Elements of Statistical Learning Notes

Dustin

March 18, 2021

Contents

This classic text was recommended to me by one of the Lead Data Scientists I used to work with. Here are notes taken upon reading through to gain a deeper understanding of its content.

1 Regression with Regularization

1.1 Least Angle Regression (LAR)

p74

An efficient LASSO.

$$\begin{aligned} \delta_k &= \left(X_{A_k}^T X_{A_k}\right)^{-1} X_{A_k}^T r_k \text{ Similar to } \hat{\beta} \\ r_k &= y - X_{A_k}\beta_{A_k} \\ \beta_{A_k}(\alpha) &= \beta_{A_k} + \alpha u_k \\ u_k &= X_{A_k}\delta_k \text{ New Direction} \end{aligned} \tag{1}$$

Initial $r = y - \bar{y}$

u_k makes the smallest (and equal) angle with each predictor in A_k. Hence, "Least Angle" Regression.

1. Questions

   - The equations seem circular?
   - What is Arc Length with respect to L_1 in LAR/LASSO?

     Do more research on this. Still don't fully understand it in a way that I can explain it well.

2 Bootstrap Method

Similar to Cross Validation. Creates B datasets from training data by sampling with replacement. Statistics can be calculated from each B_i.

For example, $V\left[S(z)\right] = \frac{1}{B-1}\sum_{b=1}^{B}\left(S(X^{*b}) - \bar{S}^{*}\right)^2$ where $S(z)$ is a quantity computed from B_i

To prevent overfitting, values present in original training set and a given B_i need to be removed. This is not a problem with Cross Validation since each K is non-overlapping.

Estimated Error

$$\hat{err}^{-1} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum^{b\in C^{-i}} L(y_i, \hat{f}^{*b}(x_i)) \tag{2}$$

C^{-i}: set of indices of bootstrap samples that do not contain i.

$|C^{-i}|$: number of samples described above.

Subject to training set bias so a 0.632 estimator can be used to alleviate this bias, *which is the average number of distinct observations in each B.*

2.1 0.632

$$\begin{aligned} P(\text{observation } i \in \text{Bootstrap Sample } b) =& 1 - (1 - \frac{1}{N})^N \\ =& 1 - e^{-1} \\ =& 0.632 \end{aligned} \tag{3}$$

$\hat{err}^{(0.632)} = 0.368 \cdot \bar{err} + 0.632 \cdot \hat{err}^{(1)}$

This works in light fitting situations by can break in overfit ones. However, this can be improved by taking into account he amount of overfitting.

2.2 Relative Overfitting Rate

$\hat{R} = \frac{\hat{err}^{(1)} - \bar{err}}{\hat{\gamma} - \bar{err}}$

$\hat{R} \in (0, 1)$

0: No overfitting 1: if overfitting equals no-info values

γ: no information error rate. The error rate if inputs and class labels are independent.

$\hat{\gamma} = \frac{1}{N^2}\sum_{i=1}^{W}\sum_{i'=1}^{N} L(y_i, \hat{f}(x_{i'}))$

3 Bayes Error Rate

Irreducible error (ϵ)

$$1 - E\left[\underset{j}{max}\ P(Y = j|X)\right]$$

aka 1 - the expected value of the class with the highest probability

4 Partial Dependence Plots

p353.

Shows marginal effect one or two features have on predicted outcomes. Always linear in a linear regression model. Can be used to diagnose potential interaction variables.

5 Trees

Trees lump observations y_i into m terminal nodes which represent a region, R_m. The ideal tree best predicts $\hat{y}_i$ based on predictors, X. Most tree building algorithms are greedy, in that they iterate over all the data. The gist is to determine the **best split points (s)** on predictors j.

Let $R_1, ..., R_M$ be M regions of the data, and C_m model the response as a constant for each R_m

$$f(x) = \sum_{m=1}^{M} C_m\ I(x \in R_m)$$

Using Least Squares minimization $\left[\Sigma(y_i - f(x_i))^2\right]$, the best $\hat{C}_m = ave(y_i|x_i \in R_m)$

The best partition is found by solving:

$$\underset{j,s}{min}\left[\underset{c_1}{min}\sum_{x_i \in R_1(j,s)}(y_i - c_1)^2 + \underset{c_2}{min}\sum_{x_i \in R_2(j,s)}(y_i - c_2)^2\right]$$

Assuming a binary tree. Thus,

$$\begin{aligned}\hat{c}_1 &= ave(y_i|x_i \in R_1(j,s)) \\ \hat{c}_2 &= ave(y_i|x_i \in R_2(j,s))\end{aligned} \tag{4}$$

This process is repeated for each R_m until a stopping point is reached.

In determining how large to grow the tree, the best practice is to grow a large tree then prune it using **cost-complexity pruning**.

Let,

T_0: Large Tree

$T \subset T_0$: T is a subset of T_0

$|T|$: number of terminal nodes. i.e. R_m

N_m: $\#\{x_i \in R_m\}$

$\hat{C}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$ $Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$

Then **cost-complexity criterion**

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|, \ \alpha \geq 0$$

Goal: For each α, find $T_\alpha \subseteq T_0$ that minimizes $C_\alpha(T)$

α: governs trade-off between tree size and goodness-of-fit to the data.

Large α means smaller trees T_α

The smallest subtree, T_α, is chosen by **weakest-link pruning**. Collapse nodes that yield smallest increase to $\Sigma_m N_m Q_m(T)$ until you have single node trees. Final tree is $T_{\hat{\alpha}}$. Find best α using CV.

5.1 Classification

$Q_m(T)$ represents impurity in a node. Cannot use Least Squares as in regression trees.

Differentiable functions are better from an optimization perspective. They are often used for *growing* trees.

5.1.1 $Q_m(T)$

1. Misclassification Error

$$\frac{1}{N_m} \sum_{i \in R_m} I(y \neq K(m)) = 1 - \hat{p}_{mk}(m)$$

   This should be used for **cross-complexity pruning**.

2. Gini Index

$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}'_{mk} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Differentiable

3. Cross Entropy/Deviance

$$-\sum_{k=1}^{K} \hat{p}_{mk}\ log(\hat{p}_{mk})$$

where $\hat{p}_{mk} = \frac{1}{N_m} \sum_{xx_i \in R_m} I(y_i = k)$. The proportion of class k in node m.

Differentiable

5.2 Misc

Unordered Categorical Predictors: Ordered by proportion in node m and pruned. This favors predictors with high cardinality which can overfit models.

6 Boosting

Additive Boosting is a technique that weights observations according to how accurate they are.

$$G(x) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

G: classifier. $G(x) \in \{-1, 1\}$ x: predictor matrix M: Iterations of modified datasets α_m: mth weight $G_m(x)$: mth classifier

Boosting is a way of fitting an additive expansion using a set of elementary *basis* functions.

General Form for Basis Function expansion

$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$

β_m: expansion coefficients x: predictors $b(x; \gamma_m)$: Simple function taking a matrix X and an m-number of γ parameters.

γ_m varies by function.

Typical case is to minimize a loss function over training data. Such as squared error or a likelihood-based function.

$$\underset{\{\beta_m, \gamma_m\}_1^m}{arg\ min} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right)$$

6.1 AdaBoost Algorithm

1. Initialize observation weights $w_i = \frac{1}{N},\ i = [1, N]$
2. For m = 1 to M:
   (a) Fit a classifier $G_m(x)$ to training data using w_i
   (b) Compute
   $$err_m = \frac{\sum_1^N w_i\ I(y_i \neq G_m(x_i))}{\sum_1^N w_i}$$
   (c) Compute
   $$\alpha_m = log\left(\frac{1 - err_m}{err_m}\right)$$
   (d) Set w_i. This increases the relative influence for the next classifier
   $$w_i = w_i \cdot exp\left(\alpha_m \cdot I(y_i \neq G_m(x))\right),\ i = [1, N]$$
3. Output
   $$G(x) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

6.2 Forward Stage-wise Additive Modeling

Approximates above by adding new basis functions without modifying old ones. For each m, the optimal basis function $b(X, \gamma_m)$ and coefficient β_m is solved for and added to the existing model.

See p342.

6.2.1 Algorithm

1. Initialize $f_0(x) = 0$
2. For m = 1 to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \underset{\beta,\gamma}{arg\ min} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + \beta\ b(x_i; \gamma)\right)$$
   (b) Set
   $$f_m(x) = f_{m-1}(x + \beta_m\ b(x; \gamma_m))$$

The additive expansion of AdaBoost estimates one-half of the log-odds $P(Y = 1|X)$. Which is why the sign can be used.

6.3 Loss Functions

6.3.1 Classification

1. Exponentional

$$L(y_i, f(x)) = exp(-yf(x))$$

   - Monotone decreasing function of the margin: $yf(x)$
   - Concentrates influence on large negative margins
     - Margins are to residuals as classification is to regression.

   The goal is to produce positive margins as frequently as possible. Penalize negative margins heavily.

2. Binomial Deviance

$$L(y, f(x)) = log\,(1 + exp(-2yf(x)))$$

   - more even influence across negative and positive margins. Thus more robust in noisy settings where **Bayes Error Rate** is not close to 0. In these cases, AdaBoost degrades.

6.3.2 Squared-Error Loss

$$L(y, f(x)) = (y - f(x))^2$$

Minimizer: $2 \cdot P(Y = 1|x) - 1$

Not good for classification because for margins $yf(x) > 1$, Loss increases quadraticly placing influence correctly on classified items rather than unclassified items.

6.3.3 Huberized Square Loss

- Combines properties of Binomial Deviance, quadratic, and SVM Hinge Loss.
- Useful alternative to Binomial Deviance to computation purposes

6.3.4 K-Class Multinomial Deviance Loss

$$L(y, f(x)) = -\sum_{k=1}^{K} I(y = G_k) f_k(x) + log\left(\sum_{l=1}^{K} exp(f_l(x))\right)$$

Like Binomial Deviance but for multiple classes.

6.4 Regression

6.4.1 Squared Error Loss

$$L(y, f(x)) = (y - f(x))^2$$

$f(x) = E(Y|x)$ is the minimizer for the population; however, emphasis is placed on observations with large absolute residuals: $|y_i - f(x_i)|$ during fitting.

Less robust for long-tailed error distributions and to outliers.

6.4.2 Absolute Error Loss

$$L(y, f(x)) = |y - f(x)|$$

More robust than squared error loss for long-tailed distributions and outliers.

6.4.3 Huber Loss

$$L(y, f(x)) = \begin{cases} [y - f(x)]^2, & |y - f(x)| \leq \delta \\ 2\delta|y - f(x)| - \delta^2, & \text{else} \end{cases}$$

Robust to outliers and heavy-tailed error distributions.

6.5 Boosting Trees

Trees partition all joint-predictor values into J regions such that

$$f(X) = \gamma_j$$

where γ_j is a constant for the jth predictor region and $x \in R_j$ where R_j is the jth predictor region. A tree can be represented as

$$T(x; \Theta) = \sum_{1}^{J} \gamma_i I(x \in R_j)$$

where $\Theta = \{T_j, \gamma_j\}_1^J$

Minimizer for Θ: $\hat{\Theta} = \underset{\Theta}{arg\,min} \sum_{j=1}^{J} L(y_i, \gamma_j)$

Model

$$f_M(x) = \sum_{m=1}^{M} T(X; \Theta_m)$$

where f_M is a sum of trees.
For each tree, solve

$$\hat{\Theta}_m = \underset{\Theta_m}{arg\,min} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i)) + T(x_i; \Theta_m)$$

for region Θ_m and constants

$$\hat{\gamma}_{jm} = \underset{\gamma_{jm}}{arg\,min} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x) + \gamma_{jm})$$

.

Used Squared-Error Loss for regression. Fairly simple since it's a single tree as in the Tree Section.

For 2-class classification, use exponential loss in AdaBoost.

Using absolute Error or Huber loss functions more robust than Squared Error for regression. Deviance more robust than exponential for classification but the algorithms are not fast.

1. Steps

   (a) Find R_j

   - Difficult to do so approximates are found
   - Greedy, top-down algorithms to find splits.
   - Sometimes the following must be used

   $$\tilde{\Theta} = \underset{\Theta}{arg\,min} \sum_{1}^{N} \tilde{L}(y_i, T(x_i; \Theta))$$

   Then given $\hat{R}_j = \tilde{R}_j$, γ_j can be estimated more precisely.

   (b) Find γ_j given R_j Typically $\hat{\gamma}_j = \bar{y}_j$ unless approximate above is used.

6.5.1 Numerical Optimization

As long as a loss function is differentiable, it can be minimized.

Let

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

where $f(x)$ is a sum of trees.

$$\hat{f} = \underset{f}{arg\ min}\ L(f)$$

where $f = \{f(x_i), ..., f(x_N)\}^T$

Essentially, $\hat{f}$ is solved for by adding component vectors based on an increment h_m.

$$f_M = \sum_{m=0}^{M} h_m,\ h_m \in \mathbb{R}^N,\ f \in \mathbb{R}^N$$

1. Steepest Descent

$$\begin{aligned} h_m &= -\rho_m g_m \\ \rho_m &= \underset{\rho}{arg\ min} L(f_{m-1} - \rho g_m) \text{ scalar} \\ g_{im} &= \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} \text{ gradient of } L(f) \text{ is evaluated at } f = f_{m-1} \\ g_m &\in \mathbb{R}^N \end{aligned} \tag{5}$$

Then $f_m = f_{m-1} - \rho_m g_m$ is updated.

This is considered very greedy. $L(f)$ is **unconstrained** in the direction of maximal descent.

6.6 Gradient Boosting

Tree predictions ($T(x; \Theta_m)$) are analogous to components of negative gradient. The key difference from **Steepest Ascent** is the tree components are not independent. $t_m = \{T(x_1; \Theta_m), ..., T(x_N; \Theta_m)\}^T$ are constrained to be predictions of a J_m-terminal node decision tree.

Since the model will be used to fit data other than the training data, the negative gradient cannot be used outright; however, t_m can be fit using Least Squres.

$$\tilde{\Theta}_m = arg \min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2$$

e.g. Fit a tree T to negative gradient values using Least Squares.

6.6.1 Loss Functions

Name	Setting	Loss Fn	Gradient
Squared-Error Loss	Regression	$\frac{1}{2}[y_i - f(x)]^2$	$y_i - f(x_i)$
Absolute-Error Loss	Regression	$abs(y_i - f(x_i))$	$sign(y - f(x_i))$
Huber	Regression	Huber	$\begin{cases} y_i - f(x_i), & abs(y \\ \delta_m \cdot sign(y_i - f(x_i)), & abs(y \\ \delta_m = \alpha th \text{ quantile } \{abs(y_i - f(x_i))\} \end{cases}$
Deviance	Classification	Dviance	kth component: $I(y_i = G_k) - \rho_k(x_i)$ where ρ_l

6.6.2 Parameters

1. J The number of max nodes for a tree to grow. e.g. J = 2 is called a stump since its a tree with 2 terminal nodes.

   In general, $4 \leq J \leq 8$ is found to be the ideal range but there's seldom improvement over $J \simeq 6$.

   This tends to reflect degrees of interaction but not always. For example, J = 2 means no interaction terms, J = 3 means 1st order interactions, etc.

2. M

   Number of iterations to run the boosting algorithm through.

3. Learning Rate

   Penalizing a gradient with a shrinkage parameter can bring down test deviance and misclassification error, at the expense of higher M (or iterations)

$$f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J} I(x \in R_{jm})$$

$0 < \nu < 1$

6.6.3 Algorithm

1. Initialize $f_0(x) = \underset{\gamma}{arg\ min} \sum_1^N L(y_i, \gamma)$ This is called the optimal constant (or a single-node tree)
2. For m = 1 to N. For classification, this step is repeated K times.
   (a) $i = [1, N]$. Compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$
   These are called pseudo/generalized residuals.
   (b) Fit regression tree to targets r_{im} giving terminal regions $R_{jm},\ j = [1, J_m]$
   (c) For $J = [1, J_m]$, Compute
   $$\gamma_{jm} = \underset{\gamma}{arg\ min} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{m=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
3. Output $\hat{f}(x) = f_M(x)$. For classification, K tree expansions are produced.

6.6.4 Stochastic Gradient Boosting

Sampling a fraction η of the training data to grow the next tree. This is usually $\eta = \frac{1}{2}$ but can be smaller for Large N. This can improve Deviance and Error but must be used with shrinkage, otherwise it makes prediction error worse.