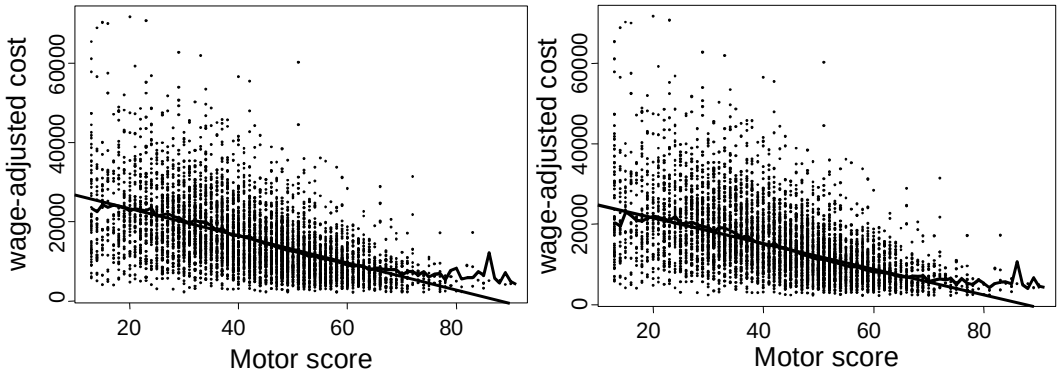
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} (y_i - \beta' \mathbf{x}_i)^2 = \arg\min_{\beta} \frac{1}{N} (\mathbf{y} - \mathbf{X}\beta)'(\mathbf{y} - \mathbf{X}\beta) \tag{16}$$

where $\mathbf{X}$ is the $N \times d$+1 feature matrix and $\mathbf{y}$ is a column vector with the N outcomes. The solution to (16) is

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}' y \tag{17}$$

solvable in a single pass through the dataset. Figure 3(a) shows the result of a linear least squares fit to stroke data from 1996. The jagged curve in the plot is the average cost at each motor score. In this simple example we have enough data at most values of the motor score to get reasonable estimates. The line generally runs through the pointwise averages and shows considerably less instability for the sparsely sampled stroke patients with high motor scores. The β that minimizes squared error loss is (30251.0, –342.6) and the resulting average squared error is 5.88×10^7.

Although this choice of β minimizes squared error for the 1996 data we would hope that such a model holds up over time. If we use the model fit to 1996 data to predict costs for the 1997, 1998, and 1999 data the average squared error for those years is 5.94×10^7, 5.90×10^7, and 6.19×10^7. In general the error on the training data will be an overoptimistic estimate of the performance on future observations.



(a) (b)

Figure 3: Predicting cost of stroke rehabilitation from motor ability score. Least squares linear fit and the pointwise average fit (a) and the least absolute deviations linear fit and the pointwise median fit (b).

Medical costs are particularly prone to extreme observations. Modeling the median rather than the mean offers a more stable model. The minimizer of absolute error does not have a convenient closed form expression, however, moderately efficient algorithms do exist (Bloomfield and Steiger, 1983). Figure 3(b) displays the linear model that minimizes absolute prediction error along with pointwise median estimates that are well approximated by the line. The β that minimizes average absolute loss is (28014.0, –320.2) producing a linear fit with a gentler slope than one obtained using the squared error loss.

4.2 Classification

In this section we will focus on procedures for 0/1 classification problems but will make brief mention of the multiple class case at the end.

4.2.1 Linear logistic regression

Statistical methods for binary classification are usually designed to produce good estimates of $p(\mathbf{x}) = P(Y = 1|\mathbf{x})$ using the Bernoulli likelihood shown in (6). Linear logistic regression, one of the earliest techniques, assumes a particular parametric form for $p(\mathbf{x})$,

$$p(\mathbf{x}) = \frac{1}{1+\exp(-f(\mathbf{x}))}, \text{ where } f(\mathbf{x}) = \beta'\mathbf{x}. \tag{18}$$

As in linear regression, the prediction depends on the feature vector only through a linear combination of the components of **x**. Again this greatly reduces the

complexity of the model by reducing the problem to estimating only the d+1 β's. The logistic function transforms the linear combination from the whole real line to the [0, 1] interval. Rewriting (18) we can see that this model assumes that the log-odds are a linear function of $\mathbf{x}$.

$$\log \frac{p(\mathbf{x})}{1-p(\mathbf{x})} = \beta' \mathbf{x}. \tag{19}$$

Inserting (18) into the Bernoulli likelihood from (6) and taking the logarithm we obtain

$$L(\beta) = \sum_{i=1}^{N} y_i \beta' \mathbf{x}_i - \log(1+\exp(\beta' \mathbf{x}_i)). \tag{20}$$

To fit the model by maximum likelihood we select the β that maximizes (20). No closed form solution exists but we can utilize a simple Newton-Raphson procedure to numerically maximize the likelihood. The first and second derivatives of (20) are

$$\frac{\partial L(\beta)}{\partial \beta} = \sum_{i=1}^{N} \mathbf{x}_i \left(y_i - \frac{1}{1+\exp(\beta' \mathbf{x}_i)} \right) = \mathbf{X}'(\mathbf{y}-\mathbf{p}) \tag{21}$$

$$\frac{\partial^2 L(\beta)}{\partial \beta^2} = -\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i' \frac{1}{1+\exp(\beta' \mathbf{x}_i)} \left(1 - \frac{1}{1+\exp(\beta' \mathbf{x}_i)} \right) = -\mathbf{X}'\mathbf{W}\mathbf{X} \tag{22}$$

where $\mathbf{p}$ is the vector of predicted probabilities and $\mathbf{W}$ is a diagonal matrix with diagonal equal to $p(\mathbf{x}_i)(1 - p(\mathbf{x}_i))$. After selecting an initial starting value for β, the Newton-Raphson update is

$$\begin{aligned} \beta &\leftarrow \beta - (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1} \mathbf{X}'(\mathbf{y}-\mathbf{p}) \\ &= (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1} \mathbf{X}'\mathbf{W}(\mathbf{X}\beta - \mathbf{W}^{-1}(\mathbf{y}-\mathbf{p})) \\ &= (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1} \mathbf{X}'\mathbf{W}\mathbf{z}. \end{aligned} \tag{23}$$

Note the similarity between (23) and the solution to the linear regression model in (17). Rather than simply having the $\mathbf{y}$ as in (17) we have a *working response* $\mathbf{z}$ and we have the weight matrix $\mathbf{W}$. The Newton update, therefore, is a weighted linear regression with observation i having weight $p_i(1 - p_i)$ where the features $\mathbf{x}_i$ predict the working response z_i. This algorithm is known as *iteratively reweighted least squares* (IRLS). In practice, convergence usually occurs after 3 to 4 iterations of IRLS. We can also begin to think about improvements including non-linear

predictors in (19) or using non-linear regression inside the IRLS algorithm. We will visit these issues shortly. Note also that the weight is largest when p_i is close to ½, close to the equal cost decision boundary, an issue that will arise again in our discussion of boosting.

Figure 4 shows a linear logistic regression model fit to the high school dropout dataset with two predictors. As expected the model structure forces the contours to be parallel. From the predicted probabilities we can apply our decision rule and estimate the expected cost per student. As before, assuming that failing to detect a dropout is 20 times more costly than failing to identify a graduate the decision boundary is $p = 0.047$, marked by the upper contour in Figure 4. All students above that line are classified as graduates and those below that line are classified as dropouts. Clearly this model puts the majority of cases as dropouts. Had we assumed everyone would graduate our expected cost would be 3.3. Assuming everyone is a dropout would cost us 0.83. But using the information in the two predictors our expected costs are reduced slightly to 0.81. While this is a modest decrease perhaps additional predictors may further reduce misclassification costs. On the other hand, perhaps the rigidity of the linear logistic model will prevent us from identifying the characteristics of the dropouts.

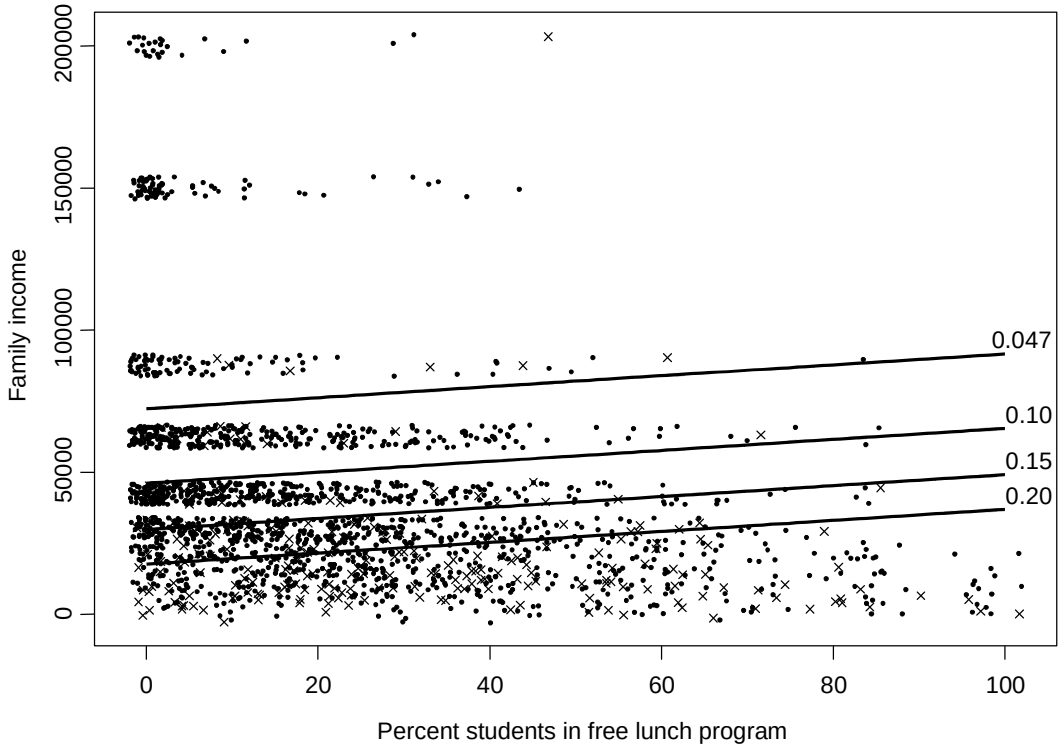


Figure 4: Probability of dropout predicted from family income and percentage of students at school in a free lunch program. The four lines indicate contours of equal probability of dropping out. The ×'s mark the dropouts and the •'s mark the graduates.

4.2.2 The naïve Bayes classifier

The naïve Bayes classifier (Hand and Yu, 2001) is, in spite of its name, a very powerful classifier. It is simple to program, fit to data, and is easy to interpret.

If Y is the class variable that we would like predict using predictors $\mathbf{x}$, then the naïve Bayes classifier has the form

$$P(Y = y \mid \mathbf{x}) \propto P(Y = y)P(\mathbf{x} \mid Y = y) \tag{24}$$

$$= P(Y = y)\prod_{j=1}^{d} P(x_j \mid Y = y) \tag{25}$$

Equation (24) is a direct application of Bayes Theorem. To transition to (25) requires a naïve assumption, that given the true class the features are independent.

For example, given that a particular child dropped out, knowing that they had poor grades gives no additional information about their socioeconomic status. This implies that dropout status is sufficient information to estimate any of the child's features. The chapter on Bayesian methods diagrams the naïve Bayes classifier as a graphical model. Refer to the figure there to see this graphically.

The naïve Bayes assumption gives the naïve Bayes classifier the same structural form as linear logistic regression described in the previous section. Letting $p(\mathbf{x}) = \mathrm{P}(Y = 1 \mid \mathbf{x})$ we see that the naïve Bayes classifier is additive on the log odds, or logit, scale.

$$\begin{aligned} \log \frac{p(\mathbf{x})}{1-p(\mathbf{x})} &= \log \frac{P(Y=1)}{P(Y=0)} + \sum_{j=1}^{d} \log \frac{P(x_j \mid Y=1)}{P(x_j \mid Y=0)} \\ &= w_0 + w_1(x_1) + w_2(x_2) + \ldots + w_d(x_d) \end{aligned} \tag{26}$$

Note that the structural form is similar to the form used for linear logistic regression in (19). If all of the x_j are categorical then the functional forms are exactly the same. Standard practice has been to discretize the continuous x_j's, creating histogram estimates of $P(x_j \mid Y)$. This creates an additive model where the w_j components are step functions rather than linear functions of x_j.

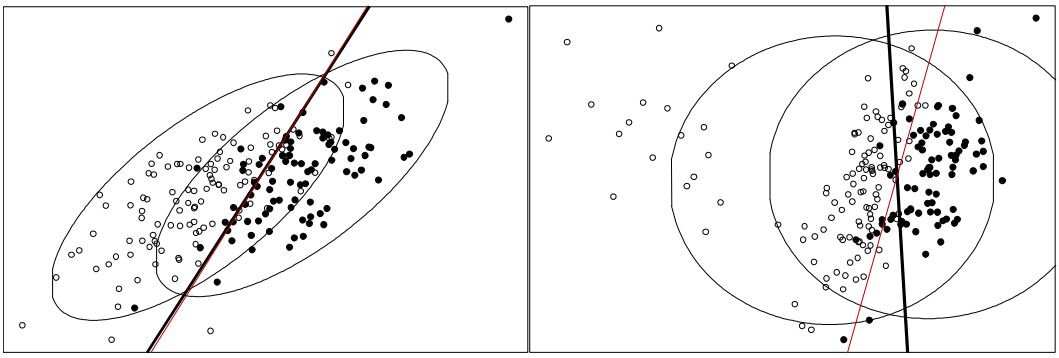
Logistic regression assumes that $P(Y = 1|\mathbf{x}) = P(Y = 0|\mathbf{x})\exp(\beta'\mathbf{x})$ where the $P(Y = 0|\mathbf{x})$ can have an arbitrary form. In some respects, naïve Bayes has a more restrictive assumption, specifying a distributional form for both classes,

The estimation procedure separately estimates the components for $y = 0$ and $y = 1$ before combining into a single classifier. This assumption has some advantages for large dataset applications. The assumption allows us to estimate this model in a single pass through the dataset and missing x_j values can be ignored. In the next section we will look at a third linear classifier with yet another set of associated assumptions.

4.2.3 Linear discriminant analysis

Fisher (1936) proposed linear discriminant analysis (LDA) for classification problems. As with the naïve Bayes classifier, LDA uses Bayes theorem to reverse the conditional probability (24) and makes an assumption about the distribution of the features with in each class. Rather than assume conditional independence as in (25), LDA assumes that $P(\mathbf{x} \mid Y = y)$ is a multivariate normal density where all the classes share a common covariance matrix. With these two assumptions the log-odds again has a form that is linear in $\mathbf{x}$. Unlike logistic regression and naïve Bayes, LDA is very sensitive to outliers in $\mathbf{x}$ and in general performs quite poorly. Figure 5 shows an LDA fit to simulated data. When the features are truly

multivariate normal as in Figure 5(a) both LDA and logistic regression produce approximately the same decision boundary. When one class is contaminated with outliers the decision boundary can move substantially and perform worse than logistic regression Figure 5(b).



(a) (b)

Figure 5: LDA decision boundaries where **x** is multivariate normal with each class (a) and when one class is contaminated with outliers (b). The heavy line is the LDA boundary and the lighter line is the logistic regression boundary.

In the last 70 years discriminant analysis, like other methods, has undergone considerable modernization. Although the simple form described here tends to perform poorly, LDA's descendants can perform quite well. They generally relax the normal distribution assumption and allow the classes to have separate distributions each of which we can model with a more flexible density estimator. See Hastie et al (1994) and Ridgeway (2002) for examples of these extensions.

4.3 Generalized linear model

In this section we briefly mention the connection to a large class of regression models that one can understand as variations on the linear logistic regression development. Section 4.2.1 showed a particular transformation of $f(\mathbf{x})$ onto the probability scale using the logistic transform (18) and then used the Bernoulli likelihood to determine the best fitting linear model. Statisticians often prefer the logistic transform as the *link function*, the function relating the probability to $f(\mathbf{x})$, mostly because it allows interpretation of model coefficients as log odds-ratios. Economists, on the other hand, have tended to use *probit regression* differing from logistic regression by its use of the inverse Gaussian cumulative distribution as the link function, $p(\mathbf{x}) = \Phi^{-1}(f(\mathbf{x}))$. There is considerable flexibility in choosing the link function although the logit and probit are by far the most common.

Besides the choice of link function for logistic regression, we can vary the likelihood itself to capture an enormous class of useful prediction problems. The Bernoulli distribution is particular to 0/1 outcomes. The *Poisson distribution* is often used to model outcomes involving counts (items purchased, cigarettes smoked, etc.) and has the form

$$P(Y = y \mid \mathbf{x}) = \frac{\lambda(\mathbf{x})^y \exp(-\lambda(\mathbf{x}))}{y!} \tag{27}$$

where $\lambda(\mathbf{x})$ represents the expected count for an observation with features $\mathbf{x}$. Oftentimes an observation will have a time measurement in addition, such as time as a customer or time since released from treatment. In such instances researchers commonly parameterize the Poisson model as

$$P(Y = y \mid \mathbf{x}, t) = \frac{(\lambda(\mathbf{x})t)^y \exp(-\lambda(\mathbf{x})t)}{y!} \tag{28}$$

so that $\lambda(\mathbf{x})$ represents a rate of occurrences. Using the log link function, $\log \lambda(\mathbf{x}) = \beta'\mathbf{x}$, we are assured that the rate will always be positive. For this reason Poisson regression is often referred to as log-linear modeling. If we have N independent observations $(y_i, \mathbf{x}_i, t_i)$ then we can write down the log-likelihood as

$$\log L(\beta) = \log \prod_{i=1}^{N} \frac{(\lambda(\mathbf{x}_i)t_i)^{y_i} \exp(-\lambda(\mathbf{x}_i)t_i)}{y_i!} \tag{29}$$

$$= \sum_{i=1}^{N} y_i \beta'\mathbf{x}_i - t_i \exp(\beta'\mathbf{x}_i) + y_i \log t_i - \log y_i! \tag{30}$$

Except for the last two terms that do not involve β, (30) closely resembles (20). Inside the sum in both cases we have the outcome, y, times the linear predictor minus a term which has the expected value of y as its derivative. Fitting the Poisson model also involves a few iterations of a simple IRLS algorithm.

The Bernoulli and Poisson prediction methods are special cases of the class of *generalized linear models* (GLM). After selecting the variables for the linear predictor, the distribution of the response, and a link function, the GLM framework packages together a likelihood based loss function and an IRLS algorithm for fitting the models. Even the linear least squares model from section 4.1 falls into this framework with a Gaussian distribution for the response and an identity link function. Simply replace the Poisson distribution in (29) with the

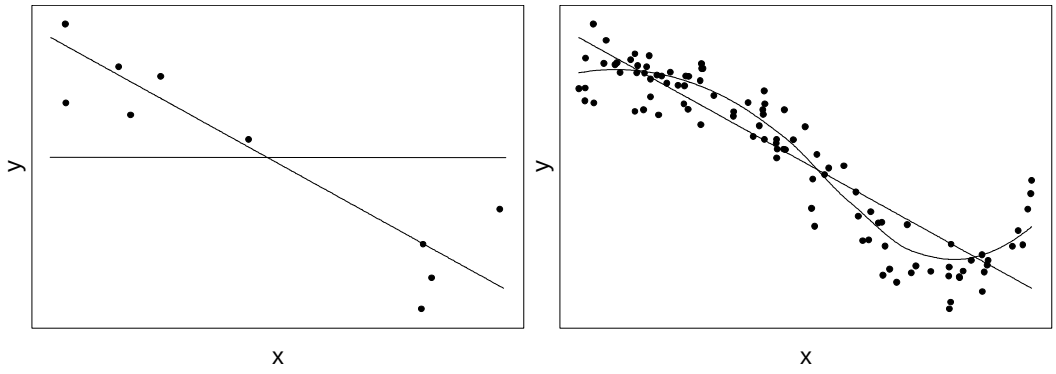
Gaussian distribution. Then setting the derivative of the log-likelihood equal to 0 and solving for β produces exactly the least squares solution we saw earlier.

Other useful GLMs include *multinomial logistic regression* for multiclass classification, *Gamma regression* for skewed outcome distributions (like cost), and *negative binomial regression* for count data with extra-Poisson variation.

McCullagh and Nelder (1989) provide a complete guide to the basics of the GLM. Greene (1999) also discusses these methods with respect to econometrics. Although the development of the GLM in this section is brief, this overview should give the impression that one has considerable flexibility in determining how to model the outcome variable. For continuous, binary, and count outcomes the GLM framework is one good starting place. The linear part is easily replaceable with any other functional form as we will see in the next section.

5 Non-linear models

In spite of their computational simplicity, stability, and interpretability, linear models have an obvious potential weakness. The actual process may not be linear and such an assumption introduces uncorrectable bias into the predictions. When data is sparse or the dimension of **x** is large, linear models often capture much of the information in the data as shown in Figure 6(a). There the linear model seems to capture much of the information. Detecting non-linear features requires more data with a low signal-to-noise ratio. Data mining applications inherently involve large datasets and so the general trend is almost always to use non-linear methods, implying that most data miners feel that their data is closer to the situation in Figure 6(b). Although the same mechanism generated both datasets, the increase in data in Figure 6(b) makes the linear model less appealing. Problems with a large number of features require caution. Such situations will more closely resemble Figure 6(a). Non-linear models run a much greater risk of being overfit to the training dataset and the chapter on performance analysis and evaluation in this handbook requires careful attention.



(a) (b)

Figure 6: Utility of linear and non-linear models

This section explores a few of the popular non-linear prediction methods. These methods generalize the previous discussion by allowing $f(\mathbf{x})$ to take on a more flexible form.

5.1 Nearest neighbor and kernel methods

The *k* nearest neighbor (knn) prediction model simply stores the entire dataset. As the name implies, to predict for a new observation the predictor finds the *k* observations in the training data with feature vectors close to the one for which we wish to predict the outcome. The prediction depends on the loss function and in general is

$$f(\mathbf{x}) = \arg\min_{\theta} J_{N(\mathbf{x})}(\theta) \tag{31}$$

where θ is a constant and $J_{N(k,\,\mathbf{x})}$ represents the loss function computed for only the *k* closest observations in a neighborhood near **x**. For example, the knn predictor for squared error loss is

$$f(\mathbf{x}) = \arg\min_{\theta} \frac{1}{k} \sum_{i \in N(k,\mathbf{x})} (y_i - \theta)^2 = \frac{1}{k} \sum_{i \in N(k,\mathbf{x})} y_i \tag{32}$$

the average of the outcomes for the *k* observations nearest to **x**. The knn classifier works similarly. It collects the *k* nearest observations and predicts the class that minimizes cost, the most popular class in the case of equal misclassification costs. Although the method may seem naïve it is often competitive with other, more sophisticated prediction methods. Figure 7 shows the knn classifier predicting high school dropout probability from family income and where *k* = 100. The

features were rescaled for the distance calculations so that they both had unit variance. The heavy contour line marks the decision boundary between predicting a dropout and predicting a graduate. Only the students from the wealthier families in schools with few students on a free lunch program will not be classified as a dropout risk.

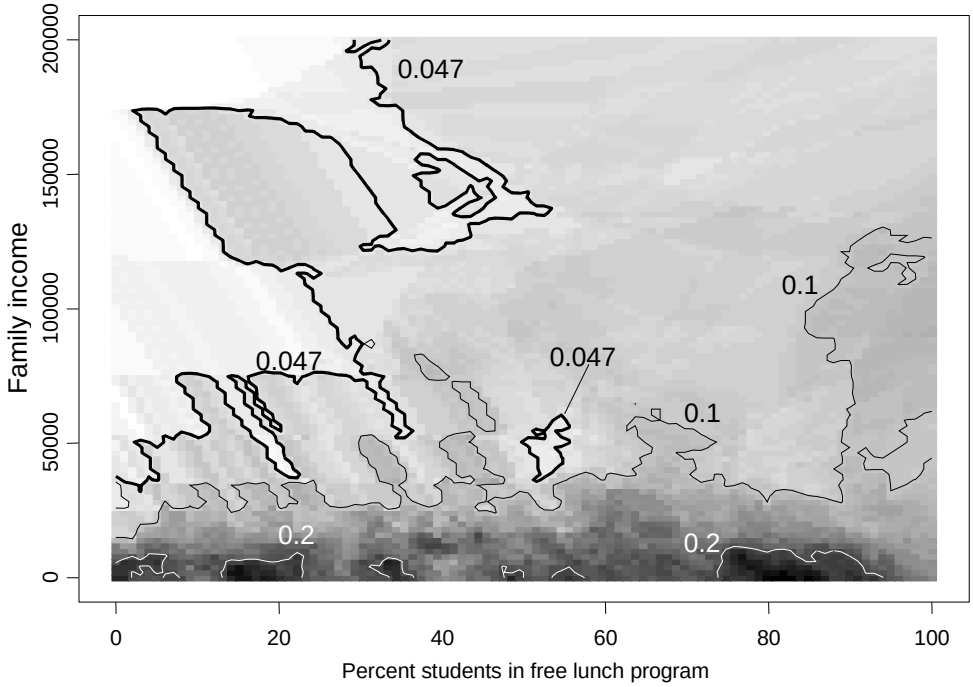


Figure 7: The 100 nearest neighbor classifier for the high school dropout data. The darker regions of the figure represent greater dropout risk. The lightest regions of the figure indicate a near 0 dropout risk.

Where the linear model is rigid the knn predictor is extremely flexible as Figure 7 clearly demonstrates. Compare Figure 7 to Figure 4. That flexibility can be abused by allowing *k* to be too small. Recalling the discussion from section 3.1.2, knn tends to offer poor probability estimates but nevertheless tends to be quite good at minimizing misclassification cost. We can look at how different choices for *k* affect prospective predictive performance as shown in Figure 8. Assuming that failure to identify a dropout is 20 times more costly than failure to identify a graduate, we can compute the average cost per student for our decision rule. Minimizing the expected cost heavily depends on careful selection of *k.* If we classify every student as a graduate the expected cost is 3.3, about what we see with the 1-nearest neighbor classifier. Classifying all students as dropouts, the decision rule when *k* gets very large, produces an expected cost of 0.83 shown as

the horizontal line in Figure 8. The minimum expected cost, 0.75, occurs when $k = 90$. The 90 nearest neighbor classifier puts 83% of the students at a greater than 4.7% chance of dropout. The linear logistics regression model has an expected cost of 0.81 and classified 90% of the students as dropouts.

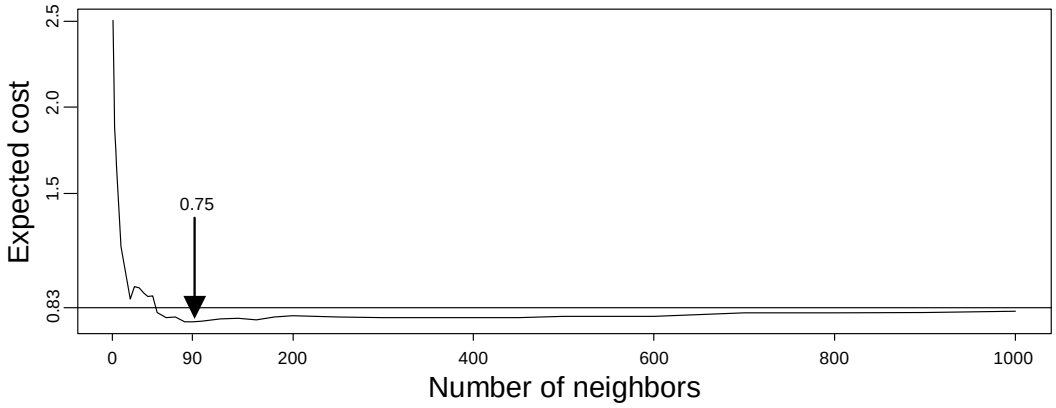


Figure 8: Predictive performance for different values of *k*. Expected cost uses probability of dropout exceeding 4.7% as the decision boundary

As N gets arbitrarily large and k grows at a certain rate (much slower than N) this predictor will converge to the true optimal predictor, a property known as *Bayes risk consistency*. However, the performance of the predictor for datasets of practical size depends heavily on k, the metric used to determine which observations are close, and the dimension of the feature vector.

A natural generalization of the knn predictor $f(\mathbf{x})$ involves having every observation contribute its outcome to the prediction weighted by its distance to $\mathbf{x}$. Returning again to squared error

$$f(\mathbf{x}) = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} K(\mathbf{x}_i, \mathbf{x})(y_i - \theta)^2 = \sum_{i=1}^{N} w_i y_i \Big/ \sum_{i=1}^{N} w_i \qquad (33)$$

a weighted average of the outcomes where $w_i = K(\mathbf{x}_i, \mathbf{x})$, a function that decreases as $\mathbf{x}_i$ moves further from $\mathbf{x}$. The knn predictor is a special case of (33), when $K(\mathbf{x}_i, \mathbf{x})$ takes value 0 or 1 depending on whether $\mathbf{x}_i$ is among the k closest observations. First considering the case with a single continuous predictor, let $K(x_i, x)$ be the Gaussian density

$$K(x_i, x) = \exp\left(-\tfrac{1}{2}((x_i - x)/\sigma)^2\right) \quad (34)$$

with mean equal to x and standard deviation σ, known as the *bandwidth* for the kernel regression model. Figure 9 shows stroke rehabilitation cost models using two kernel regression estimates with different bandwidth settings and the linear model from Figure 3. The cost axis is rescaled from Figure 3 to reveal details of the model differences. As the bandwidth gets small the kernel regressor resembles the pointwise average estimate shown in Figure 3 and exhibits a lot of variance in regions where there are fewer data points. The larger bandwidth is smoother and shows a lot of stability even in the extreme motor scores. Although all the methods align for the most common motor scores, the linear model reveals its *bias* in the extreme motor score values. In many prediction problems, data often show the presence of saturation effects (at some point additional improvements in motor ability do not decrease cost) and threshold effects (decreases in cost do not begin until motor exceeds some threshold). Note that if we only observe patients in motor score in the 30 to 60 range, the linear model would work extremely well and we would have little reason to consider other models. Other prediction methods can easily outperform linear models when a substantial portion of the dataset lies to the right of the saturation point and to the left of the threshold point.

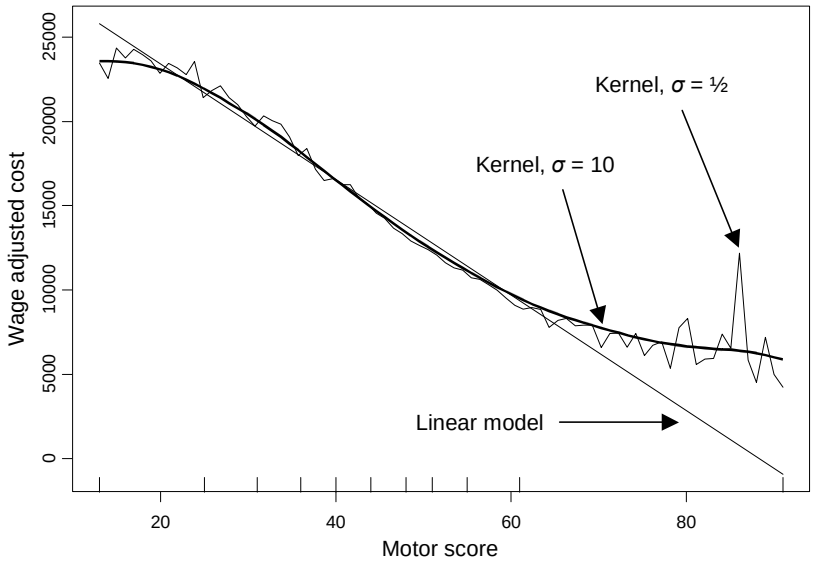


Figure 9: A kernel regression prediction model

Kernel regression methods generalize to multivariate feature vectors and the reader is referred to Hastie *et al* (2001) for more details. Support vector machines, discussed in section 5.5, are in fact a particular kind of kernel regression method.

The primary disadvantage of the knn predictor and naïve implementations of kernel methods for data mining applications is the need to store the entire dataset and the expense of searching for nearest neighbors or computing kernels in order to make a prediction for a single new observation. Some algorithms exist for trimming down the storage needs for these models. However, they still generally receive little attention as competitive data mining procedures in spite of their utility for non-linear modeling.

For the data miner interested in prediction methods, nearest neighbor and kernel methods have instructional value. The heuristic is that when predicting an outcome at **x** we borrow information from those points in the dataset that are close to **x**. We trust that the underlying function is fairly smooth so that nearness in terms of **x** implies similar outcomes. The linear models discussed in section 4 share this idea but assume the rigid functional form to interpolate between points. Remember that all prediction methods that are minimizing the same loss function differ only in how they interpolate between the points in the dataset. In some fashion they combine the outcomes from points in a neighborhood near **x**. How a method selects that neighborhood and how it combines the outcomes will determine its scalability, its interpretability, and its predictive performance.

5.2 Tree models

The chapter on tree models in this handbook extensively discusses the use of tree structured prediction methods. Tree structured predictors usually assume that $f(\mathbf{x})$ is a piecewise constant function where splits on the individual feature axes define the pieces. The terminal node of the tree defines the neighborhood and the constant that minimizes the loss function within the terminal node becomes the node prediction.

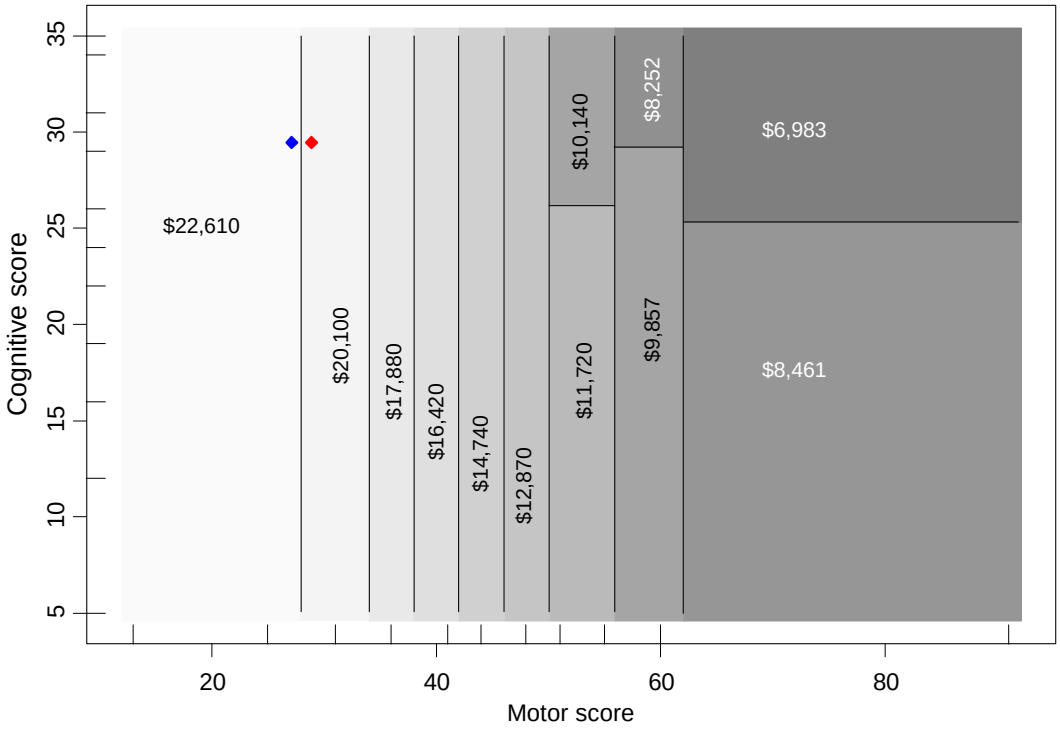


Figure 10: CART for cost of stroke rehabilitation. The two diamonds refer to the hypothetical patients discussed in the text. The inward ticks on the axes mark the deciles of the distribution of the motor and cognitive scales.

The advantages of tree predictors for data mining is that fast algorithms exist to construct them from data, prediction for new observations is quick, the method handles all types of input variables, and the model is storable in a compact form. This differs sharply from the nearest neighbor methods. However, as linear models can be criticized for their rigid functional form, the constant within-node prediction is rigid in a different way. Figure 10 shows how CART partitions stroke patients by motor and cognitive score into groups with relatively homogeneous cost. Note that in Figure 10 the model predicts that the patient with a motor score of 27 and the patient with a motor score of 28 (marked with diamonds in the figure) have costs that differ by $2,510, the difference in the predicted cost from each region. We really do not believe that cost varies so abruptly. Such biases around the partition edges can reduce prediction accuracy.

Like the nearest neighbor predictor, tree structured procedures are generally Bayes risk consistent (Gordon and Olshen, 1984). That is, as the dataset grows and the number of terminal nodes grows at a certain rate, the model converges to the minimizer of the population loss function. Although this is an interesting

mathematical fact it should be taken as a caution rather than a benefit. Trees can have a lot of variability in addition to edge biases.

Although trees are often appreciated for their apparent interpretability one should use caution due to their instability. To demonstrate we randomly split the high school dropout data into two parts and fit a CART classification tree to each. Figure 11 shows the resulting models. The two trees give very different perspectives concerning what is important in determining the risk of dropout. When several variables are correlated with each other and the outcome, the tree must select a single variable for the splitting variable, hiding the importance of other interesting inputs. Again, it is easy to be fooled by the trees transparent functional form and overlook important variables not included in the tree.

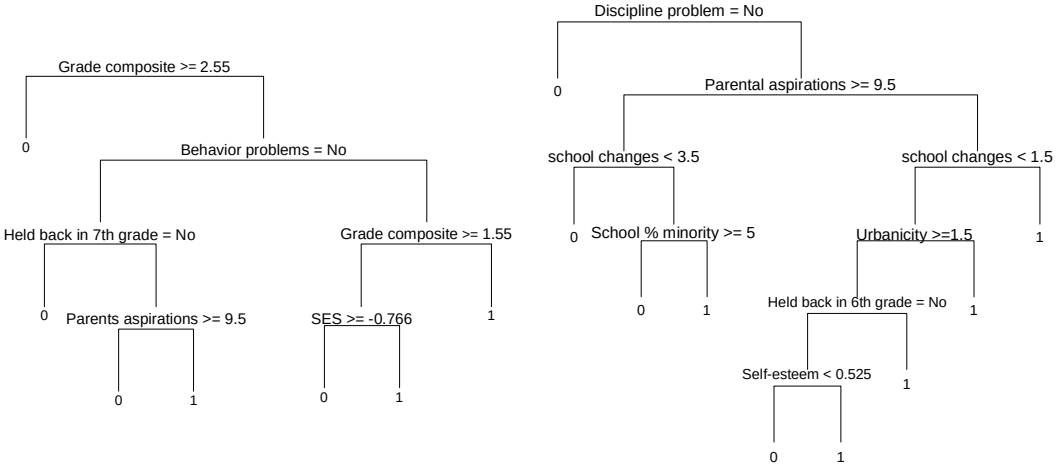


Figure 11: CART fit to two random splits of the high school dropout dataset.

5.3 Smoothing, basis expansions, and additive models

One disadvantage of tree models is that they produce discontinuous predictions and the main disadvantage of the linear models is that they enforce strict smoothness everywhere. This section considers methods between these two extremes.

Splines utilize piecewise polynomials (usually linear or cubic) to model $f(x)$. Remember that trees use piecewise constants and that linear models use a single linear equation. Splines are, therefore, one further step in this progression. As with trees, spline methods need to select split points, known as *knots* in spline terminology. Between the knots we fit a polynomial of our choice to minimize our loss function. *Natural cubic splines* are among the most popular. Between each knot these splines fit cubic polynomials but fit linear models in the range outside

the first and last knots. In addition they enforce continuity of $f(x)$, $f'(x)$, and $f''(x)$ at every knot. Figure 12 shows two examples of natural splines with differing numbers of knots. One can fit fairly flexible curves, including saturation and threshold effects previously mentioned, with only a handful of knots.

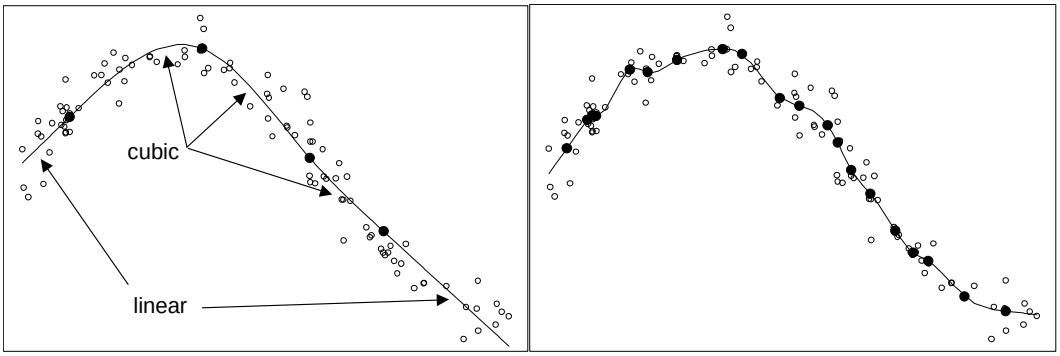


Figure 12: Natural splines with 4 and 20 knots on simulated data. The knots, shown as solid points, are placed at equally spaced quantiles of the x variable.

The motivation for splines given earlier is that they seem a natural generalization of piecewise constant and linear models. Natural splines also arise in a curious way from a redefined loss function. Rather than simply minimize squared prediction error, consider minimizing it subject to a penalty on the magnitude of the function's second derivative.

$$\hat{J}(f) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2 + \lambda\int_{-\infty}^{\infty} f''(x)^2\,dx \qquad (35)$$

If λ, known as the *smoothing parameter*, is 0 then the function can be arbitrarily jagged and interpolate the data. The best model when λ is large has $f''(x) = 0$ everywhere, the ordinary linear model. However, when λ takes on other, moderate values, the function that minimizes (35) smoothly fits the data. It turns out that the minimizer of (35) over a very rich class of functions is unique and is a *natural spline*! The minimizing natural spline has knots at every observed x_i but the coefficients of each cubic are further restricted by an amount that depends on λ. We can use cross-validation to select λ.

A discussion on the details for fitting natural splines comes later, but as discussed so far, they can be computationally expensive for use with massive datasets. An easy adaptation is to reduce the number of knots as originally shown. Rather than placing knots at every data point, 5 to 20 knots are likely sufficient for many univariate applications. A potential downside of splines is that they are *not* invariant to transformations in the x variable. If instead of using x we use $\log x$ as

the feature we will obtain a different natural spline for the same λ. While tree models are invariant to transformations in x, for smooth models one needs to think more carefully about the scale used for x. In section 5.6 we will see that boosted trees allow fairly smooth predictions but are also invariant to transformations of the features.

The above natural spline discussion focused on problems with a single feature. So far, for problems involving more than one feature we have considered only models that are linear in $\mathbf{x}$ or are local averages in neighborhoods of $\mathbf{x}$. *Basis expansions* represent yet another broad class of models. Assume that the best predictor $f(\mathbf{x})$ is well approximated by a sum of *basis functions*.

$$f(\mathbf{x}) = \sum_{k=1}^{K} \beta_k g_k(\mathbf{x}) \tag{36}$$

The basis functions in the collection, $g_k(\mathbf{x})$, are often simple functions that can be combined to describe more complex functions.

Many of the models we have previously discussed fall into this framework. Note that if $g_k(\mathbf{x}) = x_{k-1}$, with $x_0 = 1$, then (36) reduces to the linear model previously discussed. Trees can also be recast as a basis expansion where $g_k(\mathbf{x}) = I(\mathbf{x} \in N_k)$ where N_k is a defined by upper and lower limits on certain components of $\mathbf{x}$, such as the 12 partitions shown in Figure 10. Tree learning algorithms vary on how they select the number of basis functions and the form of N_k. One can naturally imagine other choices for the basis functions.

A univariate cubic spline with K knots $c_1, \ldots, c_K$ is decomposable into K+4 basis functions

$$g_1(x) = 1 \quad g_2(x) = x \quad g_3(x) = x^2 \quad g_4(x) = x^3 \quad g_{k+4}(x) = (x - c_k)_+^3 \tag{37}$$

where $(x)_+ = \max(0, x)$. The cubic spline maintains a cubic fit before c_1 and after c_K where the natural spline uses a linear fit. To fit these models we can utilize all the methods developed for fitting linear models. Note that if we have K=2 knots our predictor looks like

$$f(x) = \beta_1 + \beta_2 x + \beta_3 x^2 + \beta_4 x^3 + \beta_5 (x - c_1)_+^3 + \beta_6 (x - c_2)_+^3 \tag{38}$$

precisely a linear model with five features. With a little effort we can see that (38) is a continuous piecewise cubic function with continuous first and second derivatives at the knots. Since (38) is simply a linear model we can use (17) to fit the cubic spline.

Fitting cubic splines using the basis functions shown in (37) requires $O(NK^2+K^3)$ operations. Other sets of basis functions for cubic and natural splines are more efficient. In particular, *B-splines*, have computational properties that allow the models to be fit in $O(N \log N+K)$ operations.

General multivariate extensions to natural splines exist, known as *thin-plate splines*, but have not been developed to the computationally efficiency needs of data mining yet. *Additive models*, on the other hand, can take advantage of smoothing methods to create particularly powerful methods for exploratory analysis as well as prediction. Additive models use the basis functions $g_k(\mathbf{x}) = g_k(x_k)$ so that each basis function is a function of only one of the features. In this way we can generalize from the simple linear model while maintaining a fairly simple, stable model.

For the cost of rehabilitation data we can fit a regression model where the predictor has the form

$$f(\mathbf{x}) = \beta_0 + g_1(\text{motor}) + g_2(\text{cognitive}) \tag{39}$$

requiring that g_1 and g_2 are smooth functions of their arguments by modeling them as natural splines. Additive models are generally not Bayes risk consistent in most applications. That is, as the dataset grows they cannot capture true interaction effects. However, they can often outperform other methods in practice especially when the data is noisy. In addition plotting the $g_k(x_k)$ can give much insight into the contribution of the individual features to the outcome. Figure 13 shows the additive model fit to the cost of stroke rehabilitation data. Compare the contours with Figure 10. The marginal plots in the figure show the estimates of g_1 and g_2 and are scaled equivalently. From the marginals alone we can see that the motor score is really driving the cost. The cognitive score has some information and seems to indicate that costs are slightly lower for patients at the extremes of the cognitive scale.

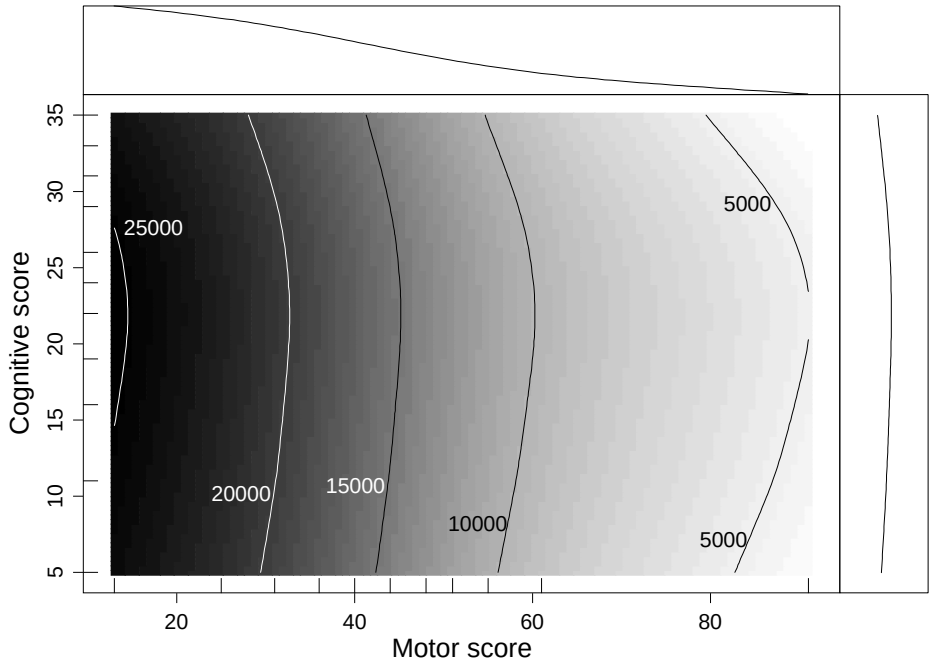


Figure 13: The estimates of g_1 and g_2 for cost of rehabilitation. The curves plotted in the margin are g_1(motor) and g_2(cognitive).

The *backfitting algorithm* is the common method of fitting additive models of this form. The constant β_0 is fixed at the mean of the y_i's and the iterative algorithm begins by fitting the univariate natural spline to predict y from motor score. Then the algorithm builds a natural spline to fit the cognitive score to the residuals left over from the motor score component. The motor score component is refit and the process iterates until convergence. This algorithm is fast as it generally involves only a few iterations, each of which uses an efficient B-spline implementation.

Hastie and Tibshirani (1990) provides more details on additive models including extensions to the GLMs from section 4.3. The interested reader might also look into multivariate adaptive regression splines (MARS) (Friedman, 1991). Just as CART implements a greedy search to construct a tree, MARS greedily constructs a model by introducing features into the prediction model (usually) through linear splines. It selects the variable and the spline transformation that offers the greatest decrease in prediction error through an exhaustive search similar to CART. The result is a modeling framework that allows some variables to enter as main effects (CART is only one big interaction term) and others to be interacted with one another. Many variations on this theme are possible.

5.4 Neural networks

While the neural network chapter in this handbook discusses these models in much greater detail, the brief discussion here casts them into the framework developed in this chapter. Like all the methods discussed so far there are variations in the functional form of the neural network predictors and the algorithms for fitting them. Generally, however, the single hidden layer neural network can be expressed in terms of a basis expansion

$$f(\mathbf{x}) = \sum_{k=1}^{K} \beta_k s(\alpha_k' \mathbf{x}) \tag{40}$$

where $s(z) = 1/(1+e^{-z})$, the logistic transformation. The K basis functions are known as the hidden units and the number of them is often fixed but may be selected by cross-validation.

For univariate classification problems neural networks model $P(Y = 1|\mathbf{x})$ like the logistic regression model (18). For M-class classification problems we create a multivariate outcome for each observation, $\mathbf{y}_i$, where the m^{th} value takes on the value 1 if the observation is in class m. Maintaining a common set of α_k's as in (40), we allow each dimension of $\mathbf{y}$ to have a different β_{km} and the prediction as

$$f_m(\mathbf{x}) = \sum_{k=1}^{K} \beta_{km} s(\alpha_k' \mathbf{x}) \quad \text{and} \quad P(Y = m \mid \mathbf{x}) = \frac{\exp f_m(\mathbf{x})}{\sum_{m'=1}^{M} \exp f_{m'}(\mathbf{x})}. \tag{41}$$

See the neural network chapter concerning methods for estimating the parameters of neural networks. Their inclusion in this section is to show that they share the same building blocks as all the other methods. Neural networks make specific choices for their basis functions but then can be used to minimize any loss function that we adopt for an application. The term *universal approximator* has been used for neural networks to mean that they can approximate any regression function. As the dataset grows and K is allowed to grow they are Bayes risk consistent (Barron, 1991). These are properties also shared by trees, nearest neighbors, and thin-plate splines. However, neural networks have the curious mathematical property that the rate at which the estimated predictor converges to the best predictor does not depend on the dimension number of features (Barron 1993).

5.5 Support vector machines

Several of the previous classification examples focused on obtaining good estimates of $P(Y = 1 \mid \mathbf{x})$ and then perhaps applying (5) to get a cost minimizing

decision rule. The support vector machine (SVM) in its basic form aims directly for the decision boundary.

We begin with an example of a linearly separable dataset shown in Figure 14. There are two continuous features and the class labels are $y = +1$ or $y = -1$ marked by clear and solid dots respectively. Many lines can separate the two classes. SVMs select the line that maximizes the *margin*, the space between the decision boundary and the closest points from each of the classes. With separable classes we can find a separating line having the form $\mathbf{w}'\mathbf{x} + b = 0$ such that

$$\mathbf{w}'\mathbf{x}_i + b \geq +1 \text{ when } y_i = +1 \tag{42}$$

$$\mathbf{w}'\mathbf{x}_i + b \leq -1 \text{ when } y_i = -1. \tag{43}$$

We can rescale $\mathbf{w}$ and b so that for some point(s) equality holds in (42) and in (43). A little geometry shows that the two planes defined by $\mathbf{w}'\mathbf{x}_i + b = 1$ and $\mathbf{w}'\mathbf{x}_i + b = -1$ are parallel, have no points between them, and are separated by a distance of d where

$$d^2 = 4 / \sum_{j=1}^{d} w_j^2 = 4 / \|\mathbf{w}\|^2. \tag{44}$$

Putting these steps together, fitting the SVM corresponds to maximizing the margin (44) subject to the constraints (42) and (43) solvable as a quadratic optimization problem.

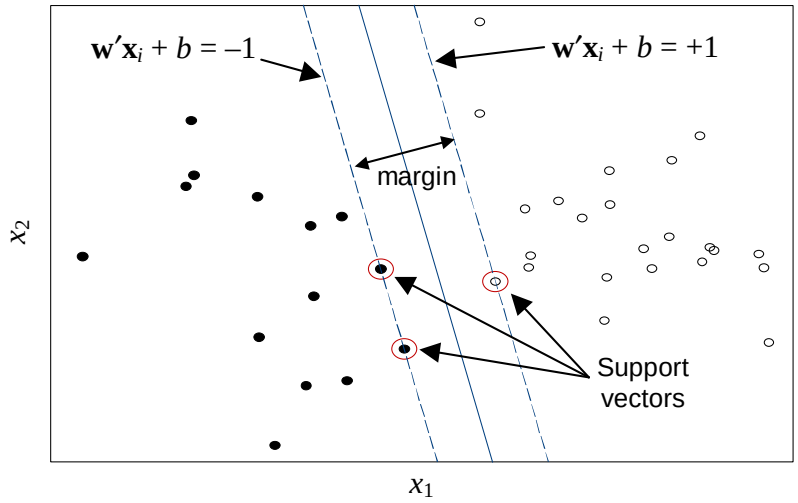


Figure 14: Geometry of the linear SVM. x_1 and x_2 represent two arbitrary continuous features

In the more practical case when the classes are not completely separable, SVMs still maximize the margin but allow for a limited number of observations to be on the wrong side of the decision boundary. We can introduce slack variables, $\xi_i \geq 0$, into the constraints (42) and (43) as

$$\mathbf{w}'\mathbf{x}_i + b \geq 1 - \xi_i \text{ when } y_i = +1 \tag{45}$$

$$\mathbf{w}'\mathbf{x}_i + b \leq -1 + \xi_i \text{ when } y_i = -1. \tag{46}$$

$$\sum_{i=1}^{N} \xi_i \leq B \tag{47}$$

Again SVMs maximize the margin subject to (45) and (46) but also penalize the sum of the ξ_i (47), restricting the budget we have for observations being to far from the decision boundary. The details of the fitting algorithm are fairly tedious but two important ideas come in their development. To fill in the details not covered here see the excellent SVM tutorial in Burges (1998). The SVM optimization problem can be recast in its Wolfe dual form as

$$J(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i' \mathbf{x}_j \tag{48}$$

along with several constraints on the α_i's relating to the observed data and the ξ_i. The solution for the prediction rule is

$$f(\mathbf{x}) = \text{sign}\left(\mathbf{w}'\mathbf{x} + b\right) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i'\mathbf{x} + b\right). \tag{49}$$

Usually most of the α_i's turn out to be 0. Those observations corresponding to a positive α_i are called the *support vectors*. Conveniently the prediction rule depends only on the support vectors. In Figure 14 there were only three support vectors.

As with the linear models described in section 4, SVMs are easily extended by expanding the set of features used in forming the decision boundary. We could assume that the decision boundary has a basis expansion form like (36). An interesting feature of the SVM is that the features only enter the fitting stage (48) and the prediction stage (49) through the inner product of the features, $\mathbf{x}_i'\mathbf{x}$. Rather than having to specify the set of basis functions explicitly, only the inner product of the basis functions needs definition. The common replacements for $\mathbf{x}_i'\mathbf{x}$ are

$$\begin{aligned} &\text{radial basis} && K(\mathbf{x}_i, \mathbf{x}) = \exp\left(\left\|\mathbf{x}_i - \mathbf{x}\right\|^2 / c\right) \\ &\text{polynomial} && K(\mathbf{x}_i, \mathbf{x}) = (1 + \mathbf{x}_i'\mathbf{x})^c \\ &\text{neural network} && K(\mathbf{x}_i, \mathbf{x}) = \tanh(c_1\mathbf{x}_i'\mathbf{x} + c_2)\,. \end{aligned} \tag{50}$$

These choices for the kernel are flexible and quick to compute. Therefore, to use SVMs for prediction one needs to select a kernel, the kernel tuning parameters, and the roughness of the decision boundary determined by B from (47).

To make the comparison of SVMs with other classifications we can reformulate the optimization problem (45)-(47) as a loss function shown in (51).

$$J(f) = \sum_{i=1}^{N} [1 - y_i f(\mathbf{x}_i)]_+ + \lambda \sum_{j=1}^{d} w_j^2\,. \tag{51}$$

Revisit Figure 2 to compare this to the other classification loss functions. The first part of the SVM loss function encourages y_i and $f(\mathbf{x}_i)$ to agree on the sign. When they disagree the penalty increases linearly in the magnitude of $f(\mathbf{x}_i)$. The second component penalizes the magnitude of the linear coefficients, equivalently controlling the margin. As with the smoothing splines (35), λ is a user specified smoothing parameter. SVMs, therefore, are characterized as using (51) as the loss function, having structural form (49) that may be generalized with a kernel choice from (50), and utilizing quadratic programming to fit the models to data.

5.6 Boosting

Although boosting has a technical definition in terms of computational learning theory, it has been more generally applied to methods that use a flexible gradient ascent approach to fit a prediction model. Schapire and Singer (1998) introduced the *Real AdaBoost* algorithm for classification problems, one of the most commonly used forms of boosting. Friedman et al (2000) decomposed this algorithm into its basic components: the loss function, the functional form, and the estimation procedure.

First consider the classification problem for which y takes on values +1 or –1. The Real AdaBoost loss function is

$$J(f) = E_{y|\mathbf{x}} \exp(-yf(\mathbf{x})). \tag{52}$$

If y and $f(\mathbf{x})$ frequently agree on the sign then (52) will be small. Furthermore, the AdaBoost loss function is an upper bound to the misclassification rate as shown in Figure 2. So intuitively, if on average y and $f(\mathbf{x})$ frequently agree on the sign then (52) is small and the misclassification rate will be small.

Boosting's main innovation comes when we look at how it estimates $f(\mathbf{x})$. Assume that $f_0(\mathbf{x})$ is our current guess for the best predictor. To improve upon our current guess we can consider adding a new function, $g(\mathbf{x})$, to "fix" it. Of course we want to choose the $g(\mathbf{x})$ that helps us to decrease the loss function (53) the most.

$$J(f) = E_{y|\mathbf{x}} \exp\big(-y(f_0(\mathbf{x}) + g(\mathbf{x}))\big). \tag{53}$$

Setting the derivative of (53) with respect to $g(\mathbf{x})$ equal to zero gives us the "direction" in which $f_0(\mathbf{x})$ needs the most improvement,

$$g(\mathbf{x}) = \frac{1}{2}\log\frac{P_w(y=+1\,|\,x)}{P_w(y=-1\,|\,x)}, \tag{54}$$

Where $P_w(y \mid \mathbf{x})$ is a weighted probability estimate with weight $\exp(-yf_0(\mathbf{x}))$. The brief derivation here deals with expectations but this is easily translated into data applications.

Begin with a naïve guess for $f_0(\mathbf{x})$, perhaps a constant, c.

For t in $1,\ldots,T$ do the following

1. Assign weights to the observations, $w_i = \exp(-yf_{t-1}(\mathbf{x}_i))$.

2. Obtain an estimate for $P_w(y = 1 \mid \mathbf{x})$ using any prediction model that can handle weights and puts out a probabilistic prediction. Classification trees are by far the most popular.
3. Form $g_t(\mathbf{x})$ as in (54) and lastly
4. Update the current guess as $f_t(\mathbf{x}) \leftarrow f_{t-1}(\mathbf{x}) + g_t(\mathbf{x})$.

For a new observation $\mathbf{x}$ the model predicts that $y = \text{sign}(f_T(\mathbf{x}))$.

It is helpful at this point to note some similarities with the linear logistic regression discussion in section 4.2.1. Like AdaBoost, the IRLS algorithm for linear logistic regression repeatedly fits weighted regression models to the dataset to obtain the best fit. In that situation the model is restricted to have a linear form and so the updates get directly absorbed into the current guess for β. In addition the IRLS weights were $p_i(1 - p_i)$, which are largest when $p_i = ½$, the point at which the equal cost decision rule is most in question. This behavior is replicated in the AdaBoost weights. Those weights are largest when y and $f_0(\mathbf{x})$ greatly disagree, perhaps the more difficult observations to classify.

Having discussed the loss function and estimation process, the last component is the predictor's form. This depends much on the base classifier selected for the $P_w(y = 1 \mid \mathbf{x})$ estimate. If we use the naïve Bayes classifier AdaBoost leaves it unchanged from the linear form (approximately so for the earlier Discrete AdaBoost algorithm, Ridgeway et al 1998). If we use stumps, decision trees with only a single split, the final model falls into the class of additive models of the form (39). To see this, note that the model that AdaBoost produces has a basis expansion form

$$f_T(\mathbf{x}) = c + g_1(\mathbf{x}) + g_2(\mathbf{x}) + \ldots + g_{T-1}(\mathbf{x}) + g_T(\mathbf{x}) \,. \tag{55}$$

In the additive model discussion the basis functions were decided upon ahead of time then fit simultaneously to the data. Boosting selects the basis functions in (55) greedily rather than simultaneously. If we use stumps then each $g_t(\mathbf{x})$ is a function of one feature alone. We can then collect all those $g_t(\mathbf{x})$'s that split on variable j into a single component.

$$f_T(\mathbf{x}) = c + g_1^*(x_1) + g_2^*(x_2) + \ldots + g_d^*(x_d) \,. \tag{56}$$

A two split decision tree for the base classifier results in an additive model with each term potentially being a function of two features. Therefore the depth of the tree allows us to select the complexity of the feature interactions.

For data mining applications decision trees turn out to be an excellent choice for the base model. Trees seamlessly handle continuous, ordinal, nominal, and missing data. The other methods that this chapter discusses do not necessarily

handle such a mix of data types easily. Trees are also invariant to one-to-one transformations of the features, using only the order information to determine the optimal partitions. For example, whether we use income or log(income) as a feature the tree will have the same form and produce the same predictions either way. The split points will be on the corresponding scale but all else will be the same. The performance of other methods can be particularly sensitive to feature transformations. This invariance in turn adds a bit of robustness to the boosted tree model.

Generalizations of the AdaBoost algorithm amount to selecting a different loss function, applying the same estimation procedure, and perhaps examining different base classifiers. Boosting can be generalized to all of the other loss functions discussed in section 3. Friedman (2001) gives a general framework for developing boosting algorithms, deriving specific algorithms for least squares, robust, and logistic regression.

To demonstrate the flexibility of boosting, consider the Cox loss function from section 0 for survival data (see also Ridgeway 1999). We will apply the boosting ideas to maximize the log-partial likelihood, the logarithm of (13).

$$J(f) = \sum_{i=1}^{N} \delta_i \left[f(\mathbf{x}_i) - \log\left(\sum_{j=1}^{N} I(t_j \geq t_i) \exp f(\mathbf{x}_j) \right) \right]. \tag{57}$$

As with AdaBoost we can find a function to add to $f(\mathbf{x}_i)$ to improve the model. The derivative of (57) pointwise with respect to $f(\mathbf{x}_i)$ indicates the direction that will adjust $f(\mathbf{x}_i)$ to increase the likelihood. The derivative equals

$$z_i = \delta_i - \sum_{j=1}^{N} \delta_j \frac{I(t_i \geq t_j) \exp f(\mathbf{x}_i)}{\sum_{k=1}^{N} I(t_k \geq t_j) \exp f(\mathbf{x}_k)}. \tag{58}$$

This means that for some step size, ρ, if all the $f(\mathbf{x}_i)$ in (57) were replaced with $f(\mathbf{x}_i) + \rho z_i$ then the likelihood would increase. In the AdaBoost algorithm ρ is exactly 1 but more generally we need to set it. Generally, z_i has information indicating how to adjust the current guess for $f(\mathbf{x})$. To pull in the features we fit a regression tree, $g(\mathbf{x})$, predicting z_i from $\mathbf{x}_i$. If $g(\mathbf{x})$ can capture the gradient information from z_i a good update is $f(\mathbf{x}) \leftarrow f(\mathbf{x}) + \rho g(\mathbf{x})$, where ρ can be selected with a line search.

To examine the performance of boosting Cox's proportional hazards model, we turn to a clinical trial for testing the drug DPCA for the treatment of primary biliary cirrhosis of the liver (PBC). This dataset has been the subject of several modern data analyses (Dickson et al 1989, Fleming et al 1991, Raftery et al

1996). The data consist of 310 patients with complete observations on the predictors. Of these, 124 patients died during the study and the remaining 186 were censored observations. Of the eight features Raftery et al (1996) considered, we selected the six continuous features for use in the model.

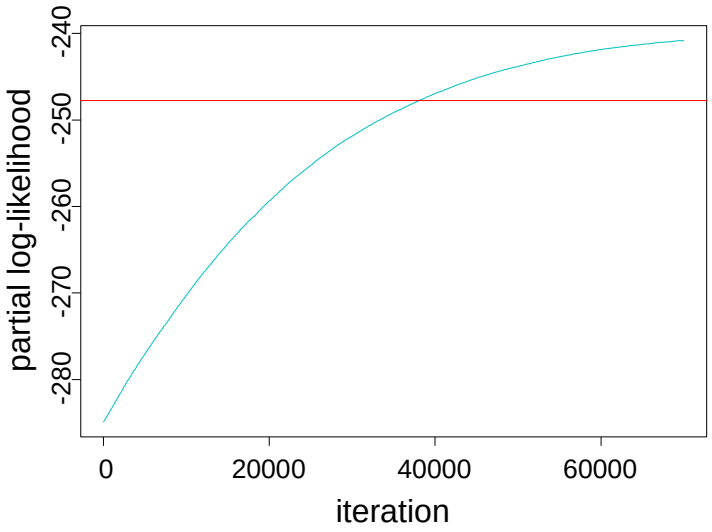


Figure 15: Partial log-likelihood for PBC data. The upward trending curve indicates the boosted trees performance on test data. The horizontal line indicates the partial likelihood from the linear Cox model.

We tested this method by comparing the out-of-sample predictive performance of the linear Cox model to the boosted Cox model using regression stumps as $g_t(\mathbf{x})$, the base regressor. To judge the out-of-sample predictive performance of the two models, we trained each on half of the observations, reserving the rest for a test set. Figure 15 shows the value of the validation log-likelihood as the algorithm proceeds. To obtain a very smooth prediction surface we shrunk the ρ by a factor of 1000, making each adjustment very slight and requiring many more iterations. We see that after 40,000 iterations the boosted estimate has surpassed the linear Cox model. Furthermore, for the next 30,000 iterations the boosted version continues to improve, although at a diminishing rate of return. Even though the model fitting used many iterations, it takes about a minute to complete the iterations on this small dataset.

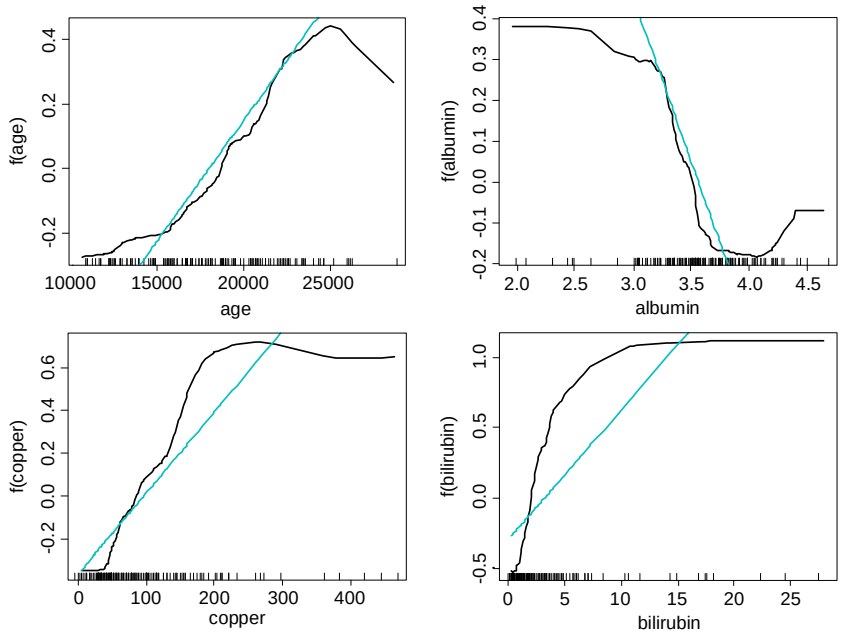


Figure 16: Boosted estimates of the main effects of the PBC data. The curves are the boosted estimates of the functions and the superimposed lines are the estimates from the linear model.

Since the base regressors were one-split decision trees, we can investigate the estimates of the main additive effects as in (56). Figure 16 shows the boosted estimates of four of those effects along with those based on the linear model. Every variable shows evidence of either a threshold effect, a saturation effect, or both. In the region where most of the data points are concentrated, for most of the variables (except bilirubin) the underlying regressor is nearly linear. For this reason we would expect the linear model to perform reasonably well. However, for a patient with a more extreme value for any of the variables the boosted model is far superior. When effects depart substantially from linearity, as in the PBC dataset, accurate survival prediction for all patients depends on a non-linear estimation procedure like boosting.

6 Availability of software

This section returns to the methods reviewed in this chapter indicating what software tools are available. Many of the available tools come in the form of add-ons to other packages, such as S-plus, Gauss, or Matlab. In general, a simple web search for the method will likely turn up everything from original source code, to Matlab scripts, to full stand-alone commercial versions. The R project

(www.r-project.org) is a great place to start for statistical algorithms. It is a free, full-featured statistical package maintained by many in the statistical community and is often the place where the latest statistical algorithms first appear. Weka (http://www.cs.waikato.ac.nz/ml/weka/) is another well integrated, ever improving, package that implements many algorithms popular in the machine learning community. There are stand-alone packages that implement individual algorithms but they often lack convenient methods for managing the datasets.

Generalized linear models - All standard statistical packages, SAS, SPSS, Stata, S-plus, R, etc, include GLM procedures as well as the Cox model and other survival models.

Generalized additive models - Original source code is available from StatLib (lib.stat.cmu.edu/general). The standard S-plus (`gam()`) and R distributions (mgcv library) come with GAM. SAS as of version 8.2 has PROC GAM.

K Nearest Neighbors - Available in R (`knn()` in the `class` library) and Weka (weka.classifiers.IBk).

Tree structured models - Salford Systems (www.salford-systems.com) maintains and develops the official implementation of CART. Rpart, developed at the Mayo clinic, is part of the standard R distribution and is full-featured including surrogate splitting for missing values. S-plus (`tree()`) and SPSS (AnswerTree) include tree structured modeling but do not implement all the features of the CART algorithm. Also look for implementations of Quinlan's C4.5 and C5.0 on the web.

MARS - Salford Systems maintains and develops MARS. There are various other implementations, usually add-ons for other packages such as S-plus or Gauss.

Support vector machines - Many research packages are now available. SVM-Light is one of the most popular, freely available at svmlight.joachims.org.

Neural networks - There are many variants on neural networks and at least as many programs out there. Matlab users may want to look at Netlab (www.ncrg.aston.ac.uk/netlab) to become more familiar with the technique. For a large catalog of free and commercial packages visit www.emsl.pnl.gov:2080/proj/neuron/neural.

Naïve Bayes - Available in Weka. Also Bayda exclusively implements naïve Bayes (www.cs.helsinki.fi/research/cosco/Projects/NONE/SW)

Boosting - To accompany his Gradient Boosting paper, Friedman has an R add-on for the LogitBoost algorithm, least squares, and robust regression methods. (www-stat.stanford.edu/~jhf). Boosting methods for AdaBoost and other loss functions including Poisson regression and the Cox model are available at

www.i-pensieri.com/gregr/gbm.shtml as an R add-on, additional functionality forthcoming. Weka can apply the AdaBoost algorithm to any of its standard classifiers

7 Summary

For prediction problems the process begins by choosing a loss function, then selecting the type of predictor, and lastly determining how best to fit the model to potentially massive datasets. Of course, this ordering is ideal, but in practice we may find that computational complexity or a policy constraint prevents us from modeling exactly how we had hoped. As section 5.6 showed algorithms designed to minimize particular loss functions can be slightly modified to minimize other loss functions that we might prefer. Indeed, SVMs have been used to maximize the logistic log-likelihood and k-nearest neighbors has been used for to minimize squared error loss. So while this chapter focused only on particular combinations commonly used in practice, data analysts can and should think creatively to match loss, model structure, and fitting algorithm to the problem at hand.

In public policy problems as well as business and scientific applications, interpretability is frequently a constraint. At times the SVM or the boosted tree model might predict the outcome better than the more easily interpreted models. However, applications often require a model fit that can be translated into humanly understandable relationships between the features and the outcome. When a model like the linear model adequately approximates the optimal predictor, interpretation is not in doubt. One should use great caution when trying to interpret an apparently interpretable model when more complex models exist that substantially outperform it. Clearly, we should be careful when creating policies and actions based on interpreting models that do not fit the data well. See Breiman (2001) and the accompanying discussion for both sides of this debate. As noted in section 5.2 interpreting the apparently interpretable decision tree actually requires a fair bit of care. Research continues to develop in the area of converting the competitive “black box” models into interpretable patterns. Friedman (2001), for example, makes substantial gains in turning complex boosted regression models into understandable relationships between features and outcomes.

While interpretability potentially is an issue, scalability is the main problem facing data miners. The simplest models such as the linear models can be fit to data in a small number of passes through the data, sometimes only once. But the benefit of having large datasets is the ability to model complex, non-linear, deeply interacted functions. As discussed in the tree model chapter, researchers have made substantial progress in creating scalable algorithms for decision trees. Research has also shown that subsampling within boosting iterations not only decreases the computational complexity but also incidentally improves predictive

performance. Computational advances are making SVMs potentially applicable to large data mining problems. Watch for continued advances in each of these areas over the next few years.

References

Barron, A.R. (1991). Complexity regularization with application to neural networks. In G. Roussas (Ed.), Nonparametric functional estimation and related topics (pp. 561-576). Dordrecht, Netherlands: Kluwer Academic Publishers.

Barron, A.R. (1993). Universal approximation bounds for superpositions of a sigmoid function. IEEE Transactions on Information Theory, 39, pp. 930–945.

Berger, J. & Wolpert, R. (1984). The likelihood principle. Institute of Mathematical Statistics.

Bloomfield, P. & Steiger, W.L. (1983). Least absolute deviations: Theory, applications, and algorithms. Boston, Mass: Birkhauser.

Brier, G.W. (1950). Verification of forecasts expressed in terms of probability. Monthly Weather Review 78(1), pp. 1–3.

Breiman, L. (2001). Statistical modeling: The two cultures. Statistical Science, 16(3), pp.199–231.

Burges, C.J.C. (1998). A tutorial on support vector machines for pattern recognition. Knowledge Discovery and Data Mining, 2(2).

Dickson, E.R., Grambsch, P.M., Fleming, T.R., Fisher, L.D., & Langworthy, A. (1989). Prognosis in primary biliary cirrhosis: Model for decision-making. Hepatology, 10, pp. 1–7.

Fisher, R.A. (1936). The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7, pp.179–188.

Fleming, T.R. & Harrington, D.P. (1991). Counting processes and survival analysis. New York, NY: Wiley.

Freund, Y. & Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1), pp.119–139.

Friedman, J., Hastie, T. & Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting (with discussion). Annals of Statistics, 28(2), pp. 337–374.

Friedman, J. (1991). Multivariate adaptive regression splines (with discussion). Annals of Statistics, 19(1), pp. 1-82.

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29(4).

Gordon, L. & Olshen, R.A. (1984). Almost surely consistent nonparametric regression from recursive partitioning schemes. Journal of Multivariate Analysis, 15, pp. 147–163.

Greene, W.H. (1999). Econometric analysis (4^{th} ed.). Prentice-Hall.

Hand, D.J. & Yu, K. (2001). Idiot's Bayes - not so stupid after all?. International Statistical Review, 69(3), pp. 385-398.

Hastie, T. & Tibshirani, R. (1990). Generalized Additive Models. London: Chapman and Hall.

Hastie, T., Tibshirani , R, & Buja, A.(1994). Flexible discriminant analysis by optimal scoring. Journal of the American Statistical Association, 89, pp.1255–1270.

Hastie, T., Tibshirani, R., Friedman , J. (2001). Elements of statistical learning: Data mining, prediction, and inference. Springer.

Huber, P. (1964). Robust estimation of a location parameter. Annals of Mathematical Statistics, 53, pp. 73–101.

McCullagh, P. & Nelder, J. (1989). Generalized linear models. London: Chapman and Hall.

Raftery, A.E., Madigan, D. & Volinsky, C.T. (1996). Accounting for model uncertainty in survival analysis improves predictive performance. In J.M. Bernardo, J.O. Berger, A.P. Dawid & A.F.M. Smith (Eds.), Bayesian statistics 5 (pp. 323–349). Oxford University Press.

Ridgeway, G. (2002). Looking for lumps: Boosting and bagging for density estimation. Computational Statistics and Data Analysis, 38(4), pp. 379–392.

Ridgeway, G., Madigan, D., Richardson, T. & O'Kane, J. (1998). Interpretable boosted naïve Bayes classification. R. Agrawal, P. Stolorz, G. Piatetsky-Shapiro. (Eds.), Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (pp. 101-104).

Sanders, F. (1963). On subjective probability forecasting. Journal of Applied Meteorology, 2, pp.191–201.

Schapire, R. E. & Singer, Y. (1998). Improved boosting algorithms using confidence-rated predictions. Proceedings of the Eleventh Annual Conference on Computational Learning Theory.

Vapnik, V. (1996). The nature of statistical learning theory. New York, NY: Springer-Verlag.

Wahba, G. (1990). Spline models for observational data. Philadelphia, PA: SIAM.

Yates, J.F. (1982). External correspondence: Decompositions of the mean probability score. Organizational Behavior and Human Performance, 30, pp. 132–156.