Stat 342 Outline

Steve Vardeman
Iowa State University

December 17, 2014

Abstract

This outline summarizes the formalisms of the main points of lectures. Examples are not here, but are in notes provided during class and available on the course web page.

Contents

1 Introduction to Statistical Inference and Some Basic Probabilistic and Statistical Decision Analysis 3
1.1 Optimal Prediction in a Fully Specified Probability Model 3
1.2 General Statistical Inference and Decision Theory 5
1.3 Bayes Inference and Decision Theory 6

2 Some More Probability Useful in Statistics (and Predictive Analytics/Machine Learning) 7
2.1 Distributions of Functions of Random Variables/Derived Distributions 7
2.2 Approximations/Convergence of Distributions 9
2.3 Simulations 11
2.3.1 IID Simulations 11
2.3.2 Markov Chain Monte Carlo (MCMC) 12
2.4 Bootstrapping 14

3 Likelihood Theory 16
3.1 Sufficiency/"(Statistically) Lossless Feature Selection" in Statistical Models 17
3.2 Fisher Information 19
3.3 (Large Sample) Maximum Likelihood Theory 21
3.4 Likelihood Ratio Testing Theory 24
3.4.1 Simple versus Simple Testing 24
3.4.2 (Large Sample) Theory for Likelihood Ratio Tests of a Point Null Hypothesis and Corresponding Confidence Sets 25

4 Classification Problems **26**
4.1 Approximating Optimal Classifiers Based on "Training Data," Nearest Neighbor Classifiers, Predictor Complexity, Over-fit, and Cross-Validation 27
4.2 Bootstrapping Basic Classifiers, Out-of-Bag Error Rates, and "Bagging"/"Majority Votes" 29
4.3 Practical Basic Classification Methods 30
4.3.1 Nearest Neighbor Rules 30
4.3.2 (Binary) Decision/Classification Trees and Random Forests 30
4.3.3 Rules Based on Logistic Regression 33
4.3.4 Support Vector Classifiers 35
4.4 "Ensembles" of Classifiers 38

5 SEL Prediction Problems **39**
5.1 Normal Linear Models and OLS 39
5.2 Other Practical Predictors Based on "Training Data" 41
5.2.1 Other (Non-OLS) Linear Predictors 41
5.2.2 Nearest Neighbor Predictors 44
5.2.3 Tree-Based SEL Predictors 45
5.2.4 Kernel Smoothing Methods 46
5.2.5 Neural Networks 50
5.3 Ensembles of SEL Predictors 52
5.3.1 "Stacking" 52
5.3.2 SEL Boosting 54

1 Introduction to Statistical Inference and Some Basic Probabilistic and Statistical Decision Analysis

We begin with some simple "decision theory" and formalisms for "statistical inference" based on probability concepts covered in Stat 341.

1.1 Optimal Prediction in a Fully Specified Probability Model

For a univariate random variable y and (possibly mulitvariate/p-dimensional) random quantity $\boldsymbol{x}$ assume that a joint distribution for the pair $(y, \boldsymbol{x})$ is specified by a "density" (a pdf, or pmf, or density for a "mixed" partially-continuous-partially-discrete distribution)

$$f(y, \boldsymbol{x})$$

Of interest is choice of a "good" function of $\boldsymbol{x}$, say $\hat{y}(\boldsymbol{x})$, to be used to predict an unobserved value y based on an observed value $\boldsymbol{x}$.

A way to approach this problem is through the definition of a "loss function" and "risk." Suppose that a (loss) function $L(\cdot, \cdot)$ maps pairs of possible values of y to non-negative real values. Then for $\hat{y}$ a fixed/non-random prediction of y, one might say that a **loss**

$$L(y, \hat{y}) \geq 0$$

is suffered in prediction. If $\hat{y}(\boldsymbol{x})$ (a random variable, as it depends upon the random $\boldsymbol{x}$) is used in prediction, the non-negative *random* loss $L(y, \hat{y}(\boldsymbol{x}))$ is suffered. (This is a function of the random pair $(y, \boldsymbol{x})$.) The expected value of this random loss

$$\mathrm{E}L(y, \hat{y}(\boldsymbol{x})) \tag{1}$$

is called the **risk** associated with the predictor. (Here, one is averaging out over the joint distribution of the pair $(y, \boldsymbol{x})$.)

Choice of $\hat{y}(\boldsymbol{x})$ optimizing (minimizing) risk can be based on the notion of "iterated expectation." That is, for a random pair (V, W) and real-valued function h, it's a Stat 341 fact that

$$\mathrm{E}h(V, W) = \mathrm{EE}[h(V, W)|W]$$

That is, one may for every value of W use the conditional distribution of V given W to average out V (producing the conditional mean given W) and then average according to the marginal distribution of W. In the present context, we apply this by representing risk (1) as

$$\mathrm{E}L(y, \hat{y}(\boldsymbol{x})) = \mathrm{EE}[L(y, \hat{y}(\boldsymbol{x}))|\boldsymbol{x}] \tag{2}$$

One may then, one $\boldsymbol{x}$ at a time, look for a $\hat{y}$ minimizing $\mathrm{E}[L(y, \hat{y})|\boldsymbol{x}]$ (the conditional mean averaging out y given $\boldsymbol{x}$ of the loss associated with prediction

$\hat{y}$) and set $\hat{y}^{\text{opt}}(\boldsymbol{x})$ equal to that minimizer. That is, the representation (2) immediately implies that a minimum risk predictor is

$$\hat{y}^{\text{opt}}(\boldsymbol{x}) = \arg\min_{\hat{y}} \text{E}\left[L\left(y, \hat{y}\right) | \boldsymbol{x}\right] \tag{3}$$

Special cases of loss functions produce particularly tractable forms for optimal predictor (3). In particular:

1. **Squared Error Loss** (SEL) is

   $$L\left(y, \hat{y}\right) = \left(y - \hat{y}\right)^2$$

   and the corresponding optimal predictor is

   $$\hat{y}^{\text{opt}}(\boldsymbol{x}) = \text{E}\left[y|\boldsymbol{x}\right]$$

   the conditional mean of y given $\boldsymbol{x}$. (This follows from the Stat 341 fact that the number c minimizing $\text{E}(V-c)^2 = \text{Var}V + (\text{E}V - c)^2$ is $c = \text{E}V$. One is simply applying this to a conditional distribution.)

2. **Absolute Error Loss** (AEL) is

   $$L\left(y, \hat{y}\right) = \left|y - \hat{y}\right|$$

   and the corresponding optimal predictor is a *median* of the conditional distribution of y given $\boldsymbol{x}$, say

   $$\hat{y}^{\text{opt}}(\boldsymbol{x}) = \text{median}\left[y|\boldsymbol{x}\right]$$

   (This follows from the Stat 341 fact that a number c minimizing $\text{E}|V-c|$ is any median of the distribution of V. One is simply applying this to a conditional distribution.)

3. In cases where y is discrete, say taking values in $\{1, 2, \ldots, K\}$, one might consider 0-1 **Loss**,

   $$L\left(y, \hat{y}\right) = I\left[y \neq \hat{y}\right]$$

   (The "indicator function" $I[\cdot]$ is 1 if the statement "$\cdot$" is true and is 0 if it is false. So this loss is 1 if the prediction $\hat{y}$ fails to match y and is 0 if the prediction matches y.) This form of prediction is typically known as "classification" in the machine learning/data mining world.

   Here

   $$\text{E}\left[L\left(y, \hat{y}\right) | \boldsymbol{x}\right] = \sum_{y \neq \hat{y}} 1 \cdot P\left[y = \hat{y}|\boldsymbol{x}\right]$$

   and so it's pretty clear that

   $$\hat{y}^{\text{opt}}(\boldsymbol{x}) = \arg\min_{\hat{y}} \text{E}\left[L\left(y, \hat{y}\right) | \boldsymbol{x}\right] = \arg\max_{\hat{y}} P\left[y = \hat{y}|\boldsymbol{x}\right]$$

   (One looks at the conditional distribution of $y|\boldsymbol{x}$ and picks the possible value of y with the largest conditional probability as one's prediction.)

1.2 General Statistical Inference and Decision Theory

Consider now the problem of **statistical inference**. This kind of problem is one where some aspect or aspects of a probability model are taken to be unknown/undetermined and data in hand are used to provide information about those model features.

Formally, a parametric statistical model is a probability model specified by a density (a pdf, a pmf, or a mixed density)

$$f(\boldsymbol{x}|\boldsymbol{\theta}) \tag{4}$$

for an observable random quantity $\boldsymbol{x}$ depending upon some (potentially p-variate) parameter $\boldsymbol{\theta}$ that is taken to be unknown. Observed "data" $\boldsymbol{x}$ are used to guide one's thinking about $\boldsymbol{\theta}$. *Treated as a function of* $\boldsymbol{\theta}$, the form (4) is called the **likelihood function**. We will have much more to say later in the course about how a likelihood function can be used in making inferences about $\boldsymbol{\theta}$ outside of a decision theory framework, but for the time being will make the simple observation that a $\boldsymbol{\theta}$ maximizing $f(\boldsymbol{x}|\boldsymbol{\theta})$ can be called a **maximum likelihood estimate** of $\boldsymbol{\theta}$. That is,

$$\hat{\boldsymbol{\theta}}^{\text{MLE}}(\boldsymbol{x}) = \arg\max_{\boldsymbol{\theta}} f(\boldsymbol{x}|\boldsymbol{\theta})$$

A decision-theoretic approach to statistical inference supposes that (based on the observable $\boldsymbol{x}$) one may take some "action" $a(\boldsymbol{x})$ and (depending upon the unknown value of $\boldsymbol{\theta}$) suffer a loss

$$L(\boldsymbol{\theta}, a(\boldsymbol{x})) \geq 0$$

(Here a loss function takes parameter vectors and actions as inputs and outputs a non-negative loss.) The mean loss suffered according to the $\boldsymbol{\theta}$ distribution of $\boldsymbol{x}$, namely

$$R(\boldsymbol{\theta}) = \mathrm{E}_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, a(\boldsymbol{x}))$$

is a function of $\boldsymbol{\theta}$ called the **risk function**. (Here one is averaging out over $\boldsymbol{x}$ according to its $\boldsymbol{\theta}$ distribution.) One looks for decision functions $a(\boldsymbol{x})$ with "good" (small) risk functions. In general it is impossible to find a decision function with risk that is smaller than all other risk functions at all $\boldsymbol{\theta}$ values.

As one concrete example of the statistical decision theory framework, one might consider a case where θ is univariate and $L(\theta, a) = (\theta - a)^2$. This is called the squared error loss estimation problem and in this context $a(\boldsymbol{x})$ is meant to approximate θ and is called an **estimator** of θ. Here the risk function is the **mean squared error** of estimation

$$R(\theta) = \mathrm{E}_\theta (\theta - a(\boldsymbol{x}))^2 = \mathrm{Var}_\theta (a(\boldsymbol{x})) + (\mathrm{E}_\theta a(\boldsymbol{x}) - \theta)^2$$

which is decomposable into the variance of the estimator plus its squared **bias** (the difference between its mean and the parameter).

A second concrete example here is that of **simple versus simple testing**. This is a case where there are only two possible values of θ (say θ may take only values in $\{0,1\}$) and a loss function is the 0-1 loss, $L(\theta,a) = I[\theta \neq a]$. Here the risk function is

$$\begin{aligned} R(\theta) &= \mathrm{E}_\theta L(\theta, a(\boldsymbol{x})) \\ &= 1 \cdot P_\theta[\theta \neq a(\boldsymbol{x})] + 0 \cdot P_\theta[\theta = a(\boldsymbol{x})] \\ &= P_\theta[\theta \neq a(\boldsymbol{x})] \end{aligned}$$

the error probability function.

1.3 Bayes Inference and Decision Theory

A so-called **Bayes** approach to statistical inference begins with the basic statistical model of Section 1.2 and in particular the likelihood function (4), but then more or less turns the inference problem into a probability problem by adding an element to the modeling. That is, one assumes that there is some so-called **prior distribution** for the parameter $\boldsymbol{\theta}$, (say) specified by a prior density

$$g(\boldsymbol{\theta})$$

This distribution is meant to encode what one believes about the value of the parameter external to data $\boldsymbol{x}$. Formally, treating the statistical model as specifying a conditional distribution for $\boldsymbol{x}$ given $\boldsymbol{\theta}$ and the prior distribution as specifying a marginal distribution for the parameter, one then has a joint distribution for the pair $(\boldsymbol{\theta}, \boldsymbol{x})$ with joint density

$$f(\boldsymbol{\theta}, \boldsymbol{x}) = f(\boldsymbol{x}|\boldsymbol{\theta}) \cdot g(\boldsymbol{\theta})$$

This joint distribution has conditionals for $\boldsymbol{\theta}$ given $\boldsymbol{x}$. These are called **posterior distributions** with densities

$$g(\boldsymbol{\theta}|\boldsymbol{x})$$

and the one for the data in hand is used to guide inference about $\boldsymbol{\theta}$. Notice that (since $g(\boldsymbol{\theta}|\boldsymbol{x}) = f(\boldsymbol{\theta}, \boldsymbol{x}) / f(\boldsymbol{x})$) as a function of $\boldsymbol{\theta}$

$$g(\boldsymbol{\theta}|\boldsymbol{x}) \propto f(\boldsymbol{x}|\boldsymbol{\theta}) \cdot g(\boldsymbol{\theta}) \tag{5}$$

and in contrast to the situation of Section 1.2 where the likelihood function (4) was pointed out as guiding inference, in this formulation it is instead the likelihood times the prior (namely the right hand side of proportionality (5)) that is used.

Notice that since $g(\boldsymbol{\theta}|\boldsymbol{x})$ specifies a (conditional given data) probability distribution, it is possible for Bayesians to talk about (posterior) probability that an inference is correct, something that is strictly not allowed in non-Bayesian contexts. That is, if data in hand lead to ordinary 95% confidence limits for θ that are, say, 3 and 7, without the Bayes assumption it is nonsensical to say

there is a "probability" .95 that θ is between 3 and 7. But using some prior $g(\theta)$ it might be the case that $P[3 < \theta < 7|\boldsymbol{x}] = .95$. In this case, the common jargon is that the interval $(3, 7)$ is a 95% (Bayesian) **credible interval** for θ.

The addition of a prior distribution to the basics of statistical modeling makes Bayes decision theory formally equivalent to the prediction theory we began with in Section 1.1. That is, armed with a joint distribution for $(\boldsymbol{\theta}, \boldsymbol{x})$ and loss $L(\boldsymbol{\theta}, a(\boldsymbol{x}))$ there is a *single number* risk

$$\mathrm{E}R(\boldsymbol{\theta}) = \mathrm{E}L(\boldsymbol{\theta}, a(\boldsymbol{x})) = \mathrm{EE}[L(\boldsymbol{\theta}, a(\boldsymbol{x}))|\boldsymbol{x}]$$

and it's possible to see how to optimize this by choice of $a(\boldsymbol{x})$. That is,

$$a^{\mathrm{opt}}(\boldsymbol{x}) = \arg\min_a \mathrm{E}[L(\boldsymbol{\theta}, a)|\boldsymbol{x}]$$

One may, one $\boldsymbol{x}$ at a time, look for an a minimizing the conditional mean (over $\boldsymbol{\theta}$) loss given $\boldsymbol{x}$, and set $a^{\mathrm{opt}}(\boldsymbol{x})$ equal to that minimizer. This has its obvious implications for common/simple loss functions like squared error and 0-1.

2 Some More Probability Useful in Statistics (and Predictive Analytics/Machine Learning)

Somewhat more probability background than is covered in a first undergraduate course in the subject is really needed to support theory for statistical inference and predictive analytics. We proceed to provide some of it here.

2.1 Distributions of Functions of Random Variables/Derived Distributions

A standard problem of probability theory is to start with a probability model for some variable $\boldsymbol{x}$ and derive the implied distribution for some (potentially multivariate) function of the variable, say $\boldsymbol{h}(\boldsymbol{x})$. In the abstract, this is a mathematically trivial problem. To evaluate $P[\boldsymbol{h}(\boldsymbol{x}) \in A]$ one considers the inverse image of the set A. Call this the set $\boldsymbol{h}^{-1}(A)$ (the set $\{\boldsymbol{x}|\boldsymbol{h}(\boldsymbol{x}) \in A\}$). $P[\boldsymbol{h}(\boldsymbol{x}) \in A]$ is then simply the probability that the distribution of $\boldsymbol{x}$ assigns to the set $\boldsymbol{h}^{-1}(A)$.

In simple discrete contexts, identifying the elements of $\boldsymbol{h}^{-1}(A)$ and simply summing up their probabilities to get $P[\boldsymbol{h}(\boldsymbol{x}) \in A]$ can be quite feasible. Even where more complicated models are involved the conceptual fundamentals involved are really only these. That is,

1. sometimes one can effectively discretize a complicated distribution for $\boldsymbol{x}$ and get a good approximation to the distribution of $\boldsymbol{h}(\boldsymbol{x})$ in this way, and
2. in continuous models where one can sometimes rely upon change of variable theorems for integrals to do computations, the arguments behind those theorems are essentially only this (with increasingly fine discretization).

For particular situations, various ad hoc approaches (like recognizing how to easily identify a cdf or a moment generating function for the distribution of $\boldsymbol{h}(\boldsymbol{x})$) can lead to simple identification of a useful form for the distribution. Chapter 6 of the Stat 341 text is full of "methods" of these various kinds that can *sometimes* be used. (They do *not* constitute anything like a comprehensive list of approaches, one of which is guaranteed to work in any problem one might care to pose.) Several "results" (statements that could be made precise with enough work) that represent these methods are the following.

Proposition 1 *If F is a cdf for the random variable X and $U \sim U(0,1)$, then $F^{-1}(U)$ has the same distribution as does X.*

Proposition 2 *If W has a continuous distribution with pdf f_W and h is strictly increasing or strictly decreasing with derivative h', then the random variable $V = h(W)$ has a distribution with pdf*

$$f_V(v) = \frac{1}{|h'(h^{-1}(v))|} f_W\left(h^{-1}(v)\right)$$

Proposition 3 *If the transform that takes a distribution for the random variable X and (potentially) produces a function of t defined by*

$$M_X(t) = E\exp(tX)$$

makes sense (the expectation is finite) for all t in some open interval around 0, then any Y with $E\exp(tY) = M_X(t)$ in that interval has the same distribution as X.

Proposition 1 specifies the so called **inverse probability integral transform**. Proposition 2 is the 1-D "transformation" theorem (and can be generalized to multivariate cases). Proposition 3 says that so-called "moment generating functions" are unique in the sense of unambiguously picking out/identifying a single distribution.

For the most-studied statistical models, various theorems have been proved providing distribution facts useful in statistical inference. As illustrations of what is possible we next record several such results.

Theorem 4 *For U_1 and U_2 independent normal random variables $U_1 + U_2$ is also normal.*

Definition 5 *For $Z_1, Z_2, \ldots, Z_\nu$ independent standard normal random variables, the distribution of*

$$Z_1^2 + Z_2^2 + \cdots + Z_\nu^2$$

is called the χ^2_ν distribution.

Theorem 6 *The χ^2_ν distribution has pdf*

$$f_\nu(x) = \frac{1}{2^{\nu/2}\Gamma\left(\frac{\nu}{2}\right)} x^{(\nu/2)-1} \exp\left(-\frac{x}{2}\right) I[x > 0]$$

Theorem 7 *For $W_1 \sim \chi^2_{\nu_1}$ independent of $W_2 \sim \chi^2_{\nu_2}$, $W_1 + W_2 \sim \chi^2_{\nu_1+\nu_2}$.*

Definition 8 *For Z standard normal independent of $W \sim \chi^2_\nu$, the distribution of*

$$Z/\sqrt{W/\nu}$$

is called the t_ν distribution.

Theorem 9 *The t_ν distribution has pdf*

$$f_\nu(x) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

Definition 10 *For $W_1 \sim \chi^2_{\nu_1}$ independent of $W_2 \sim \chi^2_{\nu_2}$, the distribution of*

$$(W_1/\nu_1)/(W_2/\nu_2)$$

is called the F_{ν_1,ν_2} distribution.

Theorem 11 *The F_{ν_1,ν_2} distribution has pdf*

$$f_{\nu_1,\nu_2}(x) = \frac{1}{B\left(\frac{\nu_1}{2},\frac{\nu_2}{2}\right)}\left(\frac{\nu_1}{\nu_2}\right)^{(\nu_1/2)} x^{(\nu_1/2)-1}\left(1+\frac{\nu_1}{\nu_2}x\right)^{-\frac{\nu_1+\nu_2}{2}}$$

Theorem 12 *For $V_1, V_2, \ldots, V_n$ independent identically distributed normal random variables with means μ and variances σ^2, the random variables*

$$\bar{V} = \frac{1}{n}\sum_{i=1}^n V_i \quad \textit{and} \quad S^2 = \frac{1}{n-1}\sum_{i=1}^n \left(V_i - \bar{V}\right)^2$$

are independent. Further,

$$\bar{V} \sim N\left(\mu, \frac{\sigma^2}{n}\right) \quad \textit{and} \quad \frac{(n-1)S^2}{\sigma^2} \sim \chi^2_{n-1}$$

2.2 Approximations/Convergence of Distributions

When it is not easy to see how to find exact probabilities for $\boldsymbol{h}(\boldsymbol{x})$ based on some probability model for $\boldsymbol{x}$, it can be attractive to instead look for approximations. The mathematics of doing this is usually phrased in terms of some sequence of variables y_n that depend upon some typically increasingly large-dimensional $\boldsymbol{x}_n$ and statements that for sets A, values $P[y_n \in A]$ converge to some appropriate constants. The simplest and most famous results of this type are so-called **laws of large numbers** that say that *sample averages* based on large samples have distributions that "pile up" near the "population" mean and **central limit theorems** that say that as they do that, their microscopic distributional shape tends to normal.

One law of large numbers is next.

Theorem 13 *(Law of Large Numbers) For $V_1, V_2, \ldots$ independent random variables with means μ and variances σ^2, and $\bar{V}_n = \frac{1}{n}\sum_{i=1}^{n} V_i$ the sample average of the first n, for every $\epsilon > 0$*

$$P\left[\left|\bar{V}_n - \mu\right| > \epsilon\right] \to 0 \quad as \quad n \to \infty$$

The theorem implies for example, that in a statistical problem where one assumes that the entries of $\boldsymbol{x}_n = (x_1, x_2, \ldots, x_n)$ are iid according some distribution specified by density $f(x|\boldsymbol{\theta})$, for large n with high probability, $\bar{x}_n$ will be "near" $\mathrm{E}_{\boldsymbol{\theta}} x_1$ (the $\boldsymbol{\theta}$ expected value of any one of the observations x_i). This is a statement about the convergence of the distribution of $\bar{x}_n$ to a point mass distribution concentrated at μ (a function of $\boldsymbol{\theta}$) no matter what is the value of the parameter $\boldsymbol{\theta}$.

While Theorem 13 guarantees that the distribution of a sample average of iid variables is increasingly concentrated around the marginal mean, one might still ask about the shape of that highly concentrated distribution. That is typically approximately normal. What can be proved is that probabilities for the *standardized* sample mean are approximately standard normal ones. Next is one such central limit theorem.

Theorem 14 *(Central Limit Theorem) For $V_1, V_2, \ldots$ iid random variables with means μ and variances σ^2, for any $-\infty \leq a \leq b \leq \infty$*

$$P\left[a \leq \frac{\bar{V}_n - \mu}{\sigma/\sqrt{n}} \leq b\right] \to \Phi(b) - \Phi(a) \quad as \quad n \to \infty$$

A useful extension of the foregoing concerns what happens when one considers a differentiable function of a variable like a sample mean that has an approximately normal distribution. The following is one version of what is sometimes called a "delta method."

Theorem 15 *Suppose that for some sequence of random variables y_n there are sequences of constants m_n and s_n with $m_n \to m$ and $s_n \to 0$ such that for any $-\infty \leq a \leq b \leq \infty$*

$$P\left[a \leq \frac{y_n - m_n}{s_n} \leq b\right] \to \Phi(b) - \Phi(a) \quad as \quad n \to \infty$$

Then, if h is differentiable at m and $h'(m) \neq 0$, for any $-\infty \leq c \leq d \leq \infty$

$$P\left[c \leq \frac{h(y_n) - m}{|h'(m)|\, s_n} \leq d\right] \to \Phi(d) - \Phi(c) \quad as \quad n \to \infty$$

Theorem 15 of course applies to the case considered in Theorem 14 where $y_n = \bar{V}_n, m_n = \mu$, and $s_n = \sigma/\sqrt{n}$. In that case it promises that not only is $\bar{V}_n$ approximately normal with mean μ and standard deviation $\sigma/\sqrt{n}$, but also that $h\left(\bar{V}_n\right)$ is approximately normal with mean $h(\mu)$ and standard deviation $|h'(\mu)|\,\sigma/\sqrt{n}$.

2.3 Simulations

Relative to the number of potentially interesting probability and statistical inference problems that might be phrased, the number that can be fruitfully attacked using methods of the previous two sections is quite small. Where exact/theoretical computation is not possible (or practical) modern cheap computing has made stochastic (probabilistic) simulation an effective alternative method of finding probabilities of interest that follow from even quite complicated model assumptions. Here we consider two kinds of modern simulations, those based on iid simulations and those based on so-called Markov chains.

2.3.1 IID Simulations

Proposition 1 already begins to suggest that the basis of all computer-implemented stochastic simulation is the development of a sequential algorithm that at each iteration returns a number between 0 and 1 in such a way that successive realized values "look like" realizations of iid $\mathrm{U}(0,1)$ random variables. A quick look at `Wikipedia` (say under "random number generation") will give the reader some idea of the basic nature of some possible simple algorithms for generation of uniforms.

With a uniform random number generator in hand, one can generate observations from a particular well-known distribution using Proposition 1, or other special tricks appropriate for particular distributions. For example, the so-called **Box-Mueller Method** (of generating pairs of independent standard normal variables) is based on the fact that for U_1 and U_2 iid $\mathrm{U}(0,1)$, the pair

$$Z_1 = (-2\ln U_1)^{\frac{1}{2}}\cos(2\pi U_2) \quad \text{and} \quad Z_2 = (-2\ln U_1)^{\frac{1}{2}}\sin(2\pi U_2)$$

are iid standard normal. Or, it's a fact that for $U_1, U_2, \ldots$ iid $\mathrm{U}(0,1)$ the variables $x_1, x_2, \ldots$ defined by

$$x_i = -\ln U_i$$

are iid Exp(1) and then that

$$y = \begin{cases} 0 & \text{if } x_1 > \lambda \\ \text{the largest } j \text{ such} & \\ \text{that } \sum_{i=1}^{j} x_i \leq \lambda & \text{otherwise} \end{cases}$$

has the Poisson(λ) distribution.

What is of most interest here are not the special tricks that are used with particular distributions, but is rather an interesting and quite general method called the **rejection method** (for generating from even completely non-standard distributions). Suppose that a p-variate $\boldsymbol{x}$ has pdf proportional to a function $q(\boldsymbol{x})$, i.e.

$$f(\boldsymbol{x}) \propto q(\boldsymbol{x})$$

where one can compute values for $q(\boldsymbol{x})$ but need not be able to easily compute values for $f(\boldsymbol{x})$ (e.g., because a normalizing constant is not available in closed

form). Suppose further that there is a p-variate joint density $g(\boldsymbol{x})$ from which it *is* possible to sample, and a known constant $M > 0$ for which

$$Mg(\boldsymbol{x}) \geq q(\boldsymbol{x}) \quad \forall \boldsymbol{x}$$

Then the following will produce a realization $\boldsymbol{x}$ from the density f.

1. Generate $\boldsymbol{x}^*$ from the distribution with density g.
2. Generate $U \sim \mathrm{U}(0,1)$.
3. If
$$MUg(\boldsymbol{x}^*) < q(\boldsymbol{x}^*)$$
then set $x = \boldsymbol{x}^*$. Otherwise return to 1.

In any event, whether by using standard software and preprogrammed random number generators to simulate values or by some personal programming, if one can simulate $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ iid from a distribution of interest, then for a function h of interest, the variables

$$h(\boldsymbol{x}_1), h(\boldsymbol{x}_2), \ldots, h(\boldsymbol{x}_n)$$

are iid with the distribution of $h(\boldsymbol{x})$. An empirical distribution (histogram/relative frequency distribution) of these can be used to approximate the distribution of $h(\boldsymbol{x})$. The law of large numbers implies that for large n the probability is large that the random variable

$$\frac{1}{n}\sum_{i=1}^{n} h(\boldsymbol{x}_i) \tag{6}$$

is close to $\mathrm{E}h(\boldsymbol{x})$. In particular, for h an indicator function, say $h(\boldsymbol{x}) = I[\boldsymbol{x} \in A]$, the sample average (6) is a sample proportion of 1's (indices for which $\boldsymbol{x}_i \in A$), $\mathrm{E}h(\boldsymbol{x}) = P[\boldsymbol{x} \in A]$, and with large probability the sample proportion is close to the actual probability that $\boldsymbol{x} \in A$.

2.3.2 Markov Chain Monte Carlo (MCMC)

One of the genuinely important advances in probability and statistics over the past two decades has been the development of simulation methods that cover a huge variety of distributions for $\boldsymbol{x}$, including in particular cases where (as for the rejection sampling method) a density for $\boldsymbol{x}$ is known only up to a multiplicative constant. (But unlike the situation for rejection sampling, no M and $g(\boldsymbol{x})$ with $Mg(\boldsymbol{x})$ bounding the known form $q(\boldsymbol{x})$ are required.) This is very important for the practical application of Bayes methods where

1. except in the simplest possible situations, exact analytical computations with a posterior density are impossible, and
2. even numerical computation of a normalizing constant for the product of the likelihood and prior densities needed to completely detail the posterior density is typically effectively impossible.

What has enabled this development is the realization that for an average like that in display (6) to approximate $\mathrm{E}h(\boldsymbol{x})$ (for $\boldsymbol{x} \sim f(\boldsymbol{x})$) one doesn't necessarily need to simulate $\boldsymbol{x}_i$ that are *iid* f. There are other models (ways of simulating) that will equally produce histograms of values $\boldsymbol{x}_i$ that approximate the density $f(\boldsymbol{x})$ and for which a simulation average like (6) approximates $\mathrm{E}h(\boldsymbol{x})$. One can employ (Markov Chain simulations or) **Markov Chain Monte Carlo** instead of iid simulations.

The most common forms of MCMC are variants of successive substitution sampling (SSS). SSS algorithms work as follows. Suppose that p-variate $\boldsymbol{x}$ is partitioned into $(k \leq p)$ components that may be of different dimensions, say $\boldsymbol{x}'_1, \boldsymbol{x}'_2, \ldots, \boldsymbol{x}'_k$. One begins a simulation algorithm at some possible value of $\boldsymbol{x}$, say $\boldsymbol{x}^0 = \left(\boldsymbol{x}'^0_1, \boldsymbol{x}'^0_2, \ldots, \boldsymbol{x}'^0_k\right)$ and updates in turn $\boldsymbol{x}'^0_1, \boldsymbol{x}'^0_2, \ldots$, and $\boldsymbol{x}'^0_k$ to produce $\boldsymbol{x}^1 = \left(\boldsymbol{x}'^1_1, \boldsymbol{x}'^1_2, \ldots, \boldsymbol{x}'^1_k\right)$. That is, one moves

$$\left(\boldsymbol{x}'^0_1, \boldsymbol{x}'^0_2, \boldsymbol{x}'^0_3, \ldots, \boldsymbol{x}'^0_k\right) \to \left(\boldsymbol{x}'^1_1, \boldsymbol{x}'^0_2, \boldsymbol{x}'^0_3, \ldots, \boldsymbol{x}'^0_k\right) \to$$
$$\left(\boldsymbol{x}'^1_1, \boldsymbol{x}'^1_2, \boldsymbol{x}'^0_3, \ldots, \boldsymbol{x}'^0_k\right) \to \cdots \to \left(\boldsymbol{x}'^1_1, \boldsymbol{x}'^1_2, \ldots, \boldsymbol{x}'^1_k\right)$$

Then, with $\boldsymbol{x}^j = \left(\boldsymbol{x}'^j_1, \boldsymbol{x}'^j_2, \ldots, \boldsymbol{x}'^j_k\right)$ in hand, one updates in turn $\boldsymbol{x}'^j_1, \boldsymbol{x}'^j_2, \ldots$, and $\boldsymbol{x}'^j_k$ to produce $\boldsymbol{x}^{j+1} = \left(\boldsymbol{x}'^{j+1}_1, \boldsymbol{x}'^{j+1}_2, \ldots, \boldsymbol{x}'^{j+1}_k\right)$. The updating steps are done in random fashion, and two versions of this are in common use. So called Gibbs and so-called Metropolis-Hastings steps are popular.

If $f(\boldsymbol{x})$ is completely known, it can sometimes be possible to find and generate from the conditional density of $\boldsymbol{x}'_l | \boldsymbol{x}'_1, \boldsymbol{x}'_2, \ldots, \boldsymbol{x}'_{l-1}, \boldsymbol{x}'_{l+1}, \boldsymbol{x}'_{l+2}, \ldots, \boldsymbol{x}'_k$. That is, a **Gibbs** algorithm randomly generates

$$\boldsymbol{x}'^{j+1}_l \sim f\left(\boldsymbol{x}'_l | \boldsymbol{x}'^{j+1}_1, \boldsymbol{x}'^{j+1}_2, \ldots, \boldsymbol{x}'^{j+1}_{l-1}, \boldsymbol{x}'^{j}_{l+1}, \boldsymbol{x}'^{j}_{l+2}, \ldots, \boldsymbol{x}'^{j}_k\right)$$

As it turns out, although the $\boldsymbol{x}^j = \left(\boldsymbol{x}'^j_1, \boldsymbol{x}'^j_2, \ldots, \boldsymbol{x}'^j_k\right)$ are not iid (they are instead a realization from a Markov Chain) averages

$$\frac{1}{n}\sum_{j=1}^{n} h\left(\boldsymbol{x}^j\right) \tag{7}$$

from a Gibbs algorithm typically approximate (in LLN fashion) the constant $\mathrm{E}h(\boldsymbol{x})$.

Often (if for no other reason because the normalizing constant is not known) how to sample from the conditionals of $f(\boldsymbol{x})$ is not clear. Then, instead of using Gibbs updates, **M-H** (**Metropolis-Hastings**) stochastic updates can sometimes be employed. One way to think of an M-W algorithm is as an adaptive version of rejection sampling applied one "coordinate" of $\boldsymbol{x}$ at a time.

Suppose that $f(\boldsymbol{x}) \propto q(\boldsymbol{x})$. A M-H update of $\boldsymbol{x}'^j_l \to \boldsymbol{x}'^{j+1}_l$ is accomplished as follows.

1. Let $J_l(\cdot|\boldsymbol{x})$ be a (proposal or jumping) density for a new $\boldsymbol{x}'_l$ (that can be simulated from and can depend upon all the entries of $\boldsymbol{x}$, including

especially a current $\boldsymbol{x}'_l$) and generate a "proposal" for $\boldsymbol{x}'^{j+1}_l$ as

$$\boldsymbol{x}'^{*}_l \sim J_l\left(\cdot|\boldsymbol{x}'^{j+1}_1, \boldsymbol{x}'^{j+1}_2, \ldots, \boldsymbol{x}'^{j+1}_{l-1}, \boldsymbol{x}'^{j}_l, \boldsymbol{x}'^{j}_{l+1}, \ldots, \boldsymbol{x}'^{j}_k\right)$$

2. Compute the "acceptance ratio"

$$r_l^{j+1} = \frac{q\left(\boldsymbol{x}'^{j+1}_1, \boldsymbol{x}'^{j+1}_2, \ldots, \boldsymbol{x}'^{j+1}_{l-1}, \boldsymbol{x}'^{*}_l, \boldsymbol{x}'^{j}_{l+1}, \boldsymbol{x}'^{j}_{l+2}, \ldots, \boldsymbol{x}'^{j}_k\right)}{q\left(\boldsymbol{x}'^{j+1}_1, \boldsymbol{x}'^{j+1}_2, \ldots, \boldsymbol{x}'^{j+1}_{l-1}, \boldsymbol{x}'^{j}_l, \boldsymbol{x}'^{j}_{l+1}, \boldsymbol{x}'^{j}_{l+2}, \ldots, \boldsymbol{x}'^{j}_k\right)} \times \frac{J_l\left(\boldsymbol{x}'^{j}_l|\boldsymbol{x}'^{j+1}_1, \boldsymbol{x}'^{j+1}_2, \ldots, \boldsymbol{x}'^{j+1}_{l-1}, \boldsymbol{x}'^{*}_l, \boldsymbol{x}'^{j}_{l+1}, \boldsymbol{x}'^{j}_{l+2}, \ldots, \boldsymbol{x}'^{j}_k\right)}{J_l\left(\boldsymbol{x}'^{*}_l|\boldsymbol{x}'^{j+1}_1, \boldsymbol{x}'^{j+1}_2, \ldots, \boldsymbol{x}'^{j+1}_{l-1}, \boldsymbol{x}'^{j}_l, \boldsymbol{x}'^{j}_{l+1}, \boldsymbol{x}'^{j}_{l+2}, \ldots, \boldsymbol{x}'^{j}_k\right)} \tag{8}$$

3. and with probability

$$\min\left(1, r_l^j\right)$$

set $\boldsymbol{x}'^{j+1}_l = \boldsymbol{x}'^{*}_l$.

4. Otherwise set $\boldsymbol{x}'^{j+1}_l = \boldsymbol{x}'^{j}_l$.

As for $\boldsymbol{x}^j = \left(\boldsymbol{x}'^{j}_1, \boldsymbol{x}'^{j}_2, \ldots, \boldsymbol{x}'^{j}_k\right)$ from a Gibbs version of SSS, averages (7) from a M-H version "typically" approximate (in LLN fashion) the constant $\mathrm{E}h(\boldsymbol{x})$.

In cases where the ratio of J_l's in display (8) is 1 (roughly speaking, the proposal probability of going from $\boldsymbol{x}'^{j}_l$ to $\boldsymbol{x}'^{*}_l$ is the same as the proposal probability of moving from $\boldsymbol{x}'^{*}_l$ to $\boldsymbol{x}'^{j}_l$ under the same circumstances/values of the other parts of $\boldsymbol{x}$) the M-H algorithm is simplified and is called a **Metropolis** algorithm. A common choice of $J_l(\cdot|\boldsymbol{x})$ is normal with mean $\boldsymbol{x}'_l$ (and some appropriate variance that controls the size of proposed jumps) and this kind of symmetric jumping distribution produces Metropolis algorithms.

While one can of course implement one's own versions of MCMC algorithms, most often existing software like `WinBUG`, `OpenBUGS`, `JAGS`, `STAN`, and `YADAS` is used in routine applications. This software provides both simulations of Markov Chains and summaries of the realizations and diagnostics intended to warn one of possible problems in the simulations.

2.4 Bootstrapping

"Bootstrapping" (the name derived from the English idiom of "pulling oneself up by one's bootstraps") is an another kind of modern application of simulation to the problem of statistical inference. Here we'll describe application of the idea to the "one-sample" statistical inference problem, beginning with a parametric version of the problem.

So to begin, suppose that data available in an inference problem are

$$\boldsymbol{y} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$$

where the $\boldsymbol{x}_i$ are iid according to some distribution $F^{\boldsymbol{\theta}}$ (that is potentially specified by some density $f(\boldsymbol{x}|\boldsymbol{\theta})$). Some statistic (function of the data)

$$T_n(\boldsymbol{y}) = T_n(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$$

is of interest and we'll let its distribution (derived from $F^{\boldsymbol{\theta}}$) be denoted by $G_n^{\boldsymbol{\theta}}$, that is

$$T_n \sim G_n^{\boldsymbol{\theta}}$$

Indeed, we'll suppose that what is of interest is a numerical characteristic of $G_n^{\boldsymbol{\theta}}$, say

$$H\left(G_n^{\boldsymbol{\theta}}\right)$$

(a function of $\boldsymbol{\theta}$ and n through the object $G_n^{\boldsymbol{\theta}}$). Computation of $H\left(G_n^{\boldsymbol{\theta}}\right)$ is a probability problem. Sometimes analytical or numerical calculations can be used. More often, simulation provides a fairly expeditious way of approximating it.

To be explicit about how simulation can be used to solve the probability problem, suppose that one can simulate (in iid fashion) from $F^{\boldsymbol{\theta}}$ to produce values $\boldsymbol{x}^*$. Then one can produce B simulated one-sample data sets of size n

$$\begin{aligned}
\boldsymbol{y}_1^* &= (\boldsymbol{x}_{11}^*, \boldsymbol{x}_{21}^*, \ldots, \boldsymbol{x}_{n1}^*) \\
\boldsymbol{y}_2^* &= (\boldsymbol{x}_{12}^*, \boldsymbol{x}_{22}^*, \ldots, \boldsymbol{x}_{n2}^*) \\
&\vdots \\
\boldsymbol{y}_B^* &= (\boldsymbol{x}_{1B}^*, \boldsymbol{x}_{2B}^*, \ldots, \boldsymbol{x}_{nB}^*)
\end{aligned}$$

and compute the B corresponding simulated values of T_n

$$T_{n1}^* = T_n(\boldsymbol{y}_1^*), T_{n2}^* = T_n(\boldsymbol{y}_2^*), \ldots, T_{nB}^* = T_n(\boldsymbol{y}_B^*)$$

that comprise a simulated random sample of size B from the distribution $G_n^{\boldsymbol{\theta}}$. Let their empirical (relative frequency) distribution be denoted by $G_n^{\boldsymbol{\theta}*}$. Law of large numbers arguments then say that $G_n^{\boldsymbol{\theta}*}$ approximates $G_n^{\boldsymbol{\theta}}$ and (assuming some kind of "continuity" of the function H) one can expect that (in LLN fashion where it is B that is going to infinity)

$$H\left(G_n^{\boldsymbol{\theta}*}\right) \approx H\left(G_n^{\boldsymbol{\theta}}\right)$$

Now suppose that one is not furnished with the value of $\boldsymbol{\theta}$, but still wants to approximate $H\left(G_n^{\boldsymbol{\theta}}\right)$. One might then estimate $\boldsymbol{\theta}$ using (for example) maximum likelihood based on the n observations in $\boldsymbol{y} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$ to produce $\hat{\boldsymbol{\theta}}^{\mathrm{MLE}}$ and apply the simulation method above (generating in iid fashion from $F^{\hat{\boldsymbol{\theta}}^{\mathrm{MLE}}}$) to approximate $H\left(G_n^{\hat{\boldsymbol{\theta}}^{\mathrm{MLE}}}\right)$ with large B. Then (if again the function H is continuous and) one can find theory that says that $\hat{\boldsymbol{\theta}}^{\mathrm{MLE}}$ approximates $\boldsymbol{\theta}$ for large n (and there is such theory soon to be discussed), one can expect that (for large n and B)

$$H\left(G_n^{\hat{\boldsymbol{\theta}}^{\mathrm{MLE}}*}\right) \approx H\left(G_n^{\hat{\boldsymbol{\theta}}^{\mathrm{MLE}}}\right) \approx H\left(G_n^{\boldsymbol{\theta}}\right)$$

(the numerical characteristic of the empirical distribution of the T_n^*'s produced using $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}^{\text{MLE}}$ is approximately the desired value). This is a so-called parametric bootstrap approximation.

An important variant of the bootstrap idea is a "non-parametric" version. This is useful in inference problems where one doesn't make the assumption that the class of possible distributions for $\boldsymbol{x}$ is so simple as to be described by a few numerical parameters (like most small families of the sort studied in Stat 341, e.g. normal and exponential). Rather, one supposes only that F is in some broad class like "the set of continuous distributions with a finite first moment." So making an MLE and using simulation from $F^{\hat{\boldsymbol{\theta}}^{\text{MLE}}}$ is not an option.

What is done instead is to simulate bootstrap samples by drawing $\boldsymbol{x}^*$ values iid from the empirical distribution of the data in hand,

$$\hat{F}_n = \text{a distribution placing probability } \frac{1}{n} \text{ on each } \boldsymbol{x}_i$$

(the relative frequency distribution of the n observations). In effect, this makes every entry of every

$$\begin{aligned}
\boldsymbol{y}_1^* &= (\boldsymbol{x}_{11}^*, \boldsymbol{x}_{21}^*, \ldots, \boldsymbol{x}_{n1}^*) \\
\boldsymbol{y}_2^* &= (\boldsymbol{x}_{12}^*, \boldsymbol{x}_{22}^*, \ldots, \boldsymbol{x}_{n2}^*) \\
&\vdots \\
\boldsymbol{y}_B^* &= (\boldsymbol{x}_{1B}^*, \boldsymbol{x}_{2B}^*, \ldots, \boldsymbol{x}_{nB}^*)
\end{aligned}$$

a random selection (made *with* replacement) from the original data set. Then a LLN type argument implies that for large n

$$\hat{F}_n \approx F$$

If $G_n^{\boldsymbol{F}}$ is the F distribution of T_n and this is a "continuous" function of F, we can then hope that

$$G_n^{\hat{\boldsymbol{F}}_n} \approx G_n^{\boldsymbol{F}}$$

For $G_n^{\hat{\boldsymbol{F}}_n *}$ the empirical distribution of the bootstrap values T_n^*, $G_n^{\hat{\boldsymbol{F}}_n *} \approx G_n^{\hat{\boldsymbol{F}}_n}$ and (again assuming that H is continuous)

$$H\left(G_n^{\hat{\boldsymbol{F}}_n *}\right) \approx H\left(G_n^{\hat{\boldsymbol{F}}_n}\right) \approx H\left(G_n^{\boldsymbol{F}}\right)$$

(the numerical characteristic of the empirical distribution of the T_n^*'s produced using $F = \hat{F}_n$ is approximately the desired value).

3 Likelihood Theory

We introduced the likelihood function (4) in Section 1.2 as the theoretical basis of much of statistical inference. In this section we provide some of the basic theory associated with its use.

3.1 Sufficiency/"(Statistically) Lossless Feature Selection" in Statistical Models

A fundamental issue in data analysis is that of **data reduction/compression** or **feature selection**. In cases where data $\boldsymbol{x}$ are "big" (e.g. consisting of a large $N \times p$ matrix of real values), for various reasons of computation and storage and interchange, etc., it is important to consider working not with $\boldsymbol{x}$ itself but rather with a (possibly multivariate) function of it, say $T(\boldsymbol{x})$. Such a function of the data is typically called a **statistic**. What one wants are statistics that compress data (by virtue of not being 1-1) without degrading one's ability to do statistical inference (based on $T(\boldsymbol{x})$ instead of $\boldsymbol{x}$). Borrowing engineering language, one is interested in "lossless"[1] feature extraction or data compression. The common statistical language is that one seeks **sufficient statistics**.

A sufficient statistic $T(\boldsymbol{x})$ is one such that for purposes of statistical inference, knowing $\boldsymbol{x}$ is no more useful than knowing $T(\boldsymbol{x})$. One formalization of this idea is next.

Definition 16 *A statistic $T(\boldsymbol{x})$ is said to be sufficient for a statistical model (specified by density $f(\boldsymbol{x}|\boldsymbol{\theta})$ or cdf $F_{\boldsymbol{\theta}}(\boldsymbol{x})$) provided for each value t of $T(\boldsymbol{x})$, the conditional distribution of $\boldsymbol{x}|T(\boldsymbol{x}) = t$ does not depend upon $\boldsymbol{\theta}$.*

There are a variety of reasons to argue that this is a good definition of sufficiency. For one thing, even though (unless $T(\boldsymbol{x})$ is 1-1) one may not be able recover $\boldsymbol{x}$ from only the value of $T(\boldsymbol{x})$, knowing only $T(\boldsymbol{x})$ one *can* in theory (without knowing $\boldsymbol{\theta}$) generate a single variable $\boldsymbol{x}^*$ that has the same $\boldsymbol{\theta}$ distribution as $\boldsymbol{x}$ for every $\boldsymbol{\theta}$. The variable is made via the prescription "observe the value t of $T(\boldsymbol{x})$ and simulate $\boldsymbol{x}^*$ from the conditional distribution of $\boldsymbol{x}|T(\boldsymbol{x}) = t$" (that is available for use since it does not depend upon $\boldsymbol{\theta}$).

Related to this is reasoning that says that since "risk" in a statistical decision problem involves averaging over $\boldsymbol{x}$ (and not the specific observed value of it) and one only needs $T(\boldsymbol{x})$ to make $\boldsymbol{x}^*$ with the same $\boldsymbol{\theta}$ distribution as $\boldsymbol{x}$, any risk function available using $\boldsymbol{x}$ is also available using $T(\boldsymbol{x})$. (One can simply plug a simulated $\boldsymbol{x}^*$ into the decision function with the desired risk function.) So it suffices for purposes of statistical decision analysis to restrict attention to decision functions that depend upon $\boldsymbol{x}$ only through a sufficient statistic $T(\boldsymbol{x})$.

Sufficiency is thus an attractive concept, and it is useful to consider how to operationally identify sufficient statistics. Basic tools in that regard are so-called **likelihood ratios**.

Definition 17 *For a statistical model for $\boldsymbol{x}$ specified by a density $f(\boldsymbol{x}|\boldsymbol{\theta})$, the function of two parameters ($\boldsymbol{\theta}'$ and $\boldsymbol{\theta}''$) corresponding to observed data $\boldsymbol{x}$ defined by*

$$\Lambda_{\boldsymbol{x}}\left(\boldsymbol{\theta}', \boldsymbol{\theta}''\right) = \frac{f\left(\boldsymbol{x}|\boldsymbol{\theta}'\right)}{f\left(\boldsymbol{x}|\boldsymbol{\theta}''\right)}$$

[1] This terminology is used in image and signal processing, where one hopes to compress an image or signal without degrading it visually or audibly. Here we're applying the language to the realm of general statistical inference.

will be called the likelihood ratio for $\boldsymbol{\theta}'$ *relative to* $\boldsymbol{\theta}''$.

With this concept available, a way of verifying that $T(\boldsymbol{x})$ is sufficient is to verify that when two values of $\boldsymbol{x}$ share the same value of $T(\boldsymbol{x})$ their likelihood ratios are all the same (across all possible pairs $(\boldsymbol{\theta}', \boldsymbol{\theta}'')$). That is, there is this.

Theorem 18 *A necessary and sufficient condition for* $T(\boldsymbol{x})$ *to be sufficient for a statistical model specified by density* $f(\boldsymbol{x}|\boldsymbol{\theta})$ *is that* $T(\boldsymbol{x}') = T(\boldsymbol{x}'')$ ***implies that***

$$\Lambda_{\boldsymbol{x}'}(\boldsymbol{\theta}', \boldsymbol{\theta}'') = \Lambda_{\boldsymbol{x}''}(\boldsymbol{\theta}', \boldsymbol{\theta}'') \quad \textit{for all pairs of parameters } (\boldsymbol{\theta}', \boldsymbol{\theta}'') \tag{9}$$

It is fairly easy to see that if $\boldsymbol{\theta}_0$ is such that $f(\boldsymbol{x}|\boldsymbol{\theta}_0) > 0 \ \forall \boldsymbol{x}$ the condition (9) is equivalent to a condition that

$$\Lambda_{\boldsymbol{x}'}(\boldsymbol{\theta}, \boldsymbol{\theta}_0) = \Lambda_{\boldsymbol{x}''}(\boldsymbol{\theta}, \boldsymbol{\theta}_0) \quad \forall \boldsymbol{\theta} \tag{10}$$

or equally that

$$\ln f(\boldsymbol{x}|\theta) - \ln f(\boldsymbol{x}'|\theta) = \ln f(\boldsymbol{x}|\theta_0) - \ln f(\boldsymbol{x}'|\theta_0) \quad \forall \boldsymbol{\theta} \tag{11}$$

This last version (11) is that the $\boldsymbol{x}$ and $\boldsymbol{x}'$ log-likelihoods differ only by a constant and has the interpretation that the two log-likelihoods have the same shape. So in this context, Theorem 18 can be phrased as "$T(\boldsymbol{x})$ is sufficient means that when $T(\boldsymbol{x}') = T(\boldsymbol{x}'')$, the $\boldsymbol{x}$ and $\boldsymbol{x}'$ log-likelihoods have the same shape (as functions of $\boldsymbol{\theta}$)."

Beyond the question of whether a statistic is sufficient is the question of whether it might itself be compressed in a statistically lossless way. That idea motivates the concept of a **minimal sufficient statistic**. This is a simple enough idea in rough terms, but very hard to make mathematically precise. (In fact it is impossible to make it precise at the mathematical level of Stat 342.) *Roughly speaking, a minimal sufficient statistic should be one that is sufficient, and for which any non-trivial compression of it fails to be sufficient.* That is, a sufficient $T(\boldsymbol{x})$ should be minimal sufficient if $S(T(\boldsymbol{x}))$ can only be sufficient if S is "essentially 1-1" (meaning something like $\cup_{s \in \text{NS}} S^{-1}(s)$ has $\boldsymbol{\theta}$ probability 0 for all $\boldsymbol{\theta}$, where NS is the set of s for which $S^{-1}(s)$ is not a singleton).

While it is difficult to be completely precise in defining minimal sufficiency, it is helpful to know that it is closely associated with the likelihood ratio ideas. In fact, there is the following (that could be made precise with enough mathematical background).

Proposition 19 *A necessary and sufficient condition for* $T(\boldsymbol{x})$ *to be minimal sufficient for a statistical model specified by density* $f(\boldsymbol{x}|\boldsymbol{\theta})$ *is that* $T(\boldsymbol{x}') = T(\boldsymbol{x}'')$ ***if and only if***

$$\Lambda_{\boldsymbol{x}'}(\boldsymbol{\theta}', \boldsymbol{\theta}'') = \Lambda_{\boldsymbol{x}''}(\boldsymbol{\theta}', \boldsymbol{\theta}'') \quad \textit{for all pairs of parameters } (\boldsymbol{\theta}', \boldsymbol{\theta}'') \tag{12}$$

Notice that the only difference between Theorem 18 and Proposition 19 is the difference between "implies that" and "if and only if."

Again in a context where $\boldsymbol{\theta}_0$ is such that $f(\boldsymbol{x}|\boldsymbol{\theta}_0) > 0\ \forall \boldsymbol{x}$, the condition (12) is equivalent to both (10) and (11) and Proposition 19 can be rephrased as "$T(\boldsymbol{x})$ is minimal sufficient means that $T(\boldsymbol{x}') = T(\boldsymbol{x}'')$ exactly when the $\boldsymbol{x}$ and $\boldsymbol{x}'$ log-likelihoods have the same shape (as functions of $\boldsymbol{\theta}$)" or "The shape of the log-likelihood is a minimal sufficient statistic."

One simple corollary of Proposition 19 concerns models where the set of possible parameter values is finite, consisting say of $\{1, 2, \ldots, K\}$ as in a K-class classification problem.

Corollary 20 *In a statistical model specified by density $f(\boldsymbol{x}|\theta)$ for $\theta \in \{1, 2, \ldots, K\}$ where $f(\boldsymbol{x}|1) > 0\ \forall \boldsymbol{x}$, the vector of $K-1$ likelihood ratios*

$$T(\boldsymbol{x}) = \left(\Lambda_{\boldsymbol{x}}(2,1), \Lambda_{\boldsymbol{x}}(3,1), \ldots, \Lambda_{\boldsymbol{x}}(K,1)\right)$$

is a minimal sufficient statistic.

Corollary 20 implies that no matter how complicated are data $\boldsymbol{x}$ in a K-class classification problem, there will typically be a $(K-1)$-dimensional sufficient statistic.

3.2 Fisher Information

The likelihood function not only provides guidance in the search for lossless data compressions/reductions, it also provides the basis for making a quantitative assessment of "how much information" about a parameter $\boldsymbol{\theta}$ is carried by an observation $\boldsymbol{x} \sim f(\boldsymbol{x}|\boldsymbol{\theta})$. We proceed to give an overview of so-called **Fisher information**. We do so in the case where the parameter θ is real-valued. Everything here generalizes to cases where $\boldsymbol{\theta} \in \Re^k$, but that development involves matrix notation and multivariate calculus that is better left to graduate-level treatments of the subject. Further, all that follows depends for precise statements and proofs on technical assumptions about the nature of $f(\boldsymbol{x}|\theta)$ (conditions on partial derivatives and the ability to interchange the order of differentiation and expectations in some expressions) that we will suppress (but that hold in many standard models), again leaving precise development to graduate-level treatments.

The loglikelihood function

$$\ln f(\boldsymbol{x}|\theta)$$

is a random function of the parameter θ (different observed values give different functions). In rough terms, one hopes that when $\boldsymbol{x} \sim f(\boldsymbol{x}|\theta_0)$ the log-likelihood function will tend to be big near θ_0 and small away from θ_0. Ideally, the log-likelihood function will tend to be very peaked with a derivative that averages to 0 at θ_0 when $\boldsymbol{x} \sim f(\boldsymbol{x}|\theta_0)$. This kind of thinking leads to the definition of Fisher Information.

We'll call the random derivative function for the log-likelihood

$$\frac{\partial}{\partial\theta}\ln f\left(\boldsymbol{x}|\theta\right)=\frac{\frac{\partial}{\partial\theta}f\left(\boldsymbol{x}|\theta\right)}{f\left(\boldsymbol{x}|\theta\right)}$$

the **score function**. Typically

$$\mathrm{E}_{\theta_0}\left(\frac{\partial}{\partial\theta}\ln f\left(\boldsymbol{x}|\theta\right)\Big|_{\theta=\theta_0}\right)=0$$

(when $\boldsymbol{x}\sim f\left(\boldsymbol{x}|\theta_0\right)$ the average score function at θ_0 is 0).

Definition 21 *The θ_0 variance of the score function at θ_0,*

$$I_{\boldsymbol{x}}\left(\theta_0\right)=Var_{\theta_0}\left(\frac{\partial}{\partial\theta}\ln f\left(\boldsymbol{x}|\theta\right)\Big|_{\theta=\theta_0}\right)$$

*is called the **Fisher information** in $\boldsymbol{x}$ about the parameter θ at the value $\theta=\theta_0$.*

The Fisher information at θ_0 is big when the log-likelihood tends to be very steep/peaked when $\boldsymbol{x}\sim f\left(\boldsymbol{x}|\theta_0\right)$.

The Fisher information has a number of interesting and useful properties. For one thing, it is typically also the *negative average curvature of the log-likelihood*. That is, under mild conditions one can make the next proposition precise.

Proposition 22 *Under appropriate conditions*

$$I_{\boldsymbol{x}}\left(\theta_0\right)=-E_{\theta_0}\left(\frac{\partial^2}{\partial\theta^2}\ln f\left(\boldsymbol{x}|\theta\right)\Big|_{\theta=\theta_0}\right)$$

The facts that 1) a joint density for two independent variables is a product of their marginal densities and 2) the logarithm function turns products into sums immediately implies that Fisher information is additive for independent data vectors.

Theorem 23 *For $\boldsymbol{x}$ and $\boldsymbol{y}$ independent $\forall\theta$,*

$$I_{(\boldsymbol{x},\boldsymbol{y})}\left(\theta\right)=I_{\boldsymbol{x}}\left(\theta\right)+I_{\boldsymbol{y}}\left(\theta\right)$$

There is no intent in Theorem 23 to restrict to cases where $\boldsymbol{x}$ and $\boldsymbol{y}$ are identically distributed. That is we may have , $\boldsymbol{x}\sim f_1\left(\boldsymbol{x}|\theta\right)$ and $\boldsymbol{y}\sim f_2\left(\boldsymbol{y}|\theta\right)$ where the two densities f_1 and f_2 are different.

Fisher information also "behaves properly" under transformations of data and has a close connection to sufficiency.

Theorem 24 *For any statistic* $T(\boldsymbol{x})$

$$I_{T(\boldsymbol{x})}(\theta) \leq I_{\boldsymbol{x}}(\theta) \ \forall\theta$$

(with equality when T *is 1-1).*

(Data compression cannot increase the amount of information in a data set!)

Theorem 25 *For a statistic* $T(\boldsymbol{x})$

$$I_{T(\boldsymbol{x})}(\theta) = I_{\boldsymbol{x}}(\theta) \ \forall\theta$$

exactly when $T(\boldsymbol{x})$ *is sufficient.*

A final note on this subject concerns an interesting relationship between Fisher information and the variance of any unbiased estimator of θ.

Theorem 26 *(The Cramér-Rao Lower Bound) Any statistic* $T(\boldsymbol{x})$ *for which* $E_\theta T(\boldsymbol{x}) = \theta \ \forall\theta$ *has*

$$Var_\theta T(\boldsymbol{x}) \geq \frac{1}{I_{\boldsymbol{x}}(\theta)} \ \forall\theta$$

Theorem 26 says that no unbiased estimator of θ has variance smaller than the reciprocal of the Fisher information. In some problems, one can identify an unbiased estimator that achieves this lower bound and thus identify an estimator that has the best SEL risk function *among those for unbiased estimators.* (SEL risk is mean squared error, is variance for unbiased estimators.)

3.3 (Large Sample) Maximum Likelihood Theory

Maximum likelihood is without doubt the most popular and effective non-Bayesian method of estimation. In the statistical folklore it is held to be typically "optimal." What can be proved about it in precise terms is less than its reputation. Part of the difficulty in studying maximum likelihood is that in general a likelihood need not have any maximum, and if it does have a maximum it's possible for it to have multiple maxima. Further, general theory about maximizers is harder than theory about, say, solutions to the "likelihood equation"

$$\frac{\partial}{\partial\theta}\ln f(\boldsymbol{x}|\theta) = 0 \tag{13}$$

and precise versions of even the latter are built on technical assumptions about $f(\boldsymbol{x}|\theta)$ that are again better left to graduate-level expositions. What we are going to do here is

1. restrict (as in the exposition of Fisher information) to cases where the parameter θ is real-valued,
2. suppress (as in the exposition of Fisher information) the precise assumptions under which the results alluded to here hold,

3. consider iid (one-sample) models as the archetype of what can sometimes be proved more generally, and

4. essentially "assume away" the difficulties of connecting maximization to solution of the likelihood equation by presuming that as n increases the probability that the log-likelihood has a single maximum at an interior point of the parameter space goes to 1 (so that the single solution to equation (13) is in fact a unique MLE).

So we consider a situation where one has data $\boldsymbol{x} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)$ and the $\boldsymbol{y}_i$ are iid according to some "nice" $f(\boldsymbol{y}|\theta)$. Then (abusing notation by letting f stand for different functions on the right and left below) the density for $\boldsymbol{x}$ is

$$f(\boldsymbol{x}|\theta) = \prod_{i=1}^{n} f(\boldsymbol{y}_i|\theta)$$

and the log-likelihood is

$$\ln f(\boldsymbol{x}|\theta) = \sum_{i=1}^{n} \ln f(\boldsymbol{y}_i|\theta)$$

We'll suppose that $\hat{\theta}_n^{\text{MLE}}(\boldsymbol{x})$ solves the likelihood equation (at least with probability going to 1 and n goes to infinity).

Taking a cue from the Fisher information discussion, we should expect that since $I_{\boldsymbol{x}}(\theta) = nI_{\boldsymbol{y}}(\theta) \to \infty$ as $n \to \infty$ the θ_0 probability that the log-likelihood gets very peaked very near θ_0 should go to 1. That suggests that with large θ_0 probability $\hat{\theta}_n^{\text{MLE}}(\boldsymbol{x})$ will be close to θ_0. That is, under appropriate conditions one can prove a precise version of the following.

Proposition 27 *For every $\epsilon > 0$*

$$P_{\theta_0}\left[\left|\hat{\theta}_n^{MLE}(\boldsymbol{x}) - \theta_0\right| > \epsilon\right] \to 0 \quad as \quad n \to \infty$$

Proposition 27 promises that the θ_0 distribution of the MLE piles up at θ_0 with increasing sample size. That property is sometimes called the "consistency" of the MLE.

Just as in Section 2.2 it was useful to consider the approximate shape of the distribution of the sample mean, it is useful to consider the approximate shape of the distribution of $\hat{\theta}_n^{\text{MLE}}(\boldsymbol{x})$. As it turns out (essentially because of a central limit theorem being applicable in light of the fact that the log-likelihood is a sum of iid variables for each θ) one can prove that properly normalized, $\hat{\theta}_n^{\text{MLE}}(\boldsymbol{x})$ is approximately standard normal.

Proposition 28 *For any $-\infty \leq a \leq b \leq \infty$*

$$P_{\theta_0}\left[a \leq \frac{\hat{\theta}_n^{MLE}(\boldsymbol{x}) - \theta_0}{1/\sqrt{nI_{\boldsymbol{y}}(\theta_0)}} \leq b\right] \to \Phi(b) - \Phi(a) \quad as \quad n \to \infty$$

Taken together, Propositions 27 and 28 are support for the statistical folklore that "usually" MLEs are consistent and approximately normal. In Proposition 28 the value θ_0 serves as the center of the approximating distribution and $1/\sqrt{nI_{\boldsymbol{y}}(\theta_0)}$ serves as the standard deviation of the approximating distribution. Roughly, "The MLE is approximately $\mathrm{N}(\theta_0, 1/nI_{\boldsymbol{y}}(\theta_0))$."

Proposition 28 describes the behavior of $\hat{\theta}_n^{\mathrm{MLE}}(\boldsymbol{x})$ but doesn't directly facilitate statistical inference. Two corollaries of it *can* be used to support the making of confidence intervals for θ.

Corollary 29 *For any* $-\infty \leq a \leq b \leq \infty$

$$P_{\theta_0}\left[a \leq \frac{\hat{\theta}_n^{MLE}(\boldsymbol{x}) - \theta_0}{1/\sqrt{nI_{\boldsymbol{y}}\left(\hat{\theta}_n^{MLE}(\boldsymbol{x})\right)}} \leq b\right] \to \Phi(b) - \Phi(a) \quad as \quad n \to \infty$$

Corollary 30 *For*

$$\hat{I}_{\boldsymbol{x}}\left(\hat{\theta}_n^{MLE}(\boldsymbol{x})\right) = -\frac{\partial^2}{\partial\theta^2} \ln f(\boldsymbol{x}|\theta)\Big|_{\theta=\hat{\theta}_n^{MLE}(\boldsymbol{x})}$$

and any $-\infty \leq a \leq b \leq \infty$

$$P_{\theta_0}\left[a \leq \frac{\hat{\theta}_n^{MLE}(\boldsymbol{x}) - \theta_0}{1/\sqrt{\hat{I}_{\boldsymbol{x}}\left(\hat{\theta}_n^{MLE}(\boldsymbol{x})\right)}} \leq b\right] \to \Phi(b) - \Phi(a) \quad as \quad n \to \infty$$

Using $1/\sqrt{nI_{\boldsymbol{y}}\left(\hat{\theta}_n^{\mathrm{MLE}}(\boldsymbol{x})\right)}$ in Corollary 29 as an approximation for $1/\sqrt{nI_{\boldsymbol{y}}(\theta_0)}$ in Proposition 28 is often called employing "expected information." In contrast, using $1/\sqrt{\hat{I}_{\boldsymbol{x}}\left(\hat{\theta}_n^{\mathrm{MLE}}(\boldsymbol{x})\right)}$ in Corollary 30 is often called employing "observed information."

It follows directly from Corollaries 29 and 30 that for z the upper $\alpha/2$ point of the standard normal distribution, both

$$\hat{\theta}_n^{\mathrm{MLE}}(\boldsymbol{x}) \pm z\frac{1}{\sqrt{nI_{\boldsymbol{y}}\left(\hat{\theta}_n^{\mathrm{MLE}}(\boldsymbol{x})\right)}} \tag{14}$$

and

$$\hat{\theta}_n^{\mathrm{MLE}}(\boldsymbol{x}) \pm z\frac{1}{\sqrt{\hat{I}_{\boldsymbol{x}}\left(\hat{\theta}_n^{\mathrm{MLE}}(\boldsymbol{x})\right)}} \tag{15}$$

can serve as approximately $(1-\alpha)\,100\%$ two-sided confidence limits for θ for large n. It is generally held that typically limits (15) are more reliable (in terms of producing intervals with actual coverage probability matching the approximate confidence level) than limits (14).

3.4 Likelihood Ratio Testing Theory

Classical statistical inference theory is comprised of "estimation" theory and "testing" theory. We've discussed some uses of the likelihood in estimation. We now consider some implications of likelihood ideas for testing. Statistical "testing" is concerned with deciding between two "hypotheses," the first (the "null" hypothesis) that $\theta \in \Theta_0$ and the second (the "alternative" hypothesis) that $\theta \in \Theta_1$ for $\Theta_0 \cap \Theta_1 = \emptyset$.

3.4.1 Simple versus Simple Testing

One case of hypothesis testing is equivalent to 2-class classification. That is, we first consider a decision problem where $\theta \in \{0, 1\}$ under 0-1 loss. We'll suppose that the density $f(\boldsymbol{x}|\theta)$ specifies the distribution of the observable. In this context, for a decision function $a(\boldsymbol{x})$, the risk

$$R(0) = \mathrm{E}_0 I[a(\boldsymbol{x}) = 1] = P_\theta[a(\boldsymbol{x}) = 1]$$

is usually denoted by α and called the "**Type I" error probability**, or sometimes, the **size** of the test. The risk

$$R(1) = \mathrm{E}_1 I[a(\boldsymbol{x}) = 0] = P_\theta[a(\boldsymbol{x}) = 0]$$

is usually denoted by β and called the "**Type II" error probability**. Related to the Type II error probability is

$$1 - \beta = 1 - R(1)$$

that is sometimes called the **power** of the test. The eventuality that $a(\boldsymbol{x}) = 1$ is termed "rejecting the null hypothesis."

Corollary 20 says that the likelihood ratio

$$\Lambda_{\boldsymbol{x}}(1, 0) = \frac{f(\boldsymbol{x}|1)}{f(\boldsymbol{x}|0)}$$

is minimal sufficient in this small (two distribution) statistical model. That means that all sensible statistical decision rules are functions of $\Lambda_{\boldsymbol{x}}(1, 0)$. In fact, in this problem all sensible tests/classifiers/decision rules $a(\boldsymbol{x})$ have for some non-negative constant c,

$$\begin{array}{ll} a(\boldsymbol{x}) = 1 & \text{if } \Lambda_{\boldsymbol{x}}(1, 0) > c \\ a(\boldsymbol{x}) = 0 & \text{if } \Lambda_{\boldsymbol{x}}(1, 0) < c \end{array} \tag{16}$$

There are several ways in which this can be made precise. One possible statement is next.

Theorem 31 *For any prior distribution specified by $g(0)$ and $g(1) = 1 - g(0)$, $a(\boldsymbol{x})$ is Bayes optimal (has minimum Bayes risk) if and only if*

$$a(\boldsymbol{x}) = 1 \quad \textit{if } \Lambda_{\boldsymbol{x}}(1, 0) > \frac{g(0)}{g(1)}$$

$$a(\boldsymbol{x}) = 0 \quad \textit{if } \Lambda_{\boldsymbol{x}}(1, 0) < \frac{g(0)}{g(1)}$$

A second statement concerns not Bayes risk, but rather the risk function $R(\theta)$.

Theorem 32 *(Neyman-Pearson Lemma) if $a(\boldsymbol{x})$ is of the form (16) then for any other decision function $a^*(\boldsymbol{x})$*

$$R_{a^*}(0) \leq R_a(0) \implies R_{a^*}(1) \geq R_a(1)$$

Theorem 32 says that it's impossible to find a test that uniformly improves upon an $a(\boldsymbol{x})$ of form (16). Somewhat differently put, no test with size as good as (as small as) that of $a(\boldsymbol{x})$ has better power. $a(\boldsymbol{x})$ is a **most powerful** test of its size.

3.4.2 (Large Sample) Theory for Likelihood Ratio Tests of a Point Null Hypothesis and Corresponding Confidence Sets

Another set of theoretical results for testing concerns the large sample properties of a kind of likelihood ratio test in statistical models more complex than the simple versus simple case just discussed. To give the sense of what kind of results exit, here we'll consider the modeling of Section 3.3, and in that context, the case where $\Theta_0 = \{\theta_0\}$ and Θ_1 consists of the balance of the parameter space (if θ is assumed to lie in $\Theta \subset \Re$ then $\Theta_1 = \Theta \setminus \Theta_0$). Here the null hypothesis ($\mathrm{H_0}$:$\theta = \theta_0$) is a "point" null hypothesis and the alternative ($\mathrm{H_a}$:$\theta \neq \theta_0$) is a "composite" hypothesis. We'll interpret the "action" $a = 0$ to mean that one decides in favor of the (point) null hypothesis and the action $a = 1$ to mean that one decides (against the point null hypothesis) in favor of the (composite) alternative hypothesis.

There are many possible likelihood ratios

$$\Lambda_{\boldsymbol{x}}(\theta, \theta_0) \quad \text{for} \quad \theta \neq \theta_0$$

that could be considered in assessing the plausibility of the null hypothesis. A possible way of using these to make a decision about θ is to compare each one to some non-negative constant and reject the null hypothesis if any of them is larger than the constant. While this method of operation cannot be shown to always be "optimal," it typically does yield sensible tests. And there is theory regarding how to choose the constant to get a desired approximate size for the test.

To elaborate, define

$$\lambda_n(x) = \ln f\left(\boldsymbol{x}|\hat{\theta}_n^{\mathrm{MLE}}(\boldsymbol{x})\right) - \ln f(\boldsymbol{x}|\theta_0)$$

This is the natural logarithm of $\Lambda_{\boldsymbol{x}}(\theta, \theta_0)$ for the value of θ producing the largest likelihood ratio relative to θ_0. So using this quantity as a building block for defining

$$a(\boldsymbol{x}) = I[\lambda_n(x) > c] \tag{17}$$

for some constant c is equivalent to choosing to reject the null hypothesis if there is some likelihood ratio relative to θ_0 that is large. As it turns out, the kind of arguments that produce the theory in Section 3.3 also produce theory for approximating the size of a likelihood ratio test (17). That is, under appropriate conditions one can make the following precise.

Proposition 33 *For $c > 0$ and F the χ^2_1 cdf,*

$$P_{\theta_0}\left[2\lambda_n(x) > c\right] \to 1 - F(c) \quad \textit{as} \quad n \to \infty$$

or equivalently

$$P_{\theta_0}\left[\lambda_n(x) > \frac{c}{2}\right] \to 1 - F(c) \quad \textit{as} \quad n \to \infty$$

Proposition 33 says that under the null hypothesis, $2\lambda_n(x)$ is approximately χ^2_1, and implies that one can choose "cut-off" values c to (approximately) control the size of the likelihood ratio test (17). For example, since the upper 5% point of the χ^2_1 distribution is roughly 3.841, rejecting the null hypothesis if $\lambda_n(x)$ exceeds 1.92 will for large n produce a test of size approximately .05.

It is an interesting and very useful fact that Proposition 33 not only provides a way to develop a test of $\mathrm{H}_0 : \theta = \theta_0$ for a particular θ_0, but also provides a way to make a confidence set for θ. That is, for c the upper α point of the χ^2_1 distribution

$$P_{\theta_0}\left[\lambda_n(x) > \frac{c}{2}\right] \approx \alpha$$

implies that

$$P_{\theta_0}\left[\ln f\left(\boldsymbol{x}|\hat{\theta}_n^{\mathrm{MLE}}(\boldsymbol{x})\right) - \ln f(\boldsymbol{x}|\theta_0) > \frac{c}{2}\right] \approx \alpha$$

i.e. that

$$P_{\theta_0}\left[\ln f\left(\boldsymbol{x}|\hat{\theta}_n^{\mathrm{MLE}}(\boldsymbol{x})\right) - \frac{c}{2} < \ln f(\boldsymbol{x}|\theta_0)\right] \approx 1 - \alpha$$

so that for large n, **the set of θ's with log-likelihood within $c/2$ of the maximum functions as an approximately $(1-\alpha)\times 100\%$ confidence set for θ.**

4 Classification Problems

In the balance of these notes we revisit the basic prediction problems introduced in Section 1.1, this time from a more practical point of view. We began in Section 1.1 supposing that one has a completely specified probability model for $(y, \boldsymbol{x})$. But except as a thought experiment, there is no magic by which one is given $f(y, \boldsymbol{x})$. Rather, in practice what one typically has is a sample of "training data" (call it $\mathcal{T}$)

$$(y_1, \boldsymbol{x}_1), (y_2, \boldsymbol{x}_2), \ldots, (y_N, \boldsymbol{x}_N)$$

from which one must fashion effective approximations to the optimal predictors, almost always under the assumption that the training data are iid from the

distribution specified by $f(y, \boldsymbol{x})$. We begin discussion of this enterprise with the case of 0-1 loss 2-class classification in this section and then turn to SEL prediction of a continuous y in the final section.

4.1 Approximating Optimal Classifiers Based on "Training Data," Nearest Neighbor Classifiers, Predictor Complexity, Over-fit, and Cross-Validation

Based on $f(y, \boldsymbol{x})$, we know that for 0-1 loss 2-class classification

$$\begin{aligned}
\hat{y}^{\text{opt}}(\boldsymbol{x}^*) &= \underset{\hat{y}\in\{0,1\}}{\arg\max} P[y = \hat{y}|\boldsymbol{x}^*] \\
&= \underset{\hat{y}\in\{0,1\}}{\arg\max} \frac{P[y = \hat{y}] f(\boldsymbol{x}^*|\hat{y})}{P[y = 0] f(\boldsymbol{x}^*|0) + P[y = 1] f(\boldsymbol{x}^*|1)} \\
&= \underset{\hat{y}\in\{0,1\}}{\arg\max} P[y = \hat{y}] f(\boldsymbol{x}^*|\hat{y}) \\
&= \underset{\hat{y}\in\{0,1\}}{\arg\max} P[y = \hat{y}] f(\boldsymbol{x}^*|\hat{y}) \Delta\boldsymbol{x} \\
&\approx \underset{\hat{y}\in\{0,1\}}{\arg\max} P[y = \hat{y} \text{ and } \boldsymbol{x} \text{ is in a region of small volume } \Delta\boldsymbol{x} \text{ including } \boldsymbol{x}^*]
\end{aligned}$$

The third and last lines of this string of equalities/approximations suggest at least 2 ways that one might think of approximating the optimal predictor based on training data. Motivated by the third line, one might think of somehow estimating the conditional densities $f(\boldsymbol{x}^*|0)$ and $f(\boldsymbol{x}^*|1)$ (with, say, $\hat{f}(\boldsymbol{x}^*|0)$ and $\hat{f}(\boldsymbol{x}^*|1)$) and using the sample fractions of 0's and 1's in the training set (call those $\hat{p}_0$ and $\hat{p}_1$) to produce a predictor

$$\hat{y}^{\text{opt}}(\boldsymbol{x}^*) = \underset{\hat{y}\in\{0,1\}}{\arg\max} \hat{p}_{\hat{y}} \hat{f}(\boldsymbol{x}^*|\hat{y}) \tag{18}$$

Based on the last line of the string of equalities/approximations for $\hat{y}^{\text{opt}}(\boldsymbol{x}^*)$, upon defining a small region containing $\boldsymbol{x}^*$ one might compare the fraction of the training set with $y = 0$ and $\boldsymbol{x}$ in the region to the fraction with $y = 1$ and $\boldsymbol{x}$ in the region, classifying to $y = 1$ when the second is larger than the first. To be slightly more precise, a so-called "k-nearest neighbor" classifier is produced as follows. For

$$\mathcal{N}_k(\boldsymbol{x}) = \{k \text{ points } \boldsymbol{x}_i \text{ from the training set closest to } \boldsymbol{x}\} \tag{19}$$

let

$$\hat{y}^{k\text{nn}}(\boldsymbol{x}^*) = I\left[\sum_{\boldsymbol{x}_i\in\mathcal{N}_k(\boldsymbol{x}^*)} I[y_i = 1] > \sum_{\boldsymbol{x}_i\in\mathcal{N}_k(\boldsymbol{x}^*)} I[y_i = 0]\right] \tag{20}$$

This predictor is based on letting the "small region" containing $\boldsymbol{x}^*$ be (adaptively in $\boldsymbol{x}^*$) of "size" just big enough to include k points $\boldsymbol{x}_i$ from the training

set[2].

It's fairly easy to see that as k decreases, the flexibility/complexity of the classifier (20) increases (1-nearest neighbor classifiers are far more flexible than N-nearest neighbor classifiers that always classify to the y most frequent in the training set). So a natural question is how one should choose k, or indeed how one might choose between k's for classifiers (20) and versions of classifiers (18).

A first idea is to judge the effectiveness of a predictor/classifier $\hat{y}(\boldsymbol{x})$ on the basis of its training set error rate

$$\overline{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} I\left[y_i \neq \hat{y}(\boldsymbol{x}_i)\right] \tag{21}$$

This will essentially always decrease with increased predictor complexity and thus point in favor of the most complex/flexible predictor. But if one holds back (from use in predictor development) a random part of a large training set (to give a "test set" $\widetilde{\mathcal{T}}$) and instead of the training error rate (21) considers a test set error rate

$$\widetilde{\text{err}} = \frac{1}{\left|\widetilde{\mathcal{T}}\right|}\sum_{(y,\boldsymbol{x})\in\widetilde{\mathcal{T}}} I\left[y \neq \hat{y}(\boldsymbol{x})\right]$$

a different picture typically emerges. $\widetilde{\text{err}}$ decreases with increased complexity *up to a point*, but thereafter *increases with increased complexity.* In a related way, for $\hat{y}$ built on a training set of size N a theoretical error rate

$$\text{Err} = \text{E}I\left[y \neq \hat{y}(\boldsymbol{x})\right]$$

where the averaging is done over the iid size-N training set used to develop $\hat{y}$ and another ($(N+1)$st) independent observation from the distribution specified by $f(y, \boldsymbol{x})$, Err first decreases with complexity and then increases.

This is the phenomenon of "over-fit." Basically one doesn't have enough data[3] for use in making predictors to support the effective use of classifiers more complex than ones with the smallest test set (or theoretical) error rate. Excessive complexity can produce huge increases in error rates when a classifier is applied to "new" data (even when they are generated by the same mechanism used to generate the training set). So choice of a practical classification methodology requires some way of assessing a reliable test set error rate. The best available technology for this is so-called **cross-validation**.

[2] We should note here that as a practical matter, the notion of "nearness" (measured presumably in terms of $\Re^p$ distance) really only makes sense if the p coordinates of $\boldsymbol{x}$ are "on the same scale"/in the same units. One way to enforce this is to standardize the coordinates of $\boldsymbol{x}$ by subtracting training set sample means and dividing by training set sample standard deviations. See Section 5.2.1 for a more detailed description of what is intended by this.

[3] Even "large" N doesn't allow for ignoring this issue. The reason is that *for purposes of predictor development* real data sets are essentially always "small." Where p predictors are involved ($\boldsymbol{x} \in \Re^p$) the "curse of dimensionality" implies that even for moderate p, data sets of reasonable size are relatively sparsely distributed. $\Re^p$ is simply huge. There are too many ways for inputs $\boldsymbol{x} \in \Re^p$ to differ for them to "pile up" anywhere.

K-fold cross-validation (the K here has nothing to do with "k"-nearest neighbors) proceeds by first randomly breaking a training set into K roughly-equal-sized subsets $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_K$ and for each, building a predictor $\hat{y}^k(\boldsymbol{x})$ using all training vectors **except** those in "fold" k, $\mathcal{T}_k$. The cross-validation error is then

$$CV = \frac{1}{N} \sum_{k=1}^{K} \sum_{\substack{i \text{ s.t.} \\ (y_i, \boldsymbol{x}_i) \in \mathcal{T}_k}} I\left[\hat{y}^k(\boldsymbol{x}_i) \neq y_i\right]$$

One uses the predictor made without case i to predict y for case i, and finds a corresponding error rate across the training set. This kind of cross-validation error rate can then be used to choose between different predictors/to identify appropriate predictor complexity. The statistical folklore is that the choice of $K = 5$ or $K = 10$ is often effective in cross-validation.[4]

4.2 Bootstrapping Basic Classifiers, Out-of-Bag Error Rates, and "Bagging"/"Majority Votes"

Another approach to preventing over-fit in a predictor is related to the nonparametric bootstrapping ideas of Section 2.4. The fundamental issue in assessing the effectiveness of a predictor is that one really needs a test set different from the training set. A way of proceeding might be to make some number B bootstrap versions of the training set, say

$$\mathcal{T}_1^*, \mathcal{T}_2^*, \ldots, \mathcal{T}_B^*$$

Each of these is a random sample made *with replacement* from $\mathcal{T}$,

$$\mathcal{T}_b^* = \left\{(y, \boldsymbol{x})_{b1}^*, (y, \boldsymbol{x})_{b2}^*, \ldots, (y, \boldsymbol{x})_{bN}^*\right\}$$

It's an easy calculation to see that for N of any appreciable size, a bootstrap sample $\mathcal{T}_b^*$ will fail to include a (random) fraction of about $e^{-1} \approx .37$ of the N training cases. That suggests that for a predictor $\hat{y}^{*b}(\boldsymbol{x})$ made using the bootstrap sample $\mathcal{T}_b^*$, the value

$$\overline{\text{err}}^{*b} = \frac{1}{|\{i \,|\, (y_i, \boldsymbol{x}_i) \notin \mathcal{T}_b^*\}|} \sum_{\substack{i \text{ s.t.} \\ (y_i, \boldsymbol{x}_i) \notin \mathcal{T}_b^*}} I\left[y_i \neq \hat{y}^{*b}(\boldsymbol{x}_i)\right]$$

is a plausible estimator of Err. In fact, a weighted average of these (across b)

$$OOB = \frac{1}{\sum_{b=1}^{B} |\{i \,|\, (y_i, \boldsymbol{x}_i) \notin \mathcal{T}_b^*\}|} \sum_{b=1}^{B} \sum_{\substack{i \text{ s.t.} \\ (y_i, \boldsymbol{x}_i) \notin \mathcal{T}_b^*}} I\left[y_i \neq \hat{y}^{*b}(\boldsymbol{x}_i)\right]$$

[4] To careful here, note that the sets $\mathcal{T}_k$ are not of size N, but rather only of size $\left(1 - \frac{1}{K}\right) N$. So technically, cross-validation is a means of approximating error rates not for the actual training set size N, but the smaller size $\left(1 - \frac{1}{K}\right) N$. The implicit presumption is thus that comparisons between methods for the smaller training set size carry over at least in terms of order of preference (if not absolute size of apparent error rate) to size N.

can be called a bootstrap (or **out-of-bag**) error rate and be used to guide choice of predictor/classifier.[5]

There is second way in which the bootstrap idea can be used in classifier development, and that is not so much to estimate an error rate as to define a version of a classifier that is itself resistant to over-fitting. Each bootstrap version of a basic predictor $\hat{y}^{*b}(\boldsymbol{x})$ is built on a bootstrap sample that in some sense is representative of the training sample, *but not the whole training sample*. A plausible way to combine these bootstrap predictors to make a new one is through "majority voting," where

$$\hat{y}^{\text{boot}}(\boldsymbol{x}) = I\left[\sum_{b=1}^{B} I\left[\hat{y}^{*b}(\boldsymbol{x}) = 1\right] > \sum_{b=1}^{B} I\left[\hat{y}^{*b}(\boldsymbol{x}) = 0\right]\right]$$

This approach to combining bootstrapped versions of a predictor is sometimes known as "**bootstrap aggregation**" or "**bagging**." The intuition is that the fact that every training case is missing from about 37% of all bootstrap samples means that it will be hard for a single training case or even small set of such cases to locally (in $\boldsymbol{x}$) dominate the form of $\hat{y}^{\text{boot}}(\boldsymbol{x})$ to the effect of causing over-fit. The majority voting by bootstrap versions of $\hat{y}$ tends to "smooth" the basic form, reducing complexity, and the out-of-bag error rate serves as a direct estimate of the error rate of a bagged predictor.

4.3 Practical Basic Classification Methods

Here we provide some introduction to the most-used basic methods of 2-class classification.

4.3.1 Nearest Neighbor Rules

We have already used the $k$nn classification idea to introduce the basic concepts of practical classification in Section 4.1. Indeed, the $k$nn rules are (at least for small-dimensional input $\boldsymbol{x}$ and substantial N) an important tool in real classification problems.

4.3.2 (Binary) Decision/Classification Trees and Random Forests

Another important practical classification methodology is built on a kind of "tree" classifier. This is primarily based on a "forward-selection"/"greedy" algorithm for inventing predictions constant on p-dimensional rectangles by successively looking for an optimal binary split of a single one of an existing set of "rectangles" in $\Re^p$. For example, with $p = 2$, one begins with a rectangle

[5] Since a bootstrap sample will on average fail to include about 37% of the original training cases (thereby substantially reducing the effective training set size) this OOB error rate is often adjusted in ways that are beyond the scope of this presentation to account for this difference in effective training set size.

$[a,b] \times [c,d]$ defined by

$$a = \min_{i=1,2,\ldots,N} x_{i1} \leq x_1 \leq \max_{i=1,2,\ldots,N} x_{i1} = b$$

and

$$c = \min_{i=1,2,\ldots,N} x_{i2} \leq x_2 \leq \max_{i=1,2,\ldots,N} x_{i2} = d$$

and looks for a way to split it at $x_1 = s_1$ or $x_2 = s_1$ so that a classifier constant on rectangles minimizes the training error. One then splits (optimally) one of the (now) two rectangles at $x_1 = s_2$ or $x_2 = s_2$, etc. Any series of binary splits of rectangles can be represented graphically as a binary tree, each split represented by a node where there is a fork and each final rectangle by an end node. It is often convenient to discuss rectangle-splitting (and where appropriate "unsplitting") in terms of operations on corresponding binary trees.

To describe a tree predictor more precisely, at a stage where l rectangles $R_1, R_2, \ldots, R_l$ in $\Re^p$ have been created, for $k = 0, 1$ let

$$\widehat{p_{mk}} = \frac{1}{\#\text{ training input vectors in } R_m} \sum_{\substack{i \text{ with } \boldsymbol{x}_i \\ \text{in } R_m}} I\left[y_i = k\right]$$

(for $m = 1, 2, \ldots, l$) be the fraction of training vectors with $\boldsymbol{x}_i$ in R_m and $y_i = k$. Then with

$$m\left(\boldsymbol{x}\right) = \text{the index of the rectangle to which } \boldsymbol{x} \text{ belongs}$$

the tree classifier based on the current set of rectangles is

$$\hat{y}_l^{\text{tree}}\left(\boldsymbol{x}\right) = \arg\max_{k \in \{0,1\}} \widehat{p_{m(\boldsymbol{x})k}}$$

the class that is most heavily represented in the rectangle to which $\boldsymbol{x}$ belongs. Then the training error rate (mis-classification rate) for this predictor is[6]

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I\left[y_i \neq \hat{y}_l^{\text{tree}}\left(\boldsymbol{x}\right)\right] = \frac{1}{N} \sum_{m=1}^{l} N_m \left(1 - \widehat{p_{mk(m)}}\right)$$

where $N_m = \#$ training input vectors in R_m, and $k\left(m\right) = \arg\max_{k \in \{0,1\}} \widehat{p_{mk}}$.

[6] Two related alternative forms for a training error are "the Gini index"

$$\overline{\text{err}} = \frac{1}{N} \sum_{m=1}^{l} N_m \left(\sum_{k=0}^{1} \widehat{p_{mk}} \left(1 - \widehat{p_{mk}}\right) \right)$$

and the so-called "cross entropy"

$$\overline{\text{err}} = -\frac{1}{N} \sum_{m=1}^{l} N_m \left(\sum_{k=0}^{1} \widehat{p_{mk}} \ln\left(\widehat{p_{mk}}\right) \right)$$

If one is to continue splitting beyond l rectangles, one looks for a value s_l to split one of the existing rectangles $R_1, R_2, \ldots, R_l$ on x_1 or x_2 or $\ldots$ or x_p and thereby produce the greatest reduction in $\overline{\text{err}}$.[7] (We note that there is no guarantee that after l splits one will have the best (in terms of $\overline{\text{err}}$) possible set of $l+1$ rectangles.)

It is, of course, possible to continue splitting rectangles/adding branches to a tree until every distinct $\boldsymbol{x}$ in the training set has its own rectangle. But that is not helpful in a practical sense, in that it corresponds to a very complex predictor. So how does one find a tree of appropriate size? One possibility is to use the number of rectangles constructed in the forward selection/greedy splitting algorithm as a complexity parameter and choose it via cross-validation.

Another (generally believed to be more effective) method involves first growing a tree to a full set of N nodes and then considering all possible prunings of it (this is a set of trees typically larger than the set met in the steps made to get to the full tree), identifying for each $\alpha \geq 0$ a tree minimizing

$$C(\alpha) = \#\text{ of rectangles} + \alpha \cdot \overline{\text{err}}$$

(this turns out to be a computationally feasible search). One then can choose an optimal α (for use on the whole training set) by cross-validation. A standard suggestion in this enterprise is to use the Gini index or cross entropy for tree growing and training mis-classification rate for tree pruning according to cost-complexity.

Beyond applying the classification tree notions more or less on their own, they form an important building block for so-called "**random forests**." These are versions of the "bagging" (bootstrap aggregation) idea of Section 4.2 applied specifically to classification trees. Suppose that one makes B bootstrap samples of N from the training set $\mathcal{T}$, say $\mathcal{T}_1^*, \mathcal{T}_2^*, \ldots, \mathcal{T}_B^*$. For each one of these samples, $\mathcal{T}_b^*$, one develops a corresponding classification tree by

1. at each node, randomly selecting m of the p input variables and finding an optimal single split of the corresponding rectangle over the selected input variables, splitting the rectangle, and
2. repeating 1 at each node until a small maximum number of training cases, $n_{\max}$, is reached in each rectangle.

(Note that no pruning is applied in this development.) Then let $\hat{y}^{*b}(\boldsymbol{x})$ be the corresponding tree-based predictor. The random forest predictor is then

$$\hat{y}_B^{\text{rf}}(\boldsymbol{x}) = \underset{k \in \{0,1\}}{\arg\max} \sum_{b=1}^{B} I\left[\hat{y}^{*b}(\boldsymbol{x}) = k\right]$$

The basic tuning/complexity parameters in the development of $\hat{y}_B^{\text{rf}}(\boldsymbol{x})$ are then m and $n_{\max}$. Commonly recommended default values are $m = \lfloor \sqrt{p} \rfloor$

[7] It is also possible to "prune" a tree by dropping off all splits below some node (thereby removing splits inside some rectangle), and typically one seeks to do this without much increasing $\overline{\text{err}}$.

and $n_{\max} = 1$. (The $n_{\max} = 1$ default is equivalent to splitting rectangles in the input space $\Re^p$ until all rectangles are completely homogeneous in terms of output values, y.) Values of these parameters might be chosen by cross-validation.

It is common practice to keep track of the OOB error rate as one builds a random forest predictor. That is, suppose that for each b one keeps track of the set of (OOB) indices $I(b) \subset \{1, 2, \ldots, N\}$ for which the corresponding training vector does not get included in the bootstrap training set $\mathcal{T}_b^*$, and lets

$$\hat{y}_{iB}^* = \underset{k \in \{0,1\}}{\arg\max} \sum_{b \leq B \text{ such that } i \in I(b)} I\left[\hat{y}^{*b}(\boldsymbol{x}_i) = k\right]$$

Then one keeps track of

$$OOB_B = \frac{1}{N} \sum_{i=1}^{N} I\left[y_i \neq \hat{y}_{iB}^*\right]$$

as a function of B. One expects convergence of OOB_B and plotting it versus B is a standard way of trying to assess whether enough bootstrap trees have been made to adequately represent the limiting predictor.

4.3.3 Rules Based on Logistic Regression

A different way to approximate an optimal classifier is to directly approximate $P[y = 0|\boldsymbol{x}]$ with, say, $\hat{p}_0(\boldsymbol{x})$ and then set

$$\hat{y}(\boldsymbol{x}) = I\left[\hat{p}_0(\boldsymbol{x}) < .5\right] \tag{22}$$

The most common/popular statistical methodology for doing the approximation/modeling job is so-called "logistic regression" methodology. It is based on the mathematically convenient assumption about the form of $P[y = 0|\boldsymbol{x}]$ that the so-called log odds are linear in $\boldsymbol{x}$, namely that

$$\ln\left(\frac{P[y = 0|\boldsymbol{x}]}{1 - P[y = 0|\boldsymbol{x}]}\right) = \beta_0 + \boldsymbol{x}'\boldsymbol{\beta} \tag{23}$$

where a constant β_0 and a p-vector $\boldsymbol{\beta}$ are to be specified. These are treated as unknown parameters of a statistical model to be estimated from the training data.

The linear log odds ratio assumption (23) produces the form

$$P[y = 0|\boldsymbol{x}] = \frac{\exp(\beta_0 + \boldsymbol{x}'\boldsymbol{\beta})}{1 + \exp(\beta_0 + \boldsymbol{x}'\boldsymbol{\beta})}$$

Displaying the dependence upon the parameters, write

$$p_0(\boldsymbol{x}|\beta_0, \boldsymbol{\beta}) = \frac{\exp(\beta_0 + \boldsymbol{x}'\boldsymbol{\beta})}{1 + \exp(\beta_0 + \boldsymbol{x}'\boldsymbol{\beta})} \quad \text{and} \quad p_1(\boldsymbol{x}|\beta_0, \boldsymbol{\beta}) = \frac{1}{1 + \exp(\beta_0 + \boldsymbol{x}'\boldsymbol{\beta})}$$

Then standard (maximum likelihood) logistic regression methodology maximizes

$$\prod_{i=1}^{N} p_{y_i}\left(\boldsymbol{x}_i|\beta_0,\boldsymbol{\beta}\right) \tag{24}$$

over choices of $\beta_0, \boldsymbol{\beta}$. This is a likelihood *conditional* on the $\boldsymbol{x}_i$ observed.

As it turns out, using not the 0-1 but rather -1-1 coding for y produces a particularly simple form for the log-likelihood. That is, with

$$w = \begin{cases} -1 & \text{if } y = 0 \\ 1 & \text{if } y = 1 \end{cases} \tag{25}$$

some algebra shows that the log-likelihood (to be optimized in ML fitting) is

$$-\sum_{i=1}^{N} \ln\left(1 + \exp\left(w_i\left(\beta_0 + \boldsymbol{x}_i'\boldsymbol{\beta}\right)\right)\right)$$

A general alternative to maximum likelihood useful in avoiding over-fitting for large N is minimization of a criterion like

$$-\ln\left(\prod_{i=1}^{N} p_{y_i}\left(\boldsymbol{x}_i|\beta_0,\boldsymbol{\beta}\right)\right) + \text{penalty}\left(\beta_0,\boldsymbol{\beta}\right)$$

For example, one popular version of this (called the "lasso") is (for a $\lambda > 0$) minimization of

$$\sum_{i=1}^{N} \ln\left(1 + \exp\left(w_i\left(\beta_0 + \boldsymbol{x}_i'\boldsymbol{\beta}\right)\right)\right) + \lambda \sum_{j=1}^{p} |\beta_j|$$

An alternative (that may be computationally easier to implement) is (again for a $\lambda > 0$) minimization of

$$\sum_{i=1}^{N} \ln\left(1 + \exp\left(w_i\left(\beta_0 + \boldsymbol{x}_i'\boldsymbol{\beta}\right)\right)\right) + \lambda \sum_{j=1}^{p} \beta_j^2$$

Because the results of these minimizations are scale (of the coordinates of $\boldsymbol{x}$) dependent, it is common to standardize the inputs before optimization. (See Section 5.2.1 for more details regarding standardization of predictor variables.)

It is common in modern data analytics to encounter situations where $g\left(0\right) = P\left[y = 0\right]$ is quite small. Rather than trying to do analysis on a random sample of $(y, \boldsymbol{x})$ pairs where there would be very few $y = 0$ cases, there are a number of potentially important practical reasons for doing analysis of a data set consisting of random sample of N_0 instances ("cases") with $y = 0$ and a random sample of N_1 instances ("controls") with $y = 1$, *where* $N_0/\left(N_0 + N_1\right)$ *is nowhere near*

as small as $P[y=0]$. (In fact, N_1 on the order of 5 or 6 times N_0 is often recommended.)

Then

$$\ln\left(\frac{P[y=0|\boldsymbol{x}]}{P[y=1|\boldsymbol{x}]}\right)=\ln\left(\frac{g(0)\,f(\boldsymbol{x}|0)}{g(1)\,f(\boldsymbol{x}|1)}\right)=\ln\left(\frac{g(0)}{g(1)}\right)+\ln\left(\frac{f(\boldsymbol{x}|0)}{f(\boldsymbol{x}|1)}\right)$$

So under the logistic regression assumption that

$$\ln\left(\frac{P[y=0|\boldsymbol{x}]}{P[y=1|\boldsymbol{x}]}\right)=\beta_0+\boldsymbol{x}'\boldsymbol{\beta}$$

fitting to a case-control data set should produce

$$\begin{aligned}\hat{\beta}_0^{\text{cc}}+\boldsymbol{x}'\hat{\boldsymbol{\beta}}^{\text{cc}} &\approx \ln\left(\frac{N_0}{N_1}\right)+\ln\left(\frac{f(\boldsymbol{x}|0)}{f(\boldsymbol{x}|1)}\right)\\ &=\ln\left(\frac{P[y=0|\boldsymbol{x}]}{P[y=1|\boldsymbol{x}]}\right)+\ln\left(\frac{N_0}{N_1}\right)-\ln\left(\frac{g(0)}{g(1)}\right)\end{aligned}$$

So (presuming that an estimate $\hat{g}(0)$ of $g(0)$ is available) estimated coefficients

$$\hat{\beta}_0\equiv\hat{\beta}_0^{\text{cc}}-\ln\left(\frac{N_0}{N_1}\right)+\ln\left(\frac{\hat{g}(0)}{1-\hat{g}(0)}\right)\quad\text{and}\quad\hat{\boldsymbol{\beta}}=\hat{\boldsymbol{\beta}}^{\text{cc}}$$

are appropriate for the original context.

In any event, however they are derived, estimates $\hat{\beta}_0$ and $\hat{\boldsymbol{\beta}}$ lead to an estimate

$$\hat{p}_0(\boldsymbol{x})=p_0\left(\boldsymbol{x}|\hat{\beta}_0,\hat{\boldsymbol{\beta}}\right)$$

and in turn a practical classifier (22). It is worth noting that by virtue of the linear form of $\beta_0+\boldsymbol{x}'\boldsymbol{\beta}$ in the input variable $\boldsymbol{x}$, the decision boundary between where $\hat{y}(\boldsymbol{x})=0$ and $\hat{y}(\boldsymbol{x})=1$ is a $(p-1)$-dimensional hyperplane in $\Re^p$. (For $p=1$ the boundary is a point. For $p=2$ the boundary is a line. For $p=3$ the boundary is a plane. Etc.) This is not so specialized as it might first appear to be once one notes that there is nothing to prevent using functions of some coordinates of $\boldsymbol{x}$ to make other coordinates. For example, with 2 basic predictors x_1 and x_2, one can make a $p=5$-dimensional input vector by setting $x_3=x_1^2, x_4=x_2^2$ and $x_5=x_1x_2$ and have quadratic surfaces in $\Re^2$ for decision boundaries. This possibility of building new predictors from existing ones opens a wide vista of potentially increasingly complex classifiers built on the logistic regression (and, for that matter, tree-based) idea(s). Choosing between them then becomes a matter of careful use of cross-validation.

4.3.4 Support Vector Classifiers

The observation that logistic regression leads to classification boundaries in $\Re^p$ that are $(p-1)$-dimensional hyperplanes suggests consideration of other ways of choosing such hyperplanes. So called "support vector classifiers" are one

popular kind of classifier with linear classification boundaries. For $\boldsymbol{\beta} \in \Re^p$ and $\beta_0 \in \Re$ we'll consider classifiers of the form

$$\hat{y}(\boldsymbol{x}) = I[\beta_0 + \boldsymbol{x}'\boldsymbol{\beta} > 0] \tag{26}$$

We will approach the problem of choosing $\boldsymbol{\beta}$ and β_0 by attempting to in some sense provide a maximal cushion around a hyperplane separating between $\boldsymbol{x}_i$ with corresponding $y_i = 0$ and $\boldsymbol{x}_i$ with corresponding $y_i = 1$.[8] We begin with (rare) cases where some hyperplane provides perfect separation.

In the case that there is a classifier of form (26) with 0 training error rate, for w_i corresponding to y_i via relationship (25) we consider the optimization problem

$$\begin{array}{l} \text{maximize} \\ \boldsymbol{u} \text{ with } \|\boldsymbol{u}\| = 1 \\ \text{and } \beta_0 \in \Re \end{array} \quad M \quad \text{subject to } w_i(\boldsymbol{x}_i'\boldsymbol{u} + \beta_0) \geq M \quad \forall i \tag{27}$$

This can be thought of in terms of choosing a unit vector $\boldsymbol{u}$ (or direction) in $\Re^p$ so that upon projecting the training input vectors $\boldsymbol{x}_i$ onto the subspace of multiples of $\boldsymbol{u}$ there is maximum separation between convex hull of projections of the $\boldsymbol{x}_i$ with $y_i = 0$ and the convex hull of projections of $\boldsymbol{x}_i$ with corresponding $y_i = 1$. (The sign on $\boldsymbol{u}$ is chosen to give the latter larger $\boldsymbol{x}_i'\boldsymbol{u}$ than the former.) M is called the "margin" and the optimization problem seeks a "maximum margin" classifier. As it turns out, (27) can be rewritten as

$$\begin{array}{l} \text{minimize} \\ \boldsymbol{\beta} \in \Re^p \\ \text{and } \beta_0 \in \Re \end{array} \quad \frac{1}{2}\|\boldsymbol{\beta}\|^2 \quad \text{subject to } w_i(\boldsymbol{x}_i'\boldsymbol{\beta} + \beta_0) \geq 1 \quad \forall i \tag{28}$$

This formulation (28) is that of a convex (quadratic criterion, linear inequality constraints) optimization problem for which there exists standard theory, algorithms, and software.

The (typically relatively few) points $\boldsymbol{x}_i$ for which $w_i(\boldsymbol{x}_i'\boldsymbol{u} + \beta_0) = M$ are cases where $\boldsymbol{x}_i$ is on the surface of the "padded hyperplane" identified in (27). They are called "support vectors" and it turns out the form of the hyperplane (and the margin) depends only upon them. In some sense that makes most of the training set irrelevant in choosing a support vector classifier, in turn making it fairly "complex" and highly sensitive to the exact values of the support vectors.

In a linearly non-separable case, the convex optimization problem (28) does not have a solution (no pair $\boldsymbol{\beta} \in \Re^p$ and $\beta_0 \in \Re$ provides $w_i(\boldsymbol{x}_i'\boldsymbol{\beta} + \beta_0) \geq 1 \ \forall i$). We might therefore (in looking for good choices of $\boldsymbol{\beta} \in \Re^p$ and $\beta_0 \in \Re$) try to relax the constraints of the problem slightly. That is, suppose that $\xi_i \geq 0$ and consider the set of constraints

$$w_i(\boldsymbol{x}_i'\boldsymbol{\beta} + \beta_0) + \xi_i \geq 1 \ \forall i$$

[8] As for classifiers built on penalized logistic regression, it is fairly standard practice to standardize the coordinates of $\boldsymbol{x}$. See again Section 5.2.1 for details of standardization.

(the ξ_i are called "slack" variables and provide some "wiggle room" in search for a hyperplane that "nearly" separates the two classes with a good margin). We might try to control the total amount of slack allowed by setting a limit

$$\sum_{i=1}^{N} \xi_i \leq C$$

for some positive C. Note that if $w_i\left(\boldsymbol{x}_i'\boldsymbol{\beta}+\beta_0\right) \geq 0$ case i is correctly classified in the training set, and so if for some pair $\boldsymbol{\beta} \in \Re^p$ and $\beta_0 \in \Re$ this holds for all i, we have a separable problem. So any non-separable problem must have at least one negative $w_i\left(\boldsymbol{x}_i'\boldsymbol{\beta}+\beta_0\right)$ for any $\boldsymbol{\beta} \in \Re^p$ and $\beta_0 \in \Re$ pair. This in turn requires that the budget C must be at least 1 for a non-separable problem to have a solution even with the addition of slack variables. In fact, this reasoning implies that a budget C allows for at most C mis-classifications in the training set. And in a non-separable case, C must be allowed to be large enough so that some choice of $\boldsymbol{\beta} \in \Re^p$ and $\beta_0 \in \Re$ produces a classifier with training error rate no larger than C/N.

In any event, we consider the optimization problem

$$\begin{array}{c}\text{minimize} \\ \boldsymbol{\beta} \in \Re^p \\ \text{and } \beta_0 \in \Re\end{array} \frac{1}{2}\|\boldsymbol{\beta}\|^2 \quad \text{subject to} \left\{\begin{array}{c} w_i\left(\boldsymbol{x}_i'\boldsymbol{\beta}+\beta_0\right)+\xi_i \geq 1 \quad \forall i \\ \text{for some } \xi_i \geq 0 \text{ with } \sum_{i=1}^{N} \xi_i \leq C\end{array}\right. \tag{29}$$

that can be thought of as generalizing the problem (28). Optimization problem (29) is equivalent to

$$\begin{array}{c}\text{maximize} \\ \boldsymbol{u} \text{ with } \|\boldsymbol{u}\|=1 \\ \text{and } \beta_0 \in \Re\end{array} M \quad \text{subject to} \left\{\begin{array}{c} w_i\left(\boldsymbol{x}_i'\boldsymbol{u}+\beta_0\right) \geq M\left(1-\xi_i\right) \quad \forall i \\ \text{for some } \xi_i \geq 0 \text{ with } \sum_{i=1}^{N} \xi_i \leq C\end{array}\right.$$

generalizing the original problem (27). A more convenient version of (29) is

$$\begin{array}{c}\text{minimize} \\ \boldsymbol{\beta} \in \Re^p \\ \text{and } \beta_0 \in \Re\end{array} \frac{1}{2}\|\boldsymbol{\beta}\|^2+C^* \sum_{i=1}^{N} \xi_i \quad \text{subject to} \left\{\begin{array}{c} w_i\left(\boldsymbol{x}_i'\boldsymbol{\beta}+\beta_0\right)+\xi_i \geq 1 \quad \forall i \\ \text{for some } \xi_i \geq 0\end{array}\right. \tag{30}$$

Problem (30) is again one for which appropriate theory, algorithms, and software exist. The constant C^* functions as a regularization/complexity parameter and large C^* in (30) correspond to small C in (29). Large C^*/small C correspond to high classifier complexity. One might then choose C^* or C by cross-validation. In fact, even where a perfect separation between $y=0$ and $y=1$ cases of $\boldsymbol{x}$ is possible, there is good reason to consider this formulation of the problem with slack variables and allow "padded hyperplanes" with some $\boldsymbol{x}_i$ "violating their margins" (slightly). In this case, all points on or violating their margins become "support vectors."

To be more explicit, the classifier is (again) of form (26)

$$\hat{y}(\boldsymbol{x}) = I[\beta_0 + \boldsymbol{x}'\boldsymbol{\beta} > 0]$$

for the optimizing $\boldsymbol{\beta}$ and β_0. As it turns out, the margin is (for the optimzing $\boldsymbol{\beta}$)

$$M = \frac{1}{\|\boldsymbol{\beta}\|}$$

Any training case with $y_i = 0$ but $\beta_0 + \boldsymbol{x}_i'\boldsymbol{\beta} > -M$ or with $y_i = 1$ but $\beta_0 + \boldsymbol{x}_i'\boldsymbol{\beta} < M$ are support vectors and figure into the exact form of the classifier.

There are two ways in which these support vector classification ideas are typically extended beyond the realm of producing linear classification boundaries. The first is the device mentioned when discussing the use of logistic regression in classification, namely employing functions of some basic coordinates of $\boldsymbol{x}$ to make other coordinates. A second is the use of values of "kernels" to make "support vector machines." This a topic beyond the scope of the present discussion, but deserves mention so that the reader knows that what is here is not the "whole story" of the popularity and success of support vector classifiers.

4.4 "Ensembles" of Classifiers

In Section 4.2 we noted that bagging (using majority voting for B versions of a basic classifier built on different bootstrap samples from the training set) is a way of combining multiple classifiers to make another one (thereby broadening the set of available methods). From the point of view that practical classification is always a matter of trying to find good training-data-based approximations to $\hat{y}^{\text{opt}}(\boldsymbol{x})$ and it is unlikely in a real problem that this is simple enough that a single methodology employed will even have the ability to "get really close" to $\hat{y}^{\text{opt}}(\boldsymbol{x})$, it should not be surprising to learn that often it is effective to combine several different individually well-chosen classifiers. Good practical classification strategy often employs "ensembles" of classifiers. That is, one might have reason to fit and try to make simultaneous use of all of a nearest neighbor classifier, a random forest classifier, several classifiers built on different forms of logistic regression, and several forms of support vector machines.

There are at least two different ways this might be approached. In the first place, some kinds of classification methods like nearest neighbor rules, decision trees, random forests and logistic regression-based rules essentially provide in their development approximations to $P[y = 0|\boldsymbol{x}]$. (This can be in terms of the fraction of k neighbors with $y = 0$, the fraction of 0's in a final rectangle, the fraction of trees in a forest voting "$y = 0$" at $\boldsymbol{x}$, the estimated logistic regression probability of 0 at $\boldsymbol{x}$, etc.) Making a weighted average of these estimates and classifying on the basis of this is one way of combining the methods. Of course, the "rub" is knowing what weights to use, and studying that issue is potentially incredibly compute-intensive. Cross-validation based searching over the space of possible weight vectors is in theory possible. Another less onerous (but also less satisfying) possibility is to simply hold out a fixed test set from fitting after

deciding on the component classification methods and then evaluate "ensemble" performance on the test set.

A second possibility for combining classifiers is to make a weighted average of the classifier 0-1 outputs and decide in favor of $y = 1$ when this average exceeds .5. Again, the practical issue to be faced is finding appropriate weights, and the comments immediately above continue to apply.

5 SEL Prediction Problems

Classification is one of the two most common statistical (machine) learning problems. Squared error loss prediction is the other. In this last main section of this outline, we consider the SEL problem from a practical point of view. This is a problem where the obvious training error for the predictor $\hat{y}(\boldsymbol{x})$ is built on

$$SSE = \sum_{i=1}^{N} \left(y_i - \hat{y}(\boldsymbol{x}_i)\right)^2 \tag{31}$$

i.e. is

$$\overline{\text{err}} = \frac{1}{N} SSE$$

5.1 Normal Linear Models and OLS

Already in Section 1.1, we saw that beginning from a joint distribution for $(y, \boldsymbol{x})$ the SEL optimal predictor is

$$\hat{y}^{\text{opt}}(\boldsymbol{x}) = \text{E}[y|\boldsymbol{x}]$$

A place to begin developing practical predictors is to assume that $\text{E}[y|\boldsymbol{x}]$ has a simple form. A particularly tractable one is

$$\text{E}[y|\boldsymbol{x}] = \boldsymbol{x}'\boldsymbol{\beta} = \sum_{j=1}^{p} \beta_j x_j$$

for some $\boldsymbol{\beta} \in \Re^p$. If one then treats $\boldsymbol{\beta}$ as an unknown parameter the question is how to fit this based on the training data. Once a fitting method has been used to produce $\hat{\boldsymbol{\beta}}$ the obvious data-based predictor of y is

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{x}'\hat{\boldsymbol{\beta}} = \sum_{j=1}^{p} \hat{\beta}_j x_j \tag{32}$$

One way to motivate a fitting method is through the use of the so-called normal linear model. For known $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ and iid $\text{N}(0, \sigma^2)$ random variables $\epsilon_1, \epsilon_2, \ldots, \epsilon_N$, the normal linear model says that

$$y_i = \boldsymbol{x}_i'\boldsymbol{\beta} + \epsilon_i$$

for $i = 1, 2, \ldots, N$. It is convenient to define vectors and matrices

$$\boldsymbol{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1' \\ \boldsymbol{x}_2' \\ \vdots \\ \boldsymbol{x}_N' \end{pmatrix}, \text{ and } \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{pmatrix}$$

and write the linear model in matrix form as

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

By appropriate choices of $\boldsymbol{X}$, a variety of standard statistical models can be seen to be of this form. For example, the choice

$$\boldsymbol{x}' = (1, z_1, z_2, \ldots, z_{p-1})$$

gives the so-called "multiple linear regression model" with $p-1$ predictors $z_1, z_2, \ldots, z_{p-1}$.

The (conditional on $\boldsymbol{X}$) log-likelihood for the normal linear model based on the training data is

$$\mathcal{L}\left(\boldsymbol{\beta}, \sigma^2\right) = -\frac{N}{2}\ln\left(2\pi\right) - \frac{N}{2}\ln\left(\sigma^2\right) - \frac{1}{2\sigma^2}\left(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\right)'\left(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\right)$$

Maximum likelihood estimates of the parameters of this statistical model are $\hat{\boldsymbol{\beta}}^{\text{MLE}}$ the minimizer of $\left(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\right)'\left(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\right)$ and with

$$\begin{aligned} SSE &= \left(\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}^{\text{MLE}}\right)'\left(\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}^{\text{MLE}}\right) \\ &= \sum_{i=1}^{N}\left(y_i - \boldsymbol{x}_i'\hat{\boldsymbol{\beta}}^{\text{MLE}}\right)^2 \end{aligned}$$

the estimate

$$\widehat{\sigma^2}^{\text{MLE}} = \frac{1}{N}SSE$$

The only question then is what form the minimizer of $\left(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\right)'\left(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\right)$ takes. Provided that ($N > p$ and) $\text{rank}(\boldsymbol{X}) = p$, $\boldsymbol{X}'\boldsymbol{X}$ **is non-singular**. Then some vector calculus shows that the **ordinary least squares** estimator of $\boldsymbol{\beta}$ is

$$\hat{\boldsymbol{\beta}}^{\text{OLS}} = \hat{\boldsymbol{\beta}}^{\text{MLE}} = \left(\boldsymbol{X}'\boldsymbol{X}\right)^{-1}\boldsymbol{X}'\boldsymbol{Y}$$

(We will continue in this exposition under the assumption that indeed $\boldsymbol{X}'\boldsymbol{X}$ is non-singular.) Of course, the version of predictor (32) corresponding to this fitting method is the "OLS predictor"

$$\hat{y}^{\text{OLS}}\left(\boldsymbol{x}\right) = \boldsymbol{x}'\hat{\boldsymbol{\beta}}^{\text{OLS}} = \sum_{j=1}^{p}\hat{\beta}_j^{\text{OLS}}x_j \tag{33}$$

The normal linear model has a rich theory and history as the basis of most elementary methods of statistical inference for continuous observations. While much *could* be said about it (and *is* said about it in courses specifically on "the linear model") we will limit our treatment here with a view to having time to discuss methods of SEL prediction beyond OLS. In particular, the following is a summary of the most basic results for normal linear models, and all that we choose to provide in this outline.

Theorem 34 *Under the normal linear model* $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ *(assuming that* $N > p$ *and rank*$(\boldsymbol{X}) = p$*)*

1. $\hat{\boldsymbol{\beta}}^{OLS}$ *and* SSE *are independent,*
2. $\frac{SSE}{\sigma^2} \sim \chi^2_{N-p}$*,*
3. *for* d_j *the* j*th diagonal element of* $\left(\boldsymbol{X}'\boldsymbol{X}\right)^{-1}$

$$\hat{\beta}_j^{OLS} \sim N\left(\beta_j, d_j\sigma^2\right) \quad \text{, and}$$

4. *for* $\boldsymbol{c} \in \Re^p$

$$\boldsymbol{c}'\hat{\boldsymbol{\beta}}^{OLS} \sim N\left(\boldsymbol{c}'\boldsymbol{\beta}, \boldsymbol{c}'\left(\boldsymbol{X}'\boldsymbol{X}\right)^{-1}\boldsymbol{c}\sigma^2\right)$$

The results in Theorem 34 are sufficient to produce confidence limits for σ^2 and the entries of $\boldsymbol{\beta}$, confidence limits for linear combinations of the entries of $\boldsymbol{\beta}$ (including ones that are mean values of y at inputs $\boldsymbol{x}$ of interest), and various kinds of normal theory "prediction limits."

5.2 Other Practical Predictors Based on "Training Data"

We now turn to consideration of other kinds of SEL predictors that are often superior to OLS linear predictors.

5.2.1 Other (Non-OLS) Linear Predictors

In modern predictive analytics problems (that feature large p and N), it is not unusual to find that ordinary least squares prediction based on a p-dimensional input produces overfit. We consider some alternatives to OLS-produced linear predictors that can be tuned via cross-validation to avoid overfit and improve prediction on a test set.

Best Subset Regression Where p is at all large (and sometimes even when it is not), there are good reasons (including reducing the tendency to overfit) to look for a subset of k of the p predictors $x_1, x_2, \ldots, x_p$, say ones with indices $j_1 < j_2 < \cdots < j_k$, to use in preference to all p inputs x_j when predicting y. Of course, as p grows there are a huge number of possibilities to consider. For example, with $p = 100$ predictors there are about 1.3×10^{30} (exactly $2^{100} -$

1) possible subsets to consider. But for small-to-moderate values of p there are clever computational algorithms that efficiently find the best (in terms of producing the smallest available value of SSE) subset of predictors of size k for each k from 1 to p. And when p is too large for this to be workable, forward selection (that begins with the predictor with the largest correlation with y and then sequentially adds the single predictor from those not in a current subset providing the largest decrease in SSE to the current subset) often provides at least a sensible sequence of nested sets of predictors of sizes $k = 1, 2, \ldots, p$. One can then treat k as a complexity parameter (large k corresponding to large predictor complexity) and choose it and its corresponding subset of predictors by cross validation. (One should probably go through the exercise of choosing the size-k subsets differently for each of the K folds.)

Notice that one way to think about subset regression is in terms of doing least squares fitting after "zeroing out" some of the elements of $\boldsymbol{\beta}$. That is, it amounts to least squares applied subject to constraints that some elements of the parameter vector are 0. From this point of view, subset regression is roughly accomplished by forcing some "shrinking" of the parameter vector toward $\mathbf{0}$ in selected dimensions. Reduced predictor complexity is then roughly associated with parameter vectors shrunken toward the origin. We next consider a different technology for doing this.

Lasso, Ridge, and Elastic Net (Penalized) Regression The next methods to be discussed produce different predictors depending upon how various inputs are scaled. So that we are talking about some specific version of them, we will in this discussion of alternative (to OLS) linear predictors assume that the response y has been centered and each x_j has been standardized. That is, if one begins with responses w_i with corresponding inputs/predictors z_{ij} for $j = 1, 2, \ldots, p$ we consider

$$\boldsymbol{Y} = \begin{pmatrix} w_1 - \bar{w} \\ w_2 - \bar{w} \\ \\ w_N - \bar{w} \end{pmatrix}$$

and for $\bar{z}_j$ and s_j respectively the sample mean and sample standard deviation of $z_{1j}, z_{2j}, \ldots, z_{Nj}$,

$$\boldsymbol{X} = \begin{pmatrix} \frac{z_{11} - \bar{z}_1}{s_1} & \frac{z_{12} - \bar{z}_2}{s_2} & \cdots & \frac{z_{1p} - \bar{z}_p}{s_p} \\ \frac{z_{21} - \bar{z}_1}{s_1} & \frac{z_{22} - \bar{z}_2}{s_2} & \cdots & \frac{z_{2p} - \bar{z}_p}{s_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{z_{N1} - \bar{z}_1}{s_1} & \frac{z_{N2} - \bar{z}_2}{s_2} & \cdots & \frac{z_{Np} - \bar{z}_p}{s_p} \end{pmatrix}$$

Then if one creates a linear predictor $\hat{y}(\boldsymbol{x}) = \boldsymbol{x}'\hat{\boldsymbol{\beta}}$ based on these centered and standardized variables, a corresponding predictor for the response on the original scale in terms of the original predictors is

$$\hat{w}(\boldsymbol{z}) = \bar{w} - \bar{\boldsymbol{z}}'\hat{\boldsymbol{\gamma}} + \boldsymbol{z}'\hat{\boldsymbol{\gamma}}$$

for $\bar{\boldsymbol{z}}$ the p-vector of sample averages of the p original predictors and

$$\hat{\boldsymbol{\gamma}} = \begin{pmatrix} \frac{\hat{\beta}_1}{s_1} \\ \frac{\hat{\beta}_2}{s_2} \\ \vdots \\ \frac{\hat{\beta}_p}{s_p} \end{pmatrix}$$

Then, we begin by reiterating that $\hat{\boldsymbol{\beta}}^{\text{OLS}}$ is a solution to the problem of minimizing

$$SSE = (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta})'(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}) = \sum_{i=1}^{N} (y_i - \boldsymbol{x}_i'\boldsymbol{\beta})^2$$

over choices of $\boldsymbol{\beta} \in \Re^p$. To find an alternative to $\hat{\boldsymbol{\beta}}^{\text{OLS}}$ that is "shrunken towards $\mathbf{0}$" we pose a related "penalized" optimization problem, namely minimization of (over choices of $\boldsymbol{\beta} \in \Re^p$)

$$\frac{1}{N}\sum_{i=1}^{N} (y_i - \boldsymbol{x}_i'\boldsymbol{\beta})^2 + \lambda \left(\alpha \left(2\sum_{j=1}^{p} |\beta_j| \right) + (1-\alpha)\sum_{j=1}^{p} \beta_j^2 \right) \tag{34}$$

for an $\alpha \in [0,1]$ and $\lambda \geq 0$. (This is the `GLMNET` version of the penalized sum of squares. Other similar/ultimately equivalent versions are possible.)

In the optimization criterion (34) the value of α is typically thought of as a fixed choice specifying the fitting method, the two extreme values $\alpha = 0$ and $\alpha = 1$ producing so-called **lasso** (least absolute shrinkage and selection operator) and **ridge** regression criteria. Any $0 < \alpha < 1$ version of criterion (34) is a so-called **elastic net** criterion. λ controls the influence of the "penalty" term on the choice of $\boldsymbol{\beta}$. The extreme case of $\lambda = 0$ reduces the criterion (34) to the OLS criterion. As $\lambda \to \infty$ it is completely plausible (and correct) that an optimizer of criterion (34) is forced to $\mathbf{0}$. λ then functions as a complexity parameter and is typically chosen (for a given choice of α) via cross-validation.

We'll call an optimizer of criterion (34) for an $0 < \alpha < 1$ by the name

$$\hat{\boldsymbol{\beta}}_{\lambda}^{\text{ElasticNet}}$$

and in the special cases of $\alpha = 0$ and $\alpha = 1$ respectively employ the notations

$$\hat{\boldsymbol{\beta}}_\lambda^{\text{Ridge}} \quad \text{and} \quad \hat{\boldsymbol{\beta}}_\lambda^{\text{Lasso}}$$

As it turns out, there is a closed form expression for $\hat{\boldsymbol{\beta}}_\lambda^{\text{Ridge}}$ namely

$$\hat{\boldsymbol{\beta}}_\lambda^{\text{Ridge}} = \left(\boldsymbol{X}'\boldsymbol{X} + N\lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}'\boldsymbol{Y}$$

but for $\alpha > 0$ no such formula for $\hat{\boldsymbol{\beta}}_\lambda^{\text{ElasticNet}}$ (or $\hat{\boldsymbol{\beta}}_\lambda^{\text{Lasso}}$) is available. However, there are efficient computational algorithms for finding $\hat{\boldsymbol{\beta}}_\lambda^{\text{ElasticNet}}$ (including $\hat{\boldsymbol{\beta}}_\lambda^{\text{Lasso}}$) coded into packages like `GLMNET`.

It's an important (not here obvious) fact that typically many entries of $\hat{\boldsymbol{\beta}}_\lambda^{\text{Lasso}}$ will be exactly 0. That means that using a lasso version of criterion (34) to choose a coefficient vector for a linear predictor imposes an attractive kind of automatic implicit "variables selection"/"features selection" solution on the fitting problem.

5.2.2 Nearest Neighbor Predictors

A *linear form* assumption/restriction for a SEL predictor is strong one (even granting the possibility of building additional predictors from some original basic ones via transformations). It's thus worthwhile to think fundamentally about the (conditional mean) SEL optimal predictor and how it might be approximated in very flexible terms. That is, the basic

$$\hat{y}^{\text{opt}}(\boldsymbol{x}) = \text{E}[y|\boldsymbol{x}]$$

says that the optimal predictor is the average response y at the input vector $\boldsymbol{x}$. We can then ask how that quantity might be approximated based on a training set.

In most instances, a training set will have few if any pairs $(y_i, \boldsymbol{x}_i)$ with $\boldsymbol{x}_i$ *exactly* equal to an $\boldsymbol{x}$ at which one wishes to make a prediction for y. So the simple notion of just averaging training responses y_i with $\boldsymbol{x}_i = \boldsymbol{x}$ in order to approximate the optimal predictor won't work. But this line of thinking does suggest something that *might* be useful. That is, using the notation

$$\mathcal{N}_k(\boldsymbol{x}) = \{k \text{ points } \boldsymbol{x}_i \text{ from the training set closest to } \boldsymbol{x}\}$$

first introduced at display (19) in the context of nearest neighbor *classifiers*, one might define a k-nearest neighbor SEL predictor of a measurement scale response y as

$$\hat{y}^{k\text{nn}}(\boldsymbol{x}) = \frac{1}{k}\sum_{\substack{i \text{ s.t. } \boldsymbol{x}_i \in \\ \mathcal{N}_k(\boldsymbol{x})}} y_i \tag{35}$$

(One uses the sample mean of numerical responses corresponding to input vectors in the neighborhood as opposed to a majority vote of classifications corresponding to inputs in the neighborhood. Typically one uses standardized

versions of the coordinates of the input vector in defining "nearness" of neighbors.) As in the classification context, as k decreases, the flexibility/complexity of the k-nearest neighbor rule (35) increases (1-nearest neighbor predictors are far more flexible than N-nearest neighbor predictors that always return the sample mean of y in the training set). So k is a complexity parameter that can be chosen via cross-validation. When p is small and N is very large, a k-nearest neighbor predictor can be effective. But perhaps of more practical importance is the fact that it provides motivation for other predictors, like those based on trees and those based on "smoothing."

5.2.3 Tree-Based SEL Predictors

There are more or less obvious SEL versions of the tree-based classification predictors introduced in Section 4.3.2.

Regression Trees To describe a "regression" tree predictor precisely, at a stage where l rectangles $R_1, R_2, \ldots, R_l$ in $\Re^p$ have been created, for $k = 0, 1$ let

$$\bar{y}_m = \frac{1}{\#\text{ training input vectors in } R_m} \sum_{\substack{i \text{ with } \boldsymbol{x}_i \\ \text{in } R_m}} y_i$$

(for $m = 1, 2, \ldots, l$) be the sample mean of the responses with $\boldsymbol{x}_i$ in R_m. Then with

$$m(\boldsymbol{x}) = \text{the index of the rectangle to which } \boldsymbol{x} \text{ belongs}$$

the regression tree predictor based on the current set of rectangles is

$$\hat{y}_l^{\text{tree}}(\boldsymbol{x}) = \bar{y}_{m(\boldsymbol{x})}$$

the sample mean response in the rectangle to which $\boldsymbol{x}$ belongs. Then the training error rate (mean squared error) for this predictor is

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}_l^{\text{tree}}(\boldsymbol{x})\right)^2$$

If one is to continue splitting beyond l rectangles, one looks for a value s_l to split one of the existing rectangles $R_1, R_2, \ldots, R_l$ on x_1 or x_2 or $\ldots$ or x_p and thereby produce the greatest reduction in $\overline{\text{err}}$. (As for classification, we note that there is no guarantee that after l splits one will have the best (in terms of $\overline{\text{err}}$) possible set of $l + 1$ rectangles.)

A version of the tree-building methodology discussed in the classification context can, of course, be applied in the SEL prediction context. One first grows a tree until each final node has no more than (say) 5 training cases associated with it and then considers all possible prunings of it (this is a set of trees typically larger than the set met in the steps made to get to the full tree), identifying for each $\alpha \geq 0$ a sub-tree minimizing

$$C(\alpha) = \#\text{ of rectangles} + \alpha \cdot \overline{\text{err}}$$

(as for classification, this is a computationally feasible search). One can choose an optimal α (for subsequent use on the whole training set) by cross-validation.

Random Forests Beyond applying the regression tree notions on their own, they form important building blocks for regression versions of random forests. These are versions of the "bagging" (bootstrap aggregation) idea of Section 4.2 applied specifically to regression trees. Suppose that one makes B bootstrap samples of N from the training set $\mathcal{T}$, say $\mathcal{T}_1^*, \mathcal{T}_2^*, \ldots, \mathcal{T}_B^*$. For each one of these samples, $\mathcal{T}_b^*$, one develops a corresponding regression tree by

1. at each node, randomly selecting m of the p input variables and finding an optimal single split of the corresponding rectangle over the selected input variables, splitting the rectangle, and
2. repeating 1 at each node until a small maximum number of training cases, $n_{\max}$, is reached in each rectangle.

(Note that no pruning is applied in this development.) Then let $\hat{y}^{*b}(\boldsymbol{x})$ be the corresponding tree-based predictor. The random forest predictor is then

$$\hat{y}_B^{\mathrm{rf}}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} \hat{y}^{*b}(\boldsymbol{x})$$

The basic tuning/complexity parameters in the development of $\hat{y}_B^{\mathrm{rf}}(\boldsymbol{x})$ are m and $n_{\max}$. Commonly recommended default values are $m = \lfloor p/3 \rfloor$ and $n_{\max} = 5$. (Better values of these parameters might be chosen by cross-validation.)

As for classification, it is common practice to keep track of an OOB error rate as one builds a regression random forest predictor. That is, suppose that for each b one keeps track of the set of (OOB) indices $I(b) \subset \{1, 2, \ldots, N\}$ for which the corresponding training vector does not get included in the bootstrap training set $\mathcal{T}_b^*$, and lets

$$\hat{y}_{iB}^* = \frac{1}{\#\text{ of } b \leq B \text{ such that } i \in I(b)} \sum_{b \leq B \text{ such that } i \in I(b)} \hat{y}^{*b}(\boldsymbol{x}_i)$$

Then one keeps track of

$$OOB_B = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_{iB}^*)^2$$

as a function of B. One expects convergence of OOB_B and plotting it versus B is a standard way of trying to assess whether enough bootstrap trees have been made to adequately represent the limiting predictor.

5.2.4 Kernel Smoothing Methods

The central idea of kernel smoothing is that when finding $\hat{y}(\boldsymbol{x}_0)$ I might weight points in the training set according to how close they are to $\boldsymbol{x}_0$, do some kind of fitting around $\boldsymbol{x}_0$, and ultimately read off the value of the fit at $\boldsymbol{x}_0$.

One-dimensional Kernel Smoothers For the time being, suppose that x takes values in $[0,1]$. Invent weighting schemes for points in the training set by defining a (usually, symmetric about 0) non-negative, real-valued function $D(t)$ that is non-increasing for $t \geq 0$ and non-decreasing for $t \leq 0$. Often $D(t)$ is taken to have value 0 unless $|t| \leq 1$. Then, a kernel function is

$$K_\lambda(x, x_0) = D\left(\frac{x - x_0}{\lambda}\right) \tag{36}$$

where λ is a "bandwidth" parameter that controls the rate at which weights drop off as one moves away from x_0 (and indeed in the case that $D(t) = 0$ for $|t| > 1$, how far one moves away from x_0 before no weight is assigned). Common choices for D are

1. the Epanechnikov quadratic kernel, $D(t) = \frac{3}{4}\left(1 - t^2\right) I[|t| \leq 1]$,
2. the "tri-cube" kernel, $D(t) = \left(1 - |t|^3\right)^3 I[|t| \leq 1]$, and
3. the standard normal density, $D(t) = \phi(t)$.

Using weights (36) to make a weighted average of training responses, one arrives at the Nadaraya-Watson kernel-weighted prediction at x_0

$$\hat{y}_\lambda(x_0) = \frac{\sum_{i=1}^N K_\lambda(x_0, x_i)\, y_i}{\sum_{i=1}^N K_\lambda(x_0, x_i)} \tag{37}$$

This typically smooths training outputs y_i in a more pleasing way than does a k-nearest neighbor average, but it has obvious problems at the ends of the interval $[0,1]$ and at places in the interior of the interval where training data are dense to one side of x_0 and sparse to the other, if the target $\mathrm{E}[y|x = z]$ has non-zero derivative at $z = x_0$. For example, at $x_0 = 1$ only $x_i \leq 1$ get weight, and if $\mathrm{E}[y|x = z]$ is decreasing at $z = x_0 = 1$, $\hat{y}_\lambda(1)$ will be positively biased. That is, with usual symmetric kernels, (37) will fail to adequately follow an obvious trend at 0 or 1 (or at any point between where there is a sharp change in the density of input values in the training set).

A way to fix this problem with the Nadaraya-Watson predictor is to replace the locally-fitted constant with a locally-fitted line. That is, at x_0 one might choose $\alpha(x_0)$ and $\beta(x_0)$ to solve the optimization problem

$$\underset{\alpha \text{ and } \beta}{\text{minimize}} \sum_{i=1}^N K_\lambda(x_0, x_i)\left(y_i - (\alpha + \beta x_i)\right)^2 \tag{38}$$

and then employ the prediction

$$\hat{y}_\lambda(x_0) = \alpha(x_0) + \beta(x_0)\, x_0 \tag{39}$$

Now the weighted least squares problem (38) has an explicit solution. Let

$$\underset{N\times 2}{\boldsymbol{B}} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{pmatrix}$$

and take

$$\underset{N\times N}{\boldsymbol{W}}(x_0) = \boldsymbol{diag}\left(K_\lambda(x_0, x_1), \ldots, K_\lambda(x_0, x_N)\right)$$

then (39) is

$$\begin{aligned} \hat{y}_\lambda(x_0) &= (1, x_0)\left(\boldsymbol{B}'\boldsymbol{W}(x_0)\boldsymbol{B}\right)^{-1}\boldsymbol{B}'\boldsymbol{W}(x_0)\boldsymbol{Y} \\ &= \boldsymbol{l}'(x_0)\boldsymbol{Y} \end{aligned} \tag{40}$$

for the $1 \times N$ vector $\boldsymbol{l}'(x_0) = (1, x_0)\left(\boldsymbol{B}'\boldsymbol{W}(x_0)\boldsymbol{B}\right)^{-1}\boldsymbol{B}'\boldsymbol{W}(x_0)$. It is thus obvious that locally weighted linear regression is (an albeit x_0-dependent) *linear* operation on the vector of outputs. The weights in $\boldsymbol{l}'(x_0)$ combine the original kernel values and the least squares fitting operation to produce a kind of "equivalent kernel" (for a Nadaraya-Watson type weighted average).

Kernel Smoothing in p Dimensions A direct generalization of 1-dimensional kernel smoothing to p dimensions might go roughly as follows. For D as before, and $\boldsymbol{x} \in \Re^p$, I might set

$$K_\lambda(\boldsymbol{x}_0, \boldsymbol{x}) = D\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}_0\|}{\lambda}\right) \tag{41}$$

and fit linear forms locally by choosing $\alpha(\boldsymbol{x}_0) \in \Re$ and $\boldsymbol{\beta}(\boldsymbol{x}_0) \in \Re^p$ to solve the optimization problem

$$\underset{\alpha \text{ and } \boldsymbol{\beta}}{\text{minimize}} \sum_{i=1}^{N} K_\lambda(\boldsymbol{x}_0, \boldsymbol{x}_i)\left(y_i - \left(\alpha + \boldsymbol{\beta}'\boldsymbol{x}_i\right)\right)^2$$

and predicting as

$$\hat{y}_\lambda(\boldsymbol{x}_0) = \alpha(\boldsymbol{x}_0) + \boldsymbol{\beta}'(\boldsymbol{x}_0)\boldsymbol{x}_0$$

This seems typically to be done only after standardizing the coordinates of $\boldsymbol{x}$ and can be effective as longs as N is not too small and p is not more than 2 or 3. However for $p > 3$, the curse of dimensionality comes into play and N points usually just aren't dense enough in p-space to make direct use of kernel smoothing effective. If the method is going to be successfully used in $\Re^p$ it will need to be applied under appropriate structure assumptions.

There are several ways that have been suggested for making use of fairly low-dimensional (and thus, potentially effective) smoothing in large p problems. One of them is discussed next.

Additive Models A way to apply structure to the p-dimensional smoothing problem is through assumptions on the form of the predictor fit. For example, one might assume **additivity** in a form

$$\alpha + \sum_{j=1}^{p} g_j(x_j) \tag{42}$$

and try to do fitting of the p functions g_j and constant α. Fitting is possible via the so-called "**back-fitting algorithm**."

That is (to generalize form (42) slightly) if I am to fit (under SEL)

$$\alpha + \sum_{l=1}^{L} g_l\left(\boldsymbol{x}^l\right)$$

for $\boldsymbol{x}^l$ some part of $\boldsymbol{x}$, I might set $\widehat{\alpha} = \frac{1}{N}\sum_{i=1}^{N} y_i$, and then cycle through $l = 1, 2, \ldots, L, 1, 2, \ldots$

1. fitting via some appropriate (often linear) operation (e.g. kernel smoothing)
$$g_l\left(\boldsymbol{x}^l\right) \quad \text{to} \quad \text{"data"} \quad \left\{\left(\boldsymbol{x}_i^l, y_i^l\right)\right\}_{i=1,2,\ldots,N}$$
for
$$y_i^l = y_i - \left(\widehat{\alpha} + \sum_{m \neq l} g_m\left(\boldsymbol{x}_i^m\right)\right)$$
where the g_m are the current versions of the fitted summands,

2. setting

   g_l = the newly fitted version
   − the sample mean of this newly fitted version across all $\boldsymbol{x}_i^l$

   (in theory this is not necessary, but it is here to prevent numerical/round-off errors from causing the g_m to drift up and down by additive constants summing to 0 across m),

3. iterating until convergence to, say, $\hat{y}(\boldsymbol{x})$.

The simplest version of this, based on form (42), might be termed fitting of a "main effects model." But the algorithm might as well be applied to fit a "main effects and two factor interactions model," at some iterations doing smoothing of appropriate "residuals" as functions of 1-dimensional x_j and at other iterations doing smoothing as functions of 2-dimensional $(x_j, x_{j'})$. One may mix types of predictors (continuous, categorical) and types of functions of them in the additive form to produce all sorts of interesting models (including semi-parametric ones and ones with low order interactions).

5.2.5 Neural Networks

Another way to approach the aproximation of a potentially quite complex form for $\hat{y}^{\text{opt}} = \text{E}[y|\boldsymbol{x}]$ is through the fitting of highly flexible but parametric functional forms. This is the possibility of using **highly flexible nonlinear regression** forms. One very popular and famous such form is that of so-called "**neural networks**."

A multi-layer feed-forward neural network is a nonlinear map of $\boldsymbol{x} \in \Re^p$ to one or more real-valued y's through the use of functions of linear combinations of functions of linear combinations of ... of functions of linear combinations of coordinates of $\boldsymbol{x}$. Figure 1 is a network diagram representation of a single hidden layer feed-forward neural net with 3 inputs 2 hidden nodes and 2 outputs.

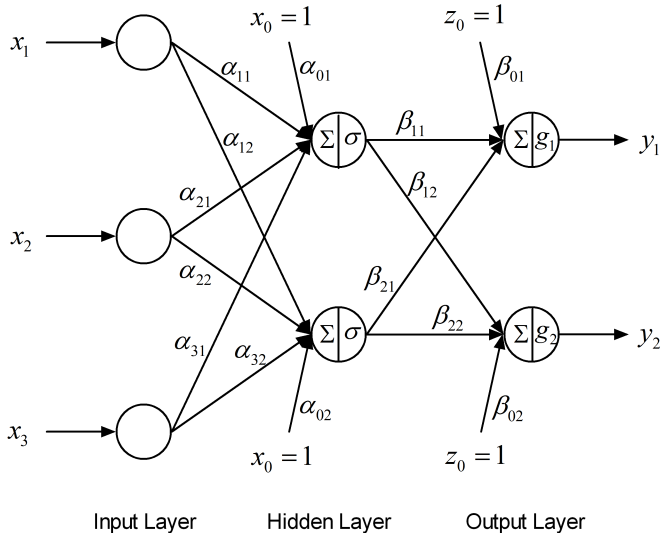

Figure 1: A Network Diagram Representation of a Single Hidden Layer Feed-foward Neural Net With 3 Inputs, 2 Hidden Nodes and 2 Ouputs

This diagram stands for a function of $\boldsymbol{x}$ defined by setting

$$z_1 = \sigma\left(\alpha_{01} \cdot 1 + \alpha_{11}x_1 + \alpha_{21}x_2 + \alpha_{31}x_3\right)$$
$$z_2 = \sigma\left(\alpha_{02} \cdot 1 + \alpha_{12}x_1 + \alpha_{22}x_2 + \alpha_{32}x_3\right)$$

and then

$$y_1 = g_1\left(\beta_{01} \cdot 1 + \beta_{11}z_1 + \beta_{21}z_2\right)$$
$$y_2 = g_2\left(\beta_{02} \cdot 1 + \beta_{12}z_1 + \beta_{22}z_2\right)$$

Of course, much more complicated networks are possible, particularly ones with multiple hidden layers and many nodes on all layers.

The most common choice of functional form σ at hidden nodes is the (sigmoidal-shaped) logistic function

$$\sigma(u) = \frac{1}{1+\exp(-u)}$$

or the (completely equivalent in this context[9]) hyperbolic tangent function

$$\sigma(u) = \tanh(u) = \frac{\exp(u) - \exp(-u)}{\exp(u) + \exp(-u)}$$

These functions are really quite linear near $u = 0$, so that for small α's the functions of $\boldsymbol{x}$ entering the g's in a single hidden layer network are nearly linear. For large α's the functions are nearly step functions. In light of the latter, it is not surprising that there are universal approximation theorems that guarantee that any continuous function on a compact subset of $\Re^p$ can be approximated to any degree of fidelity with a single layer feed-forward neural net with enough nodes in the hidden layer. This is both a blessing and a curse. It promises that these forms are quite flexible It also promises that there must be both over-fitting and identifiability issues inherent in their use (the latter in addition to the identifiability issues already inherent in the symmetric nature of the functional forms assumed for the predictors). In regression contexts, identity functions are common for the functions g, while in classification problems, in order to make estimates of class probabilities, the functions g often exponentiate one z and divide by a sum of such terms for all z's.

There are various possibilities for regularization of the ill-posed fitting problem for neural nets, ranging from the fairly formal and rational to the very informal and ad hoc. Possibly the most common is to simply use an iterative fitting algorithm and "stop it before it converges." We'll not discuss algorithms used in fitting, but note that the most common is a "back-propagation algorithm" or the "delta rule" that is a gradient descent algorithm.

In the case where one is doing SEL fitting for a single layer feed-forward neural net with M hidden nodes and K output nodes, there are $M(p+1)$ parameters α to be fit. In addition to the notation α_{0m} use the notation

$$\boldsymbol{\alpha}_m = (\alpha_{1m}, \alpha_{2m}, \ldots, \alpha_{pm})'$$

There are $K(M+1)$ parameters β to be fit. In addition to the notation β_{0k}, use the notation

$$\boldsymbol{\beta}_k = (\beta_{1k}, \beta_{2k}, \ldots, \beta_{Mk})'$$

Let $\boldsymbol{\theta}$ stand for the whole set of parameters and consider the overall SEL fitting criterion

$$R(\boldsymbol{\theta}) = \frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik} - f_k(\boldsymbol{x}_i))^2 \tag{43}$$

[9] This is because $\tanh(u) = 2\left(\frac{1}{1+\exp(-2u)}\right) - 1$.

Suppose that the various coordinates of the input vectors in the training set have been standardized and one wants to regularize the fitting of a neural net. One possible way of proceeding is to define a penalty function like

$$J(\boldsymbol{\theta}) = \frac{1}{2}\left(\sum_{l=0}^{p}\sum_{m=1}^{M}\alpha_{lm}^2 + \sum_{m=0}^{M}\sum_{k=1}^{K}\beta_{mk}^2\right) \tag{44}$$

(it is not absolutely clear whether one really wants to include $l = 0$ and $m = 0$ in the sums (44)) and seek not to partially optimize $R(\boldsymbol{\theta})$ in (43), but rather to fully optimize

$$R(\boldsymbol{\theta}) + \lambda J(\boldsymbol{\theta}) \tag{45}$$

for a $\lambda > 0$. Potentially, an appropriate value for λ might be chosen based on cross-validation.

5.3 Ensembles of SEL Predictors

Here we consider 2 versions of the basic idea of combining an ensemble of predictors in an effort to better approximate a conditional mean $\mathrm{E}[y|\boldsymbol{x}]$.

5.3.1 "Stacking"

Suppose that M predictors are available (all based on the same training data), $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_M$. Under squared error loss, one might seek a weight vector $\boldsymbol{w}$ for which the predictor

$$\hat{y}(\boldsymbol{x}) = \sum_{m=1}^{M} w_m \hat{y}_m(\boldsymbol{x})$$

is effective. One might even let $\boldsymbol{w}$ depend upon $\boldsymbol{x}$, producing

$$\hat{y}(\boldsymbol{x}) = \sum_{m=1}^{M} w_m(\boldsymbol{x})\,\hat{y}_m(\boldsymbol{x})$$

One miight pose the problem of finding

$$\widehat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \mathrm{E}^{\boldsymbol{T}}\mathrm{E}^{(\boldsymbol{x},y)}\left(y - \sum_{m=1}^{M} w_m \hat{y}_m(\boldsymbol{x})\right)^2 \tag{46}$$

or even

$$\widehat{\boldsymbol{w}}(\boldsymbol{u}) = \arg\min_{\boldsymbol{w}} \mathrm{E}^{\boldsymbol{T}}\mathrm{E}\left[\left(y - \sum_{m=1}^{M} w_m \hat{y}_m(\boldsymbol{x})\right)^2 | \boldsymbol{x} = \boldsymbol{u}\right]$$

Why might this program improve on any single one of the $\hat{y}_m$'s? In some sense, this is "obvious" since the $\boldsymbol{w}$ over which minimization (46) is done include vectors with one entry 1 and all others 0. But to indicate in a concrete setting

why this might work, consider a case where $M = 2$ and according to the $P^N \times P$ joint distribution of $(\boldsymbol{T}, (\boldsymbol{x}, y))$

$$\mathrm{E}\left(y - \hat{y}_1(\boldsymbol{x})\right) = 0$$

and

$$\mathrm{E}\left(y - \hat{y}_2(\boldsymbol{x})\right) = 0$$

Define

$$\hat{y}_\alpha = \alpha\hat{y}_1 + (1-\alpha)\,\hat{y}_2$$

Then

$$\begin{aligned}
\mathrm{E}\left(y - \hat{y}_\alpha(\boldsymbol{x})\right)^2 &= \mathrm{E}\left(\alpha\left(y - \hat{y}_1(\boldsymbol{x})\right) + (1-\alpha)\left(y - \hat{y}_2(\boldsymbol{x})\right)\right)^2 \\
&= \mathrm{Var}\left(\alpha\left(y - \hat{y}_1(\boldsymbol{x})\right) + (1-\alpha)\left(y - \hat{y}_2(\boldsymbol{x})\right)\right) \\
&= (\alpha, 1-\alpha)\,\mathrm{Cov}\begin{pmatrix} y - \hat{y}_1(\boldsymbol{x}) \\ y - \hat{y}_2(\boldsymbol{x}) \end{pmatrix}\begin{pmatrix} \alpha \\ 1-\alpha \end{pmatrix}
\end{aligned}$$

This is a quadratic function of α, that (since covariance matrices are non-negative definite) has a minimum. Thus there is a minimizing α that typically (is not 0 or 1 and thus) produces better expected loss than either $\hat{y}_1(\boldsymbol{x})$ or $\hat{y}_2(\boldsymbol{x})$.

As a practical matter, what seems common is to define

$$\boldsymbol{w}^{\text{stack}} = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N}\left(y_i - \sum_{m=1}^{M} w_m \hat{y}_m^i(\boldsymbol{x}_i)\right)^2$$

for $\hat{y}_m^i$ the mth predictor fit to $\boldsymbol{T} - \{(\boldsymbol{x}_i, y_i)\}$, the training set with the ith case removed. ($\boldsymbol{w}^{\text{stack}}$ optimizes a kind of leave-one-out cross-validation error for a linear combination of $\hat{y}_m$'s.) Then ultimately the "stacked" predictor of y is

$$\hat{y}(\boldsymbol{x}) = \sum_{m=1}^{M} w_m^{\text{stack}} \hat{y}_m(\boldsymbol{x})$$

An ad hoc version of stacking-type averaging that is probably far more common than the careful approach outlined above is choice of weight vector $\boldsymbol{w}^{\text{stack}}$ based on informal consideration of one's (CV-supported) evaluation of the effectiveness of the individual predictors $\hat{f}_m(\boldsymbol{x})$ and the (training set) correlations between them. (Averaging multiple highly correlated predictors can't be expected to be particularly helpful, and individually effective predictors should get more weight than those that are relatively speaking ineffective.) The usefulness of a given set of proposed weights might be investigated by ordinary cross-validation.

5.3.2 SEL Boosting

A different line of thinking that leads to the use of weighted linear combinations of predictors is called **boosting**. (There is a general "gradient boosting" method for other losses, including one especially crafted for 0-1 loss classification problems. Here we consider only the SEL special case, because this version is both particularly easy to understand and explain, and of high practical value. The basic idea is to try to sequentially build a good approximator for $\mathrm{E}[y|\boldsymbol{x}]$ by successively adding small corrections (based on modeling residuals) to current approximators.

A SEL boosting algorithm is:

1. Start with $\hat{y}_0(\boldsymbol{x}) = \bar{y}$.
2. With $\hat{y}_{m-1}(\boldsymbol{x})$ in hand and possible model for $y - \hat{y}_{m-1}(\boldsymbol{x})$ available, say $\beta_m h_m(\boldsymbol{x}, \boldsymbol{\gamma})$ for some given form $h_m(\cdot,\cdot)$ and unknown parameters β_m and $\boldsymbol{\gamma}_m$, fit $\hat{\beta}_m$ and $\hat{\boldsymbol{\gamma}}_m$ by least squares.
3. For some "learning rate" $\nu \in (0,1)$ set

$$\hat{y}_m(\boldsymbol{x}) = \hat{y}_{m-1}(\boldsymbol{x}) + \nu\hat{\beta}_m h_m(\boldsymbol{x}, \hat{\boldsymbol{\gamma}}_m)$$

   and return to 2.

One iterates on m from 1 through some number of iterations, M (possibly chosen by cross-validation). Commonly quoted choices for ν are numbers like .01 and the smaller is ν, the larger must be M.

SEL boosting successively corrects a current predictor by adding to it some small fraction of a predictor for its residuals. The value ν functions as a complexity or regularizing parameter, as does M. Small ν and large M correspond to large complexity. The boosting notion is different in spirit from sacking or model averaging, but like them ends with a linear combination of fitted forms as a final predictor/approximator for $\mathrm{E}[y|\boldsymbol{x}]$.

It's worth noting that this kind of sequential modification of a predictor is not discussed in ordinary regression/linear models courses because if a base predictor is an OLS predictor for a fixed linear model, corrections to an initial fit based on this same model fit to residuals will predict that all residuals are 0. In this circumstance boosting does nothing to change or improve an initial OLS fit.