Chapter 2 Notes: Statistical Learning

Statistical Learning with R

Jon Geiger

January 27, 2022

2.1: What is Statistical Learning?

Input Variables are also called *predictors*, *independent variables*, *features*, or just *variables*, denoted by X_i's. **Output Variables** are also called *responses* or *dependent variables*, denoted using the symbol Y.

With a quantitative response Y and p predictors $X = (X_1, X_2, \ldots, X_p)$, we assume there is some relationship between Y and X which takes the general form:

$$Y = f(X) + \epsilon$$

f is an unknown function of all the X_i's, and ϵ is a random *error term*, independent of X with a zero mean. f represents the *systematic* information that X provides about Y.

Statistical Learning refers to a set of approaches for estimating f.

We estimate f for two reasons: **prediction**, and **inference**.

We can *predict* Y with $\hat{Y} = \hat{f}(X)$, where $\hat{f}(X)$ represents our estimate for f, and $\hat{Y}$ is the resulting prediction of Y. Our goal is to minimize the reducible error (the error which can be changed by modifying f), keeping in mind that ϵ cannot be reduced.

In *inference*, we wish to understand the association between Y and each of the X_i's. A problem of inference could be driven by the following questions:

- Which predictors are associated with the response?
- What is the relationship between the response and the predictor?
- Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?

There are two methods to estimating f: **Parametric Methods**, and **Non-Parametric Methods**.

Parametric Methods involve a two-step, model-based approach:

1. We make an assumption about the functional form of f. One simple example might be that f is linear in X:
   $$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$
   With a linear model, we estimate $p + 1$ coefficients from β_0 to β_p.

2. After model selection, we need to *fit* or *train* the model. This is the process of estimating the parameters β_0 through β_p such that in the case of the linear model above,

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

The process is called *parametric* because we are *estimating parameters* to fit the model. This is generally simpler because it is easier to fit a set of parameters than it is to fit an entirely arbitrary function f. A parametric model has the disadvantage in that it typically does not match the true unknown form of f.

Non-parametric models have the advantage of not assuming a form for f, and can thus fit a wider range of possible shapes of f. These models need many more observations to work effectively. An example of a non-parametric model is a *thin-plate spline*, which can be either rough or smooth, where the roughness is analogous to overfitting data in a parametric model.

In choosing a model, there's a tradeoff between the *interpretability* of that model and the *flexibility* of that model. Generally, the more flexible a model is (bagging, boosting, SVMs, Deep Learning), the less interpretable the output is. Likewise, less flexible models (Subset Selection, Lasso, Least Squares) tend to be more interpretable.

- Choosing a more *restrictive* (less flexible) model is preferable in problems of inference, as the outcome needs to be interpretable.
- Conversely, more flexible models can be better for prediction problems when interpretability is not strictly necessary.

All examples from this past chapter have been examples of *supervised learning*. These techniques include linear regression, logistic regression, Generalized Additive Models (GAMs), boosting, Support Vector Machines (SVMs), and boosting. These all have in common that for a single observation x_i, there is an associated response measurement y_i.

Unsupervised learning tackles a different challenge, namely, that for each observation i, we have a set of measurements x_i with no associated response y_i. These types of problems are usually *cluster analysis* problems, where we try to find patterns, order, and/or groups in the data. More specifically, the goal is to figure whether or not the data fall into distinct groups, and if they do, which observations comprise those groups.

There do exist some problems which can be categorized as *semi-supervised learning* problems, namely scenarios in which our response data is incomplete. Namely, if we have both predictor and response measurements for m observations, where $m < n$, then for the remaining $m - n$ measurements we have predictor data but no response data.

Regression problems have a *quantitative response*. e.g. Least Squares.

Classification problems have a *categorical response*. e.g. Despite its name, logistic regression.

Some statistical methods, such as K-nearest neighbors (KNN) and boosting, can be used for either quantitative or categorical responses.

2.2: Assessing Model Accuracy

In evaluating statistical learning methods, we need some measure of how well its predictions match the observed data. In the case of regression, this is the *mean squared error* (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

A "good prediction" will yield a small MSE, as the error term $(y_i - \hat{f}(x_i))$ will be small for each observation.

With the case of linear regression, though, it is quite possible to overfit the data to perfectly capture each response (thus the MSE would be zero). We don't care how our model performs on our *training data*, but rather how it performs when we feed it *test data*. Because of this, we care about fitting parameters for $\hat{f}$ that will minimize the *test MSE* rather than the *training MSE*.

Often when choosing a model, it is useful to plot the training MSE and test MSE as a function of flexibility. Typically, the most flexible models will can yield a very low training MSE but will fail to produce a low test MSE (overfitting). Similarly, the least flexible models cannot produce a low MSE in either the training nor the test data, so the best models have a training MSE and test MSE that are roughly equal.

The test MSE for a given value x_0 can always be decomposed into the sum of three fundamental quantities:

1. The *variance* of $\hat{f}(x_0)$
2. The squared *bias* of $\hat{f}(x_0)$
3. The variance of the error terms ϵ

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \text{Var}(\hat{f}(x_0)) + \left[\text{Bias}(\hat{f}(x_0))\right]^2 + \text{Var}(\epsilon)$$

In this case, $E\left(y_0 - \hat{f}(x_0)\right)^2$ is the *expected test MSE* at x_0. If our goal is to minimize the MSE, we need to estimate f such that the learning method achieves both a *low bias* and a *low variance*. This is known as the **bias-variance trade-off**.

The **variance** of a statistical learning method refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. Ideally, the estimate for f should not vary much between training sets. This would be a *low variance*. In general, more flexible statistical methods have higher variance (overfitting to the training data).

The **bias** of a statistical learning method refers to the error that is introduced by approximating a real-life problem (usually very complicated) by a much simpler model. It's unlikely that a real-life problem actually has a linear relationship (as an example), so performing linear regression will result in some bias in estimating f.

It's very easy to achieve a model with low bias and high variance (draw a line that goes through every data point), as well as a model with high bias and low variance (fit a horizontal line through the data). Minimizing the test MSE is the key to finding the perfect bias-variance trade-off.

The bias-variance tradeoff also applies to the classification setting. In the case of classification, we assess our estimate of f with the training **error rate**, or the proportion of mistakes that are made if we apply our estimate to the training observations:

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i) \quad \text{where} \quad I = \begin{cases} 1 & \text{if } y_i \neq \hat{y}_i \\ 0 & \text{if } y_i = \hat{y}_i \end{cases}$$

In other words, I makes a binary indication of whether or not a classification error was made. Applying a classification method to test data, then, the *test error rate* with a set of test observations (x_0, y_0) is given by

$$\text{Ave}(I(y_0 \neq \hat{y}_0))$$

A good classifier is one for which the test error is minimized.

The **Bayes Classifier** is one which utilizes conditional probability to classify points in a two-class problem. Particularly, it seeks to find $Pr(Y = j|X = x_0)$, or "given that the observation is x_0, what's the probability that it belongs to class j?"

- The *Bayes error rate* is given by $1 - \max_j Pr(Y = j|X = x_0)$
- In general, the overall Bayes error rate is given by $1 - E\left(\max_j \Pr(Y = j|X = x_0)\right)$

The K**-Nearest Neighbors Classifier** (KNN) allows us to construct a decision boundary based on real data. It accomplishes this by choosing a test observation x_0, identifying the K closest points ($\mathcal{N}_0$), then estimates the conditional probability that x_0 would be in class j as the fraction of points whose response values equal j:

$$\Pr(Y = j|X = x_0) = \frac{1}{K}\sum_{i \in \mathcal{N}_0} I(y_i = j)$$

Similar to regression methods from earlier sections, KNN can be very flexible with low values of K, and for any set of training data, it can be important to optimize the value of K selected to minimize the test error.