Modern Applied Supervised Statistical Machine Learning from 30,000 Feet

Stephen B. Vardeman

Max D. Morris

V.71 July 10, 2017

Abstract

There is a single fairly simple unifying story that describes effective practice of modern supervised statistical machine learning (both "regression" type prediction and classification). Popular methodologies current in predictive analytics fit naturally into this story and their most effective roles become apparent when placed into the framework.

Context, Notation, and Basic Modeling

We consider the development of an approximator of output y based on input vector $\mathbf{x} \in \Re^p$. This is a function $f(\mathbf{x})$ for which (in some appropriate sense) typically $y \approx f(\mathbf{x})$. Where the set of possible values of y is $\Re$ it is common to speak of a "prediction" problem and where y takes values in a finite set of K elements[1], the terminology "classification" or "pattern recognition" is standard. A set of N data pairs $(\mathbf{x}_i, y_i)$ upon which to develop the predictor or classifier is called a "training set," $\mathbf{T}$, and where the training set has been employed in the development of the predictor or classifier, we'll use the notation $\hat{f}$ for it.

It is more or less standard to treat the training set as if it is a random sample from some fixed but unknown distribution, say P, so that the form of a predictor/classifier is random. To guide the development of $\hat{f}$, one may for prediction or classification $\hat{y}$ adopt a loss function

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

and for $(\mathbf{x}, y) \sim P$ independent of the training set, attempt to control the test or generalization error

$$\text{Err} = \text{E}^{(\mathbf{x},y)}\text{E}^{\mathbf{T}} L\left(y, \hat{f}(\mathbf{x})\right)$$

(where the superscripts on the expectations indicate what is being averaged out).

[1] For sake of simplicity, we'll here limit attention to the case of $K = 2$.

For illustrative purposes we'll use squared error loss (SEL)

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

for the prediction problem, and 0-1 loss

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

for classification.

For $\hat{y}(\mathbf{x})$ an arbitrary function of the input vector, straightforward consideration of

$$\mathrm{E}L\left(y,\hat{y}(\mathbf{x})\right) = \mathrm{EE}\left[L\left(y,\hat{y}(\mathbf{x})\right)|\,\mathbf{x}\right]$$

shows that if one knew P, a best predictor or classifier would be available as

$$f^{\mathrm{opt}}(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}}\,\mathrm{E}\left[L(y,\hat{y})|\,\mathbf{x}\right]$$

In the SEL case, this means that no $\hat{f}(\mathbf{x})$ can be better than (the non-realizable, as P is seen only through the training set)

$$f^{\mathrm{opt}}(\mathbf{x}) = \mathrm{E}[y|\,\mathbf{x}]$$

In the 0-1 loss case, this means that no $\hat{f}(\mathbf{x})$ can be better than

$$f^{\mathrm{opt}}(\mathbf{x}) = \underset{\hat{y}}{\arg\max}\,P[y=\hat{y}|\,\mathbf{x}]$$

In both prediction and classification problems, the general goal then becomes to use $\mathbf{T}$ in a way to make $\hat{f} \approx f^{\mathrm{opt}}$.

More Detail on 2-Class Classification

Consider the 2-class classification problem with y taking values in $\{-1,1\}$ and let $\pi_{-1} = P[y=-1]$ and $\pi_1 = P[y=1]$, and $g_{-1}(\mathbf{x})$ and $g_1(\mathbf{x})$ be class-conditional densities for $\mathbf{x}$ (densities for $\mathbf{x}|\,y$). In this notation, a non-realizable optimal classifier is

$$f^{\mathrm{opt}}(\mathbf{x}) = \mathrm{sign}\left[\frac{g_1(\mathbf{x})}{g_{-1}(\mathbf{x})} - \frac{\pi_{-1}}{\pi_1}\right] \tag{1}$$

The ratio $g_1(\mathbf{x})/g_{-1}(\mathbf{x})$ is the likelihood ratio and the shifted version of it inside the $\mathrm{sign}(\cdot)$ function might be called a "voting function." Most standard classification methods can be thought of as based on a ($\mathbf{T}$-dependent) underlying voting function that produces a classification

through examination of its sign. Good applied classification methods thus implicitly involve good empirical approximation to a function equivalent (in terms of its ordering of the values of $\mathbf{x}$) to the likelihood ratio.

In this regard, for a "voting function" $v(\mathbf{x})$, a classification error is made by the classifier $\text{sign}\left[v(\mathbf{x})\right]$ when $yv(\mathbf{x}) < 0$, and the expected 0-1 loss for such a classifier is

$$\text{E}\, I\left[yv(\mathbf{x}) < 0\right]$$

Further, if $h(u) \geq I[u < 0]$,

$$\text{E}h\left(yv(\mathbf{x})\right) \tag{2}$$

serves as an upper bound on the expected 0-1 loss for $\text{sign}\left[v(\mathbf{x})\right]$. Further, for some standard smooth choices of h (like $h(u) = \exp(-u)$) a function $v^{\text{opt}}(\mathbf{x})$ optimizing the expected function-loss (2) can be easily derived and turns out to be equivalent to the shifted likelihood ratio in display (1). The upshot of this is that good applied classification methods might also be sought in terms of voting functions approximately optimizing an empirical version of the expected function-loss (2).

Judging Optimal-Function Approximation Efficacy

Whether one seeks a function approximating the optimizer of

$$\text{E}\left(y - \hat{y}(\mathbf{x})\right)^2 \tag{3}$$

in squared error loss prediction, or an approximation to the optimizer of the expected function loss (2) in 0-1 loss classification, there is the issue of reliable judging of how effective one has been. The most reliable and absolutely standard-in-practice methodology for assessing this is cross-validation (and where computationally feasible, averaged repeated cross-validation).

One round of k-fold cross-validation consists of randomly splitting the training set into k approximately equal-sized "folds," developing the predictor or voting function on only the "remainder" of the training set for each fold, and totaling losses (values $L(y_i, \hat{y}_i)$) across the fold using that predictor or voting function on the fold. These are then totaled across folds and divided by N to get a cross-validation approximation to Err. Comparing cross-validation errors provides a sound basis for comparing predictors.

Penalized Fitting and Other Complexity Parameters

The blessing of "big data" is the potential to fit relatively flexible predictors and voting functions to a training set. The trick is to not overdo it and to find a complexity that training set will actually support, to match method flexibility to real information content in the data available. So what one needs is a spectrum of versions of a predictor to compare via cross-validation and to ultimately choose the version that seems best.

Some machine learning methods have parameters that are explicitly penalization parameters. Smoothing splines, penalized regression methods, cost-complexity tree pruning, penalized logistic-regression-for-classification, and some developments of support vector machines are examples. The penalty parameters control complexity of fit, and as they vary provide a spectrum of predictors and voting functions.

Other machine learning methods (like nearest neighbor methods, local regression smoothing, MARSs, GAMs, random forests, and LDA) have tuning parameters that are not explicitly connected to a penalty, but surely do control the flexibility of the method in a particular application. In either case, as one varies method parameters across a grid of values and examines cross-validation errors, the best version of the method for an application is identified and can subsequently be applied using the entire training set to produce $\hat{f}$.

"Boosting" and Successive Approximation of an Optimizing Function

The single most effective existing supervised machine learning methodology is "boosting." This is true whether the objective is find an approximation to $v^{\text{opt}}(\mathbf{x})$ optimizing the expected function-loss (2) and use it in classification, or to find an approximation to $\hat{y}^{\text{opt}}(\mathbf{x}) = \mathrm{E}[y \mid \mathbf{x}]$ optimizing mean squared prediction error (3). The currently wildly popular `XGBoost` algorithms represent one implementation of this notion (and a single logic suffices to cover both prediction and classification cases).

It is important to realize that "boosting" is simply the time-honored notion of successive approximation in numerical analysis applied to the optimization of an empirical version of an expected loss, based on repeated correction of an approximating function by elements of some particular set of basis functions (often trees). The earliest version of the method was the AdaBoost.M1 algorithm and Friedman's gradient boosting is the foundation of most currently popular methods.

Boosting algorithms are controlled by parameters (including number of iterations and magnitudes of correction) that must be subjected to cross-validation in order to avoid overfitting.

"Stacking," "Super-Learners," and Meta-Predictors/Classifiers

It has long been understood in practice that "ensembles" of predictors or classifiers made via different methodologies can often be "combined" to produce better predictions or classifications than produced by any single constituent. Popular predictive analytics contests (beginning with the famous Netflix competition through current Kaggle games and other such competitions) are traditionally won by "end of game" combining of leading teams who somehow pool their separately-derived predictions. There is a large (mostly misguided) literature on "classifier fusion" aimed at this phenomenon.

Applied prediction problems are typically more complex than can be handled well by any single existing methodology (except perhaps, an appropriate version of boosting). Combining predictors amounts to a way to reduce model bias/lack of fit in the face of this reality.

It has been common in SEL prediction problems to employ some kind of averaging (stacking) of multiple predictors. In classification, heuristics like "majority voting" of classifiers have been common. But recently, methods with far more potential are being tested out (in admittedly somewhat ad hoc forms) in predictive analytics contests. These might be called super-learners or meta-predictors/classifiers.

Once one recognizes that applied supervised learning is basically optimal-function approximation guided by cross-validation, it becomes obvious that the outputs (approximate voting functions or conditional means) of several methods might themselves be advantageously treated as inputs/features in a top-level prediction or classification methodology (to simply produce a better approximation than any constituent). Particularly in the classification context where different voting functions can be on different scales, a tree-based methodology (like a simple tree, a random forest, or tree-based boosting) that is invariant to monotone transforms of inputs seems most natural as the top-level method.

The practical difficulty here is honest cross-validation. The computational load implied by doing in the remainder of each of k folds what will be done with the whole training set at the end (including the fitting of all constituent predictors or classifiers and using them as inputs across a set of parameters for the top-level method) in order to get honest cross-validation errors is daunting. But without it, wildly optimistic "error" figures will be produced--that are of no help in good choice of details of the top-level method.

The optimization of parameters of a meta-prediction/classification scheme is in essence a problem in response surface analysis or even the design and analysis of computer experiments. In contrast to common formulations of those problems, the cross-validation errors for different parameter sets for a given split into folds cannot be legitimately thought of as independent random variables. So both in terms of simply making cross-validation fast enough and in terms of making sensible inferences about ultimate parameter combinations this seems like an area where research is needed and might substantially advance practice.