Machine Learning Cheat Sheet

Cameron Taylor*

November 14, 2019

Introduction

This cheat sheet introduces the basics of machine learning and how it relates to traditional econometrics. It is accessible with an intermediate background in statistics and econometrics. It is meant to show people that machine learning does not have to be hard and mysterious! Descriptions are favored over illustrations though in future iterations examples and illustrations could be added.

Concepts

Machine learning methods are statistical methods, and if you are familiar with statistical methods quite well then much of machine learning is not mysterious. The major concept necessary to machine learning methods that is not involved in traditional statistical or econometric methods is the idea of **tuning** and **cross-validation**. In machine learning methods, it is often the case that there is some parameter λ such that the objective function (of the data and parameters) being minimized to estimate the parameters is also a function of λ. Thus, one needs to select λ before estimating the parameters. This is usually done by cross-validation, explained below.

Method Details

Cross-Validation

I cover the most popular form of cross-validation: K**-fold cross validation**. This works as an input into finding λ that will be used in estimation. Split the data randomly into K roughly equally sized

*Stanford Graduate School of Business. Email: cntaylor@stanford.edu

bins B_k. For each $k = 1, \ldots, K$ and each value of the tuning parameter λ_m over a grid considered, compute the validation error

$$E_k(\lambda) = \sum_{i \in B_k} (y_i - \hat{f}_\lambda^{-k}(x_i))^2 \tag{1}$$

where $\hat{f}_\lambda^{-k}(x_i)$ is the estimate at tuning parameter λ leaving out data in B_k.

Then the total CV objective function is

$$CV(\lambda) = \frac{1}{N} \sum_{k=1}^{K} E_k(\lambda) \tag{2}$$

and minimizing this is how we choose λ. When $K = N$ this is called *leave-one-out* cross validation. In practice most people choose K around 10 for computational reasons.

Supervised Learning and Regression

These are problems concerning estimating $\mathbb{E}[Y|X]$.

- OLS: Standard, many tools and resources for this.
- LASSO: Start from OLS but think about penalizing the number of coefficients that are larger than 0. The estimator is

$$\hat{\beta}_{\text{lasso}} = \text{argmin}_\beta \left\{ \frac{1}{2} \sum_{i=1}^{N} (y_i - X_i\beta) + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{3}$$

  where p is the total number of variables not a constant. This is the **L1** penalty.

  λ is chosen by cross-validation: usually 10-fold cross validation is preferred.

- Ridge: Start from OLS but again think about penalizing the number of coefficients larger than 0. The estimator is

$$\hat{\beta}_{\text{ridge}} = \text{argmin}_\beta \left\{ \frac{1}{2} \sum_{i=1}^{N} (y_i - X_i\beta) + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{4}$$

  where p is the total number of variables not a constant. This is the **L2** penalty.

- LASSO vs. Ridge: The difference is hard vs. soft thresholding (Statistical Learning Book

Figure 3.11, Table 3.4). Suppose that the columns of X are orthonormal. Then

$$\hat{\beta}_{j,\text{lasso}} = \text{sign}(\hat{\beta}_{j,\text{ols}})(|\hat{\beta}_{j,\text{ols}}| - \lambda)_+$$
$$\hat{\beta}_{j,\text{ridge}} = \hat{\beta}_{j,\text{ols}}/(1+\lambda)$$

As well computational trade-offs: ridge has closed form solution, lasso more computationally intensive. LASSO more popular for feature selection when you think the underlying data generating process or structural model is sparse and you have a large number of variables.

- Elastic Net: Combine LASSO and Ridge to get

$$\hat{\beta}_{\text{elastic net}} = \text{argmin}_\beta \left\{ \frac{1}{2}\sum_{i=1}^{N}(y_i - X_i\beta) + \lambda\Big(\alpha\sum_{j=1}^{p}|\beta_j| + (1-\alpha)\sum_{j=1}^{p}\beta_j^2\Big) \right\} \tag{5}$$

where α is usually chosen without CV (pre-specified), although can be maximized over jointly for two-dimensional surface CV. This is useful because when p is large compared to N it overcomes the problem that LASSO has of selecting at most N covariates for the model.

- Kernel-Based Regression:

Kernel Regression fits the regression function as

$$\hat{f}_{\text{kernel}}(x) = \frac{\sum_i K(\frac{x-X_i}{h})y_i}{\sum_i K(\frac{x-X_i}{h})} \tag{6}$$

where h is the bandwidth chosen by cross-validation and $K(\cdot)$ is a kernel function which satisfies a few properties. To fix ideas, think of this as a standard normal density. Note that this fits *constants* weighted by the kernel around y_i.

Local-Linear/Polynomial Regression generalizes kernel regression by estimating

$$\hat{f}_{\text{local}}(x) = \min_{\beta_0,\beta_1}\sum_i K(\frac{x-X_i}{h})(y_i - \beta_0 - \beta_1(x-X_i))^2 \tag{7}$$

There is a closed form solution for the local linear function. Local polynomial generalizes to higher order polynomials of $(x - X_i)$.

- Basis Functions and Splines: A general basis expansion is to pick M and then write

$$f(X) = \sum_{m=1}^{M}\beta_m h_m(X)$$

where $h_m : \mathbb{R}^p \to \mathbb{R}$ are transformations.

When we choose the h_m in a specific way we call it *splines*. In particular, splines have *knots* at a vector of points in the feature space ξ that allow us to interpolate continuously between the different regions defined by the knots. A common spline choice is the natural cubic spline which with K knots which has basis functions:

$$N_1(X) = 1, N_2(X) = X, N_{k+2}(X) = d_k(X) - d_{K-1}(X)$$

where $d_k(X) = \frac{(X-\xi_k)_+^3 - (X-\xi_K)_+^3}{\xi_K - \xi_k}$. The natural cubic spline is nice because it mitigates boundary bias (bias at the boundaries of the feature space or X's) by forcing the estimated function to be linear beyond the boundary knots.

Note that these procedures have many tuning parameters: degree of splines, number of knots and where to place the knots. Multivariate Adaptive Regression Splines get over this problem by greedily selecting the knots in a fashion similar to regression trees.

- Additive Models: The generalized additive model is given by

$$\hat{f}_{\text{additive}}(X) = \sum_{j=1}^{p} f_j(X_j) \tag{8}$$

and we aim to minimize the sum of squared errors. This model can be fit with the *backfitting algorithm* which consists of iteratively applying a smoother (e.g. kernel regression, local linear, etc.) to each dimension j and updating. Thus we do, in some senses, parameterize this model by picking smoothers to estimate each separate f_j.

- Regression Tree: Regression trees consist of splitting the feature space into regions that predict a constant for the dependent variable in each region. Thus the regression tree adaptively selects both the variable/feature j and a split point in that variable s to minimize the sum of squared errors, predicting at the mean in each region. Cross validation is done on the number of splits J we allow.

  Pruning: To make sure to pick up important relationships, it is often helpful to grow a "big" tree first and then prune the tree by removing splits to optimize.

  Bump Hunting/PRIM: For classification, can adapt regression tree like methods to find modes/maximums of the dependent variable in the feature space.

- Random Forest: Random forests add smoothness to regression trees by introducing randomness in two ways. Consider applying a regression tree multiple times on bootstrapped

samples of the data and where, at each step, we consider only a random subset of the independent variables to split on. Then the resulting random forest estimate is the average over these bootstrapped regression trees.

- Boosting: The general idea behind boosting is to apply a simple, even naive method, iteratively to improve the fit. The method is iteratively applied to residuals from the previous methods. At its most basic level boosting fits an additive model. The number of times boosted M is a tuning parameter here. In particular, at any iteration m the following is fitted

$$\min_{\gamma_m} \sum_i L(y_i, \hat{G}_{m-1}(X_i) + g(X_i; \gamma_m)) \tag{9}$$

  where L is the loss function, $\hat{G}_0(x) \equiv 0$ and $g(\cdot)$ is the single method for each boosting step (e.g. single split tree). One can weight the resulting g_m by ϵ_m in the overall additive prediction. Tuning usually involves: number of boosts/iterations to apply, some CV on the simple method (ex: how many splits in the tree), and the learning rate ϵ_m.

  AdaBoost: A particular popular example of a boosting algorithm for classification. This fits a classification method and then weights each classification (in overall additive method) based on the computed error of the iterative step. It uses exponential loss. Many different loss functions can be considered for boosting. The difference between AdaBoost and Gradient Boosting is that in AdaBoost a single weak learner is applied sequentially where the only thing that varies among iterates is the weighting of the different data points for prediction based on previous error. The final prediction function is a weighted sum of all the weak learners (like Gradient boosting). The weights are varied to adapt to "more difficult cases".

- Neural Nets: Neural nets are large parametric models. They consist of hidden layers and nodes in each layer. There is also an input layer and an output layer. Each node is a linear function of terms that are non-linear transformations of nodes inn the previous layer. So, if, for example, $g(\cdot)$ is linear, then it is just a very large linear model with many interactions. No asymptotic theory exists and value is in ability to estimate with thousands of parameters. To estimate, regularize the linear parameters for each node, and pick the regularization penalty through CV.

Supervised Learning and Classification

These are problems concerning estimating $\mathbb{P}(Y|X)$ or alternatively classifying Y into buckets given X.

- Logistic Regression: Standard, many tools and resources for this. An elegant way to perform estimation is through iteratively weighted least squares - each step solves a different least squares problem. Like linear regression we can add regularization to logistic regression with a regularization parameter λ.

- Regression Tree for Classification: When doing classification, we change the loss function. In particular, look at the **impurity** of leaves in the tree, which measures how varied the classifications/predictions are in a leaf. Then the loss function for classification is a weighted average over the impurities of each leave. Two standard impurity measures are Gini Impurity and Information Criterion.

- Neural Network for Classification: Same but the output layer is M dimensional for each class, then run a logit on the output layer to get a classification prediction.

- Linear Discriminant Analysis: Suppose each class k has conditional density of X given in class k of $f_k(x)$ and prior probabilities then we can write

$$\mathbb{P}(Y = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{l=1}^{K} f_l(x)\pi_l}$$

  Then LDA models each class density as a multivariate Gaussian density with a common covariance matrix. Then the linear discriminant functions are

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k \tag{10}$$

  and then we classify based on the largest δ_k for each x. We use the training data to estimate π_k, μ_k and Σ. These estimators come from the analogy principle. This estimator is closely related to linear regression.

  We can generalize LDA to QDA by assuming that classes have different covariance matrices. Then the decision boundaries between classes are defined by quadratic equations. Note that the number of parameters to estimate can increase by a lot. One way to approximate QDA is to add dimensions to the LDA with quadratics.

- Support Vector Machines: Consider the case of binary classification $y_i \in \{-1, 1\}$. A hyperplane is defined by

$$\{x : x'\beta + \beta_0 = 0\}$$

  A SVM classifier classifies based on what sign/side of the hyperplane the x's put it on.

  In some cases we may be able to perfectly separate the classification space. If we can do that

then we pick the hyperplane that "puts biggest distance between the classes". This is useful for out-of-sample predictions and generalizing the model.

The method is called support vectors because the parameter estimates forming the hyperplane depend on only a few points (vectors) that are pivotal for forming the hyperplane. In particular $\hat{\beta} = \sum_{i \in \mathcal{S}} \hat{\alpha}_i x_i$ where $\mathcal{S}$ is the set of support vectors.

In general, we can write the problem as

$$\min_{\alpha} \sum_{i=1}^{N} \left[1 - y_i \Big(\alpha_0 + \sum_{j=1}^{N} \alpha_j x_j^T x_i \Big) \right]_+ + \lambda \sum_{i,j} \alpha_i x_i^T x_j \alpha_j \tag{11}$$

where λ is a regularization parameter and $a_+ = \mathbf{1}\{a > 0\} \cdot a$.

We can make the classifier more flexible by replacing $x_j^T x_i$ in the estimation problem with kernel functions $K(x_j, x_i)$ to allow for non-linear interactions.

Unsupervised Learning

These are problems concerning grouping X's with no dependent variable.

- Principal Components: Provides sequence of best linear approximations to the data by essentially fitting a q dimensional hyperplane to explain the variance in X. A modern alternative for non-negative data is matrix factorization.

- k means clustering: Goal is to categorize each data point x_i into one of K clusters $C(i) \in \{1, \ldots, K\}$. Suppose that the X's are quantitative. Define the **dissimilarity measure** between two points x_i and x_j as

$$d(x_i, x_j) = ||x_i - x_j||^2$$

Then to estimate consider the *within cluster scatter* for a certain dissimilarity measure

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(j)=k} d(x_i, d_j)$$

In this case this simplifies nicely and has a nice zig-zag algorithm to optimize where we pick the means of clusters and then categorize points, iterating the process. We can generalize $d(\cdot, \cdot)$ to avoid issues with outliers and large quantitative values - the estimation algorithm still works well. Note that we cannot do CV on K here, we must choose it. One way

to choose it is to plot $\log(W_K)$ as a function of K and look for a "gap". We can also do hierarchical clustering to avoid having to pick K altogether.

- Multidimensional Scaling: Given data or even a dissimilarity index, look for low representation representation of the data $z = (z_1, \ldots, z_n)$ by minimizing stress function

$$S_M(z) = \sum_{i \neq j} (d_{ij} - ||z_i - z_j||)^2$$

with alternatives. The idea is that the low representation will still preserve the distances between the points. Has recently been expanded to non-linear setup.

- Google PageRank: Consider pages that link to each other. Let $L_{ij} = 1$ if page j links to page i. Let $c_j = \sum_i L_{ij}$ be total links. Then page rank is defined recursively by

$$p_i = (1 - d) + d \sum_{j=1}^{N} \frac{L_{ij}}{c_j} p_j \tag{12}$$

where the idea is that a page is important if important pages link to it. In matrix form this can be written as

$$p = (1 - d)e + dLD_c^{-1}p = [(1 - d)ee^T/N + dLD_c^{-1}]p = Ap$$

where the second equality uses a normalization the average page rank to be 1. Then A has largest real eigenvalue 1 so we can use the power method to find the fixed point. The reason this works is that it forms an irreducible aperiodic Markov chain.

- Generative Adversarial Networks: Suppose that we have real observations with empirical distribution $\hat{F}_X(\cdot)$ in $\mathbb{X}$. Then consider some noise distribution $F_Z(\cdot)$ (e.g. multivariate normal) in $\mathbb{Z}$. We require a generator $g_\theta : \mathbb{Z} \to \mathbb{X}$. For example, this can be a neural network or another flexible model. Finally we require a discriminator/critic to tell fake and real data apart, this can also be a neural network or other flexible classifier. Then the goal of GAN is to find a θ so that $g_\theta(Z) \sim \hat{F}_X$ according to the discriminator/critic.

  We can generally write this as a min-max problem where the inner maximum problem is for fitting the classifier and the outer minimization is to minimize the optimization criteria so that the generator makes it very difficult to distinguish between the data generating processes. Then we can estimate by updating the classifier and then using gradient descent to update the GAN.

  In general, often simplify the critic/discriminator step to some measure between probability

distributions. Common ones are KL, Jensen-Shannon and Wasserstein which has received the most attention probably. Wasserstein has the form (ignoring a Lipschitz constraint)

$$\min_{\theta} \max_{\phi} \left[\frac{1}{N_F} \sum_{i:Y_i=F} f(g_\theta(Z_i), \phi) - \frac{1}{N_R} \sum_{i:Y_i=R} f(X_i, \phi) \right] \tag{13}$$

where N_F is the number of fake/artificial obs and N_R is the number of real obs. Here f is still the classifier with parameter ϕ to be estimated.