1 Notes on The Elements of Statistical Learning

By *Weikai Mao*

Notes on The Elements of Statistical Learning

2 3. Linear Methods for Regression

3 3.1 Introduction

A linear regression model assumes that the regression function $E(Y|X)$ is linear in the inputs $X_1, \ldots, X_p$. The linear regression model has the form $\hat{Y} = f(X) = \underset{n*p}{X} \underset{p*1}{\beta} = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$.

The residual sum of squares: $RSS(\beta) = ||Y - \hat{Y}||_1^2 = ||Y - X\hat{\beta}||_1^2 = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$.

Estimator $\hat{\beta} = \underset{\beta}{\operatorname{argmin}}\, RSS(\beta) = (X^T X)^{-1} X^T Y$.

Hat matrix $H = X(X^T X)^{-1} X^T$.

Assume $\epsilon \sim N(0, \sigma^2)$, then $\hat{\beta} \sim N(\beta, (X^T X)^{-1} \sigma^2)$.

More on *The elements of Statistical Learning* Page 47-49.

3 3.2 Linear Regression Models and Least Squares

4 3.2.2 The Gauss–Markov Theorem

Least squares estimates of the parameters β have the smallest variance among all linear unbiased estimates.

3 3.3 Subset Selection

There are two reasons why we do variable subset selection:

- The first is **prediction accuracy**: the least squares estimates often have low bias but large variance. Prediction accuracy can sometimes be improved by shrinking or setting some coefficients to zero. By doing so we sacrifice a little bit of bias to reduce the variance of the predicted values, and hence may improve the overall prediction accuracy.
- The second reason is **interpretation**. With a large number of predictors, we often would like to determine a smaller subset that exhibit the strongest effects.

4 3.3.2 Forward- and Backward-Stepwise Selection

Forward-stepwise selection

Forward-stepwise starts with the intercept, and then sequentially adds into the model the predictor that most improves the fit. It is a greedy algorithm, producing a nested sequence of models.

It can be used even when $p >> N$. It is a more constrained search, and will have lower variance, but perhaps more bias. Updating algorithms can exploit the QR decomposition for the current fit to rapidly establish the next candidate (Exercise 3.9).

Backward-stepwise selection

Backward-stepwise starts with the full model, and sequentially deletes the predictor that has the least impact on the fit.

The candidate for dropping is the variable with the smallest Z-score (Exercise 3.10).

We can implement hybrid stepwise-selection strategies that consider both forward and backward moves at each step, and select the “best” of the two.

4 3.3.3 Forward-Stagewise Regression

Forward-stagewise regression (FS) is even more constrained than forward-stepwise regression. It starts like forward-stepwise regression, with an intercept equal to $\bar{y}$, and centered predictors with coefficients initially all 0. At each step the algorithm identifies the variable most correlated with the current residual. It then computes the simple linear regression coefficient of the residual on this chosen variable, and then adds it to the current coefficient for that variable. This is continued till none of the variables have correlation with the residuals—i.e. the least-squares fit when N > p.

Forward stagewise can take many more than p steps to reach the least squares fit, but this "slow fitting" can pay dividends in high-dimensional problems.

2 4. Linear Methods for Classification

3 4.1 Introduction

The decision boundaries are linear.

Build **discriminant functions** $\delta_k(x)$ for each class k, and then classify x to the class with the largest value for its discriminant function. For example, model the posterior probabilities $\delta_k(x) = Pr(G = k|X = x)$.

If the discriminant functions $\delta_k(x)$ is linear in x, then the decision boundaries will be linear.

Actually, all we require is that some monotone transformation of $\delta_k(x)$ (or $Pr(G = k|X = x)$) be linear for the decision boundaries to be linear. For example, use logit transformation to get logistic regression: $\log[p/(1-p)] = X\beta$, where $p/(1-p)$ is odds and $\log[p(1-p)]$ is log-odds.

The decision boundary is the set of points for which a the log-odds are zero, and this is a hyperplane defined by $\{X|X\beta = 0\}$.

We discuss two very popular but different methods that result in linear log-odds (or logits): **linear discriminant analysis** and **linear logistic regression**. The essential difference between them is in the way the linear function is fit to the training data.

We will look at two methods that explicitly look for "separating hyperplanes". The first is the well-known perceptron model of Rosenblatt (1958), with an algorithm that finds a separating hyperplane in the training data, if exists. The second method, due to Vapnik (1996), finds an optimally separating hyperplane if exists, else finds a hyperplane that minimizes some measure of overlap in the training data.

3 4.3 Linear Discriminant Analysis

Bayes discriminant rule

Decision theory for classification (Section 2.4) tells us that we need to know the class posteriors $Pr(G|X)$ for optimal classification. Suppose $f_k(x)$ is the class-conditional density P of X in class G = k, and let π_k be the prior probability of class k, with $\sum_{k=1}^{K} \pi_k = 1$.

Many techniques are based on models for the class densities:

- inear and quadratic discriminant analysis use Gaussian densities;
- more flexible mixtures of Gaussians allow for nonlinear decision boundaries (Section 6.8);
- general nonparametric density estimates for each class density allow the most flexibility (Section 6.6.2);
- Naive Bayes models are a variant of the previous case, and assume that each of the class densities are products of marginal densities; that is, they assume that the inputs are conditionally independent in each class (Section 6.6.3).

For Gaussian example, suppose that we model each class density as multivariate Gaussian. If we assume different classes have the same variance, then we can get the linear discriminant functions (Page 109). If not same variance, we can get We then get quadratic discriminant functions (QDA) (Page 110).

The output of the quadratic discriminant (QDA) and LDA in quadratic polynomial space (e.g. $X_1, X_2, X_1X_2, X_1^2, X_2^2$) are similar. (Page 111 FIGURE 4.6)

Both LDA and QDA perform well on an amazingly large and diverse set of classification tasks.

Why LDA and QDA performs well?

A reason is that the data can only support simple decision boundaries such as linear or quadratic, and the estimates provided via the Gaussian models are stable. This is a bias variance trade-off—we can put up with the bias of a linear decision boundary because it can be estimated with much lower variance than more exotic alternatives. This argument is less believable for QDA, since it can have many parameters itself, although perhaps fewer than the non-parametric alternatives.

4 4.3.3 Reduced-Rank Linear Discriminant Analysis

Fisher linear discriminant rule

View informative low-dimensional projections of the data.

Find the linear combination $Z = a^T X$ such that the between-class variance is maximized relative to the within-class variance. Fisher's problem therefore amounts to maximizing the Rayleigh quotient.

3 4.4 Logistic Regression

4 4.4.5 Logistic Regression or LDA?

The logistic regression model is more robust and more general, in that it makes less assumptions.

LDA is not robust to gross outliers, because observations far from the decision boundary (which are down-weighted by logistic regression) play a role in estimating the common covariance matrix.

LDA assumes the distribution of X. By relying on the additional model assumptions, we have more information about the parameters, and hence can estimate them more efficiently (lower variance). Otherwise, it will pay a price for focusing on the (noisier) data.

3 4.5 Separating Hyperplanes

4 4.5.2 Optimal Separating Hyperplanes

The optimal separating hyperplane separates the two classes and maximizes the distance to the closest point from either class (Vapnik, 1996). Not only does this provide a unique solution to the separating hyperplane problem, but by maximizing the margin between the two classes on the training data, this leads to better classification performance on test data.

The optimal hyperplane focuses more on the points that count, and is more robust to model misspecification. The LDA solution, on the other hand, depends on all of the data, even points far away from the decision boundary. Note, however, that the identification of these support points required the use of all the data.

Of course, if the classes are really Gaussian, then LDA is optimal, and separating hyperplanes will pay a price for focusing on the (noisier) data at the boundaries of the classes.

2 8. Model Inference and Averaging

3 8.1 Introduction

For most of this book, the fitting (learning) of models has been achieved by **minimizing a sum of squares** for regression, or by **minimizing cross-entropy** for classification. In fact, both of these minimizations are instances of the maximum likelihood approach to fitting.

3 8.2 The Bootstrap and Maximum Likelihood Methods

Nonparametric bootstrap: sample with replacement from the training data.

Parametric bootstrap: in which we simulate new responses by adding Gaussian noise to the predicted values.

Bootstrap can access the accuracy (e.g. derive estimates of standard errors and confidence intervals) of a parameter estimate or a prediction.

3 8.7 Bagging

Bootstrap aggregation or bagging averages the prediction over a collection of bootstrap samples, thereby reducing its variance. Bagging can dramatically reduce the variance of unstable procedures like trees, leading to improved prediction. For regression tree, we simply fit the same regression tree many times to bootstrap-sampled versions of the training data, and average the result. For classification, a committee of trees each cast a vote for the predicted class.

For each bootstrap sample $\mathbb{Z}^{*b}$, $b = 1, 2, \ldots, B$, we fit our model, giving prediction $\hat{f}^{*b}(x)$. The bagging estimate is defined by

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

The bagged estimate $\hat{f}_{bag}(x)$ will differ from the original estimate $\hat{f}(x)$ only when the latter is a nonlinear or adaptive function of the data.

The probability of a observation is not be selected in the bootstrap sample is $(1-\frac{1}{n})^n \to \frac{1}{e}$. For each bootstrap sample, we have about 36.8% $(\frac{1}{e})$ of observations are not included in it. We can use these **out-of-bag samples** to compute the **out-of-bag error** (global estimate of the mean prediction error) of the model build on the bootstrap sample.

How to compute out-of-bag error?

For each x_i in training data set, get prediction $\hat{y}_{i(oob)}$ by the trees that were trained without x_i. Then the out-of-bag error is $\frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{y}_{i(oob)})$.

	$\mathbb{Z}^{*1} \to f^{*1}$	$\mathbb{Z}^{*2} \to f^{*2}$	$\mathbb{Z}^{*3} \to f^{*3}$	...	$\mathbb{Z}^{*B} \to f^{*B}$
(x_1, y_1)	✓	✗	✗	...	✓
(x_2, y_2)	✗	✗	✓	...	✓
...					
(x_N, y_N)	✓	✓	✗	...	✗

In the example above, (x_2, y_2) is not used in bootstrap samples $\mathbb{Z}^{*1}$ and $\mathbb{Z}^{*2}$ to grow the trees f^{*1} and f^{*2}, so $\hat{y}_{2(oob)} = average\left(\hat{f}^{*1}(x_2), \hat{f}^{*2}(x_2)\right)$.

2 9. Additive Models, Trees, and Related Methods

3 9.1 Generalized Additive Models

In the regression setting, a generalized additive model has the form

$$E(Y|X_1, X_2, \ldots, X_p) = \alpha + f_1(X_1) + f_2(X_2) + \ldots + f_p(X_p).$$

where the f_j's are unspecified smooth (“nonparametric”) functions.

We fit each function using a scatterplot smoother (e.g., a cubic smoothing spline or kernel smoother), and provide an algorithm for simultaneously estimating all p functions (Section 9.1.1). The estimated function $\hat{f}_j$ can then reveal possible nonlinearities in the effect of X_j .

In general, the conditional mean $\mu(X)$ of a response Y is related to an additive function of the predictors via a link function g:

$$g[\mu(X)] = \alpha + f_1(X_1) + \ldots + f_p(X_p).$$

For example, when $g(\mu) = \text{logit}(\mu)$, it is additive logistic regression model. This model replaces each linear term by a more general functional form: $\log(\frac{\mu(X)}{1-\mu(X)}) = \alpha + f_1(X_1) + \ldots + f_p(X_p)$.
compared to logistic regression model: $\log(\frac{\mu(X)}{1-\mu(X)}) = \alpha + \beta_1 X_1 + \cdots + \beta_p X_p$.

4 9.1.1 Fitting Additive Models

The additive model has the form: $Y = \alpha + \sum_{j=1}^{p} f_j(X_j) + \epsilon$. where the error term ε has mean zero.

The penalized sum of squares:

$$\mathrm{PRSS}(\alpha, f_1, f_2, \ldots, f_p) = \sum_{i=1}^{N} [y_i - \alpha - \sum_{j=1}^{p} f_j(x_{ij})]^2 + \sum_{j=1}^{p} \lambda_j \int f_j''(t_j)^2 d\,t_j. \quad (9.7)$$

where the $\lambda_j \geq 0$ are tuning parameters.

It can be shown that the minimizer of (9.7) is an additive cubic spline model.

3 9.2 Tree-Based Methods

4 9.2.1 Background

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one. They are conceptually simple yet powerful. We first describe a popular method for tree-based regression and classification called CART, and later contrast it with C4.5, a major competitor.

A key advantage of the recursive binary tree is its interpretability.

4 9.2.2 Regression Trees

Suppose first that we have a partition into M regions $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

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

If we adopt as our criterion minimization of the sum of squares $(y_i - f(x_i))^2$, it is easy to see that the best $\hat{c}_m$ is just the average of y_i in region R_m: $\hat{c}_m = \mathrm{avg}(y_i | x_i \in R_m)$.

5 9.2.2.1 How to build a regression tree?

Now finding the best binary partition in terms of minimum **sum of squares** is generally computationally infeasible. Hence we proceed with a **greedy algorithm**. Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j, s) = \{X | Xj \leq s\} \text{ and } R_2(j, s) = \{X | X_j > s\}.$$

Then we seek the splitting variable j and split point s that solve

$$\min_{j,s} \Big[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \Big].$$

For any choice j and s, the inner minimization is solved by

$$\hat{c}_1 = ave(y_i | x_i \in R_1(j, s)) \;\; and \;\; \hat{c}_2 = ave(y_i | x_i \in R_2(j, s)).$$

5 9.2.2.2 How to decide the tree size?

We first grow a large tree T_0, stopping the splitting process only when some minimum node size (say 5) is reached. Then this large tree is pruned using **cost-complexity pruning**.

We define a subtree $T \subset T_0$ to be any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal (non-terminal) nodes. We index terminal nodes by m, with node m representing region R_m. Let $|T|$ denote the number of terminal nodes in T. Letting

$$\begin{aligned} N_m &= \#\{x_i \in R_m\}, \\ \hat{c}_m &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i, \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2. \text{ (loss function)} \end{aligned}$$

Where $Q_m(T)$ is the node impurity, which is measured by squared loss. We define the cost complexity criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T| = \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2 + \alpha|T|.$$

The tuning parameter $\alpha \geq 0$ governs the trade-off between tree size and its goodness of fit to the data. The idea is to find, for each α, the subtree $T_\alpha \subseteq T_0$ to minimize $C_\alpha(T)$.

Large values of α result in smaller trees T_α, and conversely for smaller values of α. As the notation suggests, with α = 0 the solution is the full tree T_0.

For each α one can show that there is a unique smallest subtree T_α that minimizes $C_\alpha(T)$. For each α, to find T_α, we use **weakest link pruning**: we successively collapse the internal node that produces the smallest per-node increase in $\sum_m N_m Q_m(T)$, and continue until we produce the single-node (root) tree. This gives a (finite) sequence of subtrees, and one can show this sequence must contain T_α.

To choose α, we use five- or tenfold **cross-validation**: we choose the value $\hat{\alpha}$ to minimize the cross-validated sum of squares. Our final tree is $T_{\hat{\alpha}}$.

4 9.2.3 Classification Trees

The only difference between regression trees and classification trees are the methods of splitting nodes and pruning the tree.

In a node m, representing a region R_m with N_m observations. The proportion of class k observations in node m:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

We classify the observations in node m to the majority class in node m:

$$k(m) = \underset{k}{\operatorname{argmax}}\, I(y_i = k)$$

Different measures $Q_m(T)$ of node impurity include the following:

$$\text{Misclassification error: } 1-\hat{p}_{mk(m)} = \frac{1}{N_m}\sum_{x_i \in R_m} I(y_i \neq k(m))$$

$$\text{Gini index: } \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk})$$

$$\text{Cross entropy or deviance: } -\sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk})$$

All three are similar, but cross-entropy and the Gini index are differentiable, and hence more amenable to numerical optimization.

Cross-entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification rate. For example, in a two-class problem with 400 observations in each class (denote this by (400, 400)), suppose one split created nodes (300, 100) and (100, 300), while the other created nodes (200, 400) and (200, 0). Both splits produce a misclassification rate of 0.25, but the second split produces a pure node and is probably preferable. Both the Gini index and cross entropy are lower for the second split.

For this reason, either the Gini index or cross-entropy should be used when growing the tree. To guide cost-complexity pruning, any of the three measures can be used, but typically it is the misclassification rate.

Growing a tree: Gini index, cross-entropy.
Cost-complexity pruning: misclassification rate.

The Gini index can be interpreted in two interesting ways. (read the book for details)

We can treat the class proportions in the terminal node as the class probability estimates.

4 Some aspects of decision tree learning:

1. Multi-way splits?

- fragments the data very quickly;
- you can always realize a multi-way split by doing a series of binary splits.

2. Linear combinations of splits? E.x. $R_1 = \{X : 2X_1 + X_2 < 0\}, R_2 = \{X : 2X_1 + X_2 \geq 0\}$.

- maybe good for prediction;
- not good for interpretation;
- expensive for computation.

3. What types of functions does regression tree have trouble approximating?

- additive function $f(X) = \sum_{j=1}^{p} f_j(X_j)$. e.x. $f(X) = \sum_{j=1}^{p} \beta_j X_j$. (note that $Y = f(X) + \epsilon$)
- $f(X) = I\{X_1 \leq t_1\} + I\{X_2 \leq t_2\}$.

4. Categorical variables.

- order them according to the increase of largest classification probability $\max\{\hat{p}_{m1}, \hat{p}_{m2}, \ldots, \hat{p}_{mK}\}$ when we use this category to split data.

5. Ways to handle missing data

- remove observations containing missing values;
- add category "missing" and check whether "missing" has any predictive value;
- surrogate splits.

6. CART produce non-smooth predictions

- MARS (Multivariate adaptive regression spline)

2 10. Boosting and Additive Trees

Notations:

J : number of regions (nodes); j : index of regions (nodes); M : number of estimators (trees); m : index of estimators (trees).

3 10.1 Boosting Methods

The motivation for boosting was a procedure that combines the outputs of many “weak” classifiers to produce a powerful “committee”.

A weak classifier is one whose error rate is only slightly better than random guessing. The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers $G_m(x), m = 1, 2, \ldots, M$.

The predictions from all of them are then combined through a weighted majority vote to produce the final prediction:

$$G(x) = \operatorname{sign}\Big(\sum_{m=1}^{M} \alpha_M G_m(x)\Big).$$

Here $\alpha_1, \alpha_2, \ldots, \alpha_M$ are computed by the boosting algorithm, and weight the contribution of each respective $G_m(x)$. Their effect is to give higher influence to the more accurate classifiers in the sequence. The data modifications at each boosting step consist of applying weights $w_1, w_2, \ldots, w_N$ to each of the training observations $(x_i, y_i), i = 1, \ldots, N$.

Algorithm 10.1 AdaBoost.M1.

1. Initialize the observation weights $w_i = 1/N, i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:

   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.

   (b) Compute

$$err_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$

(c) Compute

$$\alpha_m = \log(\frac{1 - err_m}{err_m}).$$

(d) Set

$$w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$$

3. Output $G(x) = \text{sign}\Big(\sum_{m=1}^M \alpha_M G_m(x)\Big)$.

XGBoost uses variable selection frequency for variable importance.

3 10.2 Boosting Fits an Additive Model

Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions. Basis function expansions take the form

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

where $\beta_m, m = 1, 2, \ldots, M$ are the expansion coefficients, and $b(x; \gamma) \in \mathbb{R}$ are usually simple functions of the multivariate argument x, characterized by a set of parameters γ. We discuss basis expansions in some detail in Chapter 5.

In the algorithm 10.1 AdaBoost.M1. The basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$.

Algorithm 10.2 Forward Stagewise Additive Modeling.

```
1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to $M$:
    (a) Compute
        $$(\beta_m, \gamma_m) = \underset{\beta,\gamma}{\text{argmin}} \sum_{i=1}^N L\Big(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)\Big).$$
    (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.
3. Output $f_M(x)$.
```

In m^{th} step, we need to do optimization with $1 + p$ parameters (β_m is 1 dimension and γ_m is p). At the end, $f_M(x)$ have $M(1 + p)$ parameters. That is much easier than we do optimization with $M(1 + p)$ parameters: $(\beta, \gamma) = \text{argmin}_{\beta,\gamma} \sum_{i=1}^N L\Big(y_i, f_M(x_i)\Big)$.

The philosophy is like Taylor expansion.

10.9 Boosting Trees

Tree ensemble methods (boosting: e.g. GBM, XGBoost, bagging: e.g. random forest) are widely used. Almost half of data mining competition are won by using some variants of tree ensemble methods. We talk about boosting trees in this chapter, and random forest in chapter 15.

As for CART, a constant γ_j is assigned to each region R_j and the predictive rule is $x \in R_j \Rightarrow f(x) = \gamma_j$.

Thus a tree can be expressed as the additive model form:

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

with parameters $\Theta = \{R_j, \gamma_j\}_1^J$. J is usually treated as a meta-parameter. The parameters are found by minimizing the empirical risk

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j),$$

It is useful to divide the optimization problem into two parts:

1. Finding γ_j given R_j: Given the R_j, often $\hat{\gamma}_j = \bar{y}_j$, the mean of the y_i falling in region R_j.
2. Finding R_j: A typical strategy is to use a greedy, top-down recursive partitioning algorithm to find the R_j.

In Section 9.2 we described such a strategy for classification trees. The Gini index replaced misclassification loss in the growing of the tree (identifying the R_j).

The **boosted tree** model is a sum of such trees,

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

induced in a forward stagewise manner (Algorithm 10.2 Forward Stagewise Additive Modeling.).

10.4 Exponential Loss and AdaBoost

We now show that AdaBoost.M1 (Algorithm 10.1) is equivalent to forward stagewise additive modeling (Algorithm 10.2) using the exponential loss function $L(y, f(x)) = \exp(-yf(x))$.

10.5 Why Exponential Loss?

Denote $\text{margin}_i := y_i f(x_i)$ for classification and $\text{margin}_i := y_i - f(x_i)$ for regression. Exponential loss is suitable for classification but not for regression. For classification, observations with positive margin are correctly classified and negative are incorrectly. Any loss criterion for classification should penalize negative margins more heavily than positive ones. Exponential loss pay more attention to misclassified observations, so it is not robust.

3 10.10 Numerical Optimization via Gradient Boosting

4 10.10.2 Gradient Boosting

Gradient boosting is base on gradient descent. We induce a weak regressor or classifier $h_m(x)$ (e.g. a tree $T(x;\Theta_m)$) at the m^{th} iteration to fit the negative gradient, which means the predictions (a vector with dimension $n \times 1$) by weak model $h_m(x)$ are as close as possible to the negative gradient.

We want to minimize $Loss = L(y_i, f(x_i)), i = 1, \dots N$, where f is the independent variable. According to gradient descent, at m^{th} step, $Loss_m = Loss_{m-1} - s_m \cdot \nabla Loss_m$, where s_m is the step size and gradient

$$\nabla Loss_m = \frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$$

If $L(y_i, f_{m-1}(x_i)) = \frac{1}{2}[y_i - f_{m-1}(x_i)]^2$, then the gradient $\frac{\partial L(y_i, f(x_i))}{\partial f_{m-1}(x_i)} = f_{m-1}(x_i) - y_i$, which is the residual r_{im}.

Recall Taylor expansion: $f(x + \Delta x) \approx f(x) + f'(x)\Delta x$, we want to minimize the loss function value

$$\sum_{i=1}^{N} L\Big(y_i, f_{m-1}(x_i) + h_m(x_i)\Big) \approx \sum_{i=1}^{N} \Big[L(y_i, f_{m-1}(x_i)) + \frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)} h_m(x)\Big]$$

If $L(y_i, f_{m-1}(x_i)) = \frac{1}{2}[y_i - f_{m-1}(x_i)]^2$, we have

$$\sum_{i=1}^{N} L\Big(y_i, f_{m-1}(x_i) + h_m(x_i)\Big) \approx \sum_{i=1}^{N} \Big[L(y_i, f_{m-1}(x_i)) - [\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}]^2\Big]$$

As we can see, the loss function value decreases in every step. However, if we use second derivative to approach the loss function, the loss function value will decrease faster, which is the idea of XGBoost and it is similar to Newton's method.

In practice, the GBM works very well when the number of terminal nodes is between 4 and 8, and very well when the step size (learning rate) is smaller than 0.1.

4 10.10.3 Implementations of Gradient Boosting

Algorithm 10.3 presents the generic gradient tree-boosting algorithm for regression. Specific algorithms are obtained by inserting different loss criteria $L(y, f(x))$. The first line of the algorithm initializes to the optimal constant model, which is just a single terminal node tree. The components of the negative gradient computed at line 2(a) are referred to as generalized or pseudo residuals, r.

Algorithm 10.3 Gradient Tree Boosting Algorithm.

```
1. Initialize $f_0(x) = \text{argmin}_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
    (a) For $i = 1, 2, \ldots, N$ compute
            $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$.
    (b) Fit a regression tree to the targets $r_{im}$'s to get terminal regions $R_{jm}, j = 1, 2, .., J_m$.
    (c) For $j = 1, 2, \ldots, J_m$, compute $\gamma_{jm}$. Now we get a new tree
            $h_m(x) = T(x; \Theta_m) = \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
    (d) Compute step size $s_m = \text{argmin}_s \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + s \cdot h_m(x_i))$ .
    (e) Update
            $f_m(x) = f_{m-1}(x) + s_m \cdot h_m(x)$,
        where $h_m(x) = \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ is a new tree ($m^{th}$ tree).
3. Output $\hat{f}(x) = f_M(x)$.
```

---

3 XGBoost

At m^{th} step, we want to minimize the objective

$$Obj^{(m)} = \sum_{i=1}^n L(y_i, f_{m-1}(x) + h_m(x_i)) + \Omega(f_{m-1} + h_m),$$

then $h_m = \text{argmin}_{h_m} Obj^{(m)}$.

In the formula above, $\Omega(f_m)$ is the **regularization** term, which is used to measure the complexity of f_m.

$$\Omega(f_m) = \mu T + \frac{1}{2}\lambda \sum_{j=1}^T w_j^2,$$

where T is the number of leaves, w_j is the prediction score of leaf j, and $\sum_{j=1}^T w_j^2$ is the L2 norm of leaf scores.

Why penalize $\sum_{j=1}^T w_j^2$ to control the complexity? In this case, a large value of w_i would correspond to a terminal (leaf) node giving a very large and significant update to the prior model. However, the idea of a gradient booster is to carefully and slowly reduce the bias of the model by adding these trees one by one. (source)

We use **second order Taylor expansion** to get the approximation of loss function. Recall Taylor expansion: $f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$. We do that for $L(y_i, f_{m-1}(x) + h_m(x_i))$, and we have

$$\begin{aligned} Obj^{(m)} &= \sum_{i=1}^{n} L(y_i, f_{m-1}(x) + h_m(x_i)) + \Omega(f_{m-1}) + \Omega(h_m) \\ &\approx \sum_{i=1}^{n} \left[L(y_i, f_{m-1}(x)) + g_i h_m(x_i) + \frac{1}{2} h_i h_m^2(x_i) \right] + \Omega(f_{m-1}) + \Omega(h_m), \end{aligned}$$

where $g_i = \frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}, h_i = \frac{\partial L^2(y_i, f_{m-1}(x_i))}{\partial f_{m-1}^2(x_i)}$.

Here is a brief description of XGBoost algorithm (read references for computation details):

For $m = 1, \ldots, M$:

1. $h_m = \underset{h_m}{\text{argmin}}\, Obj^{(m)} = \underset{h_m}{\text{argmin}} \left[g_i h_m(x_i) + \frac{1}{2} h_i h_m^2(x_i) \right] + \Omega(h_m).$
2. $f_m(x) = f_{m-1}(x) + s_m \cdot h_m(x),$ where s_m is called step-size or shrinkage, usually set around 0.1.

The **shrinkage** means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting.

Regularization, second order Taylor expansion, and shrinkage are the key reasons why XGBoost is better than GBM.

References: Introduction to Boosted Trees by Tianqi Chen; GBDT and XGBoost by 张凌寒.

3 10.13 Interpretation

4 10.13.1 Relative Importance of Predictor Variables

For a single decision tree T_m, a measure of importance for variable X_j is $\mathcal{I}_l^2(T_m)$, which is the sum of the improvements in squared error after partition regions by variable X_j over all nodes.

For additive tree expansions (10.28), it is simply averaged over the trees $\mathcal{I}_l^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_l^2(T_m)$.

4 10.13.2 Partial Dependence Plots

The goal is to produce a visual description of the effect of X_S on f via a plot of $\hat{f}(X_S)$. Let $\mathcal{S} \subset \{1, 2, \ldots, p\}$. Let $\mathcal{C}$ be the complement set, with $\mathcal{S} \cup \mathcal{C} = \{1, 2, \ldots, p\}$. A general function $f(X)$ will in principle depend on all of the input variables: $f(X) = f(X_\mathcal{S}, X_\mathcal{C})$.

The average or partial dependence of $f(X)$ on $X_\mathcal{S}$ can be defined as $f_\mathcal{S}(X_\mathcal{S}) = E_{X_\mathcal{C}} f(X_\mathcal{S}, X_\mathcal{C})$, and can be estimated by

$$\bar{f}_\mathcal{S}(X_\mathcal{S}) = \frac{1}{N} \sum_{i=1}^{N} f(X_\mathcal{S}, x_{i\mathcal{C}}),$$

where $\{x_{1\mathcal{C}}, \ldots, x_{N\mathcal{C}}\}$ are the values of $X_\mathcal{C}$ occurring in the training data.

Partial dependence functions $f_\mathcal{S}(X_\mathcal{S})$ can be used to interpret the results of any "black box" learning method. However, it can be computationally intensive. Fortunately with decision trees, $\bar{f}_\mathcal{S}(X_\mathcal{S})$ can be rapidly computed from the tree itself without reference to the data (Exercise 10.11).

The partial dependence of $f(X)$ on $X_{\mathcal{S}}$ is a marginal average of f, and can represent the effect of $X_{\mathcal{S}}$ on $f(X)$ after accounting for the (average) effects of the other variables $X_{\mathcal{C}}$ on $f(X)$. It is not the effect of $X_{\mathcal{S}}$ on $f(X)$ ignoring the effects of $X_{\mathcal{C}}$. The latter is given by the conditional expectation $\tilde{f}_{\mathcal{S}}(X_{\mathcal{S}}) = E(f(X_{\mathcal{S}}, X_{\mathcal{C}})|f(X_{\mathcal{S}})$.

2 11. Neural Networks

3 11.2 Projection Pursuit Regression

Kolmogorov Arnold Representation theorem states that any continuous function f can be exactly represented by (possible infinite) sequence of addition, multiplication and composition with functions that are universal (do not depend on f).

The projection pursuit regression (PPR) model

$$f(x_i) = \sum_{m=1}^{M} g_m(w_m^T x_i)$$

This is a linear additive model, but in the "derived" features $w_m^T x_i$, rather than the input variables.

The function g_m called ridge function in $\mathbb{R}^p$ and is a non-linear function. The $p \times 1$ vector w_m is called the inner-layer or hidden layer weights vector. The scalar M is is called the width (the number of hidden units). The scalar m is the index of hidden unit. The scalar variable $w_m^T x_i$ is the projection of x_i onto the unit vector w_m, and we seek w_m so that the model fits well, hence the name "projection pursuit".

If M is taken arbitrarily large, for appropriate choices of g_m, f can approximate any continuous function to any desired level of accuracy. Note: when $M = 1$, i.e. $f_1(X) = g_1(w_m^T X)$, which is called single hidden layer.

For classification, we can use log loss function. For regression, we can use squared error loss function $\sum_{i=1}^{N} L(y_i, \hat{y}_i) = \sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} g_m(w_m^T x_i)\right]^2$.

We seek the approximate minimizers g_m's and w_m's of the error function. Given g_m's, we want to minimize over w_m's. A Gauss-Newton search is convenient for this task. This is a quasi-Newton method.

3 11.3 Neural Networks

Activation function $\sigma(w_m^T x_i + b)$ is usually a tanh function, where b is called bias. Nowadays, the most popular activation is called the rectified linear unit (ReLU) $g(z) = \max(0, z)$.

In K-class classification, we use softmax function:

$$\sigma(\mathbf{z})_l = \frac{e^{z_l}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } l = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K.$$

Gradient descent for minimizing $\hat{R}$ (neural networks) is known as back-propagation.

2 15. Random Forests

3 15.1 Introduction

Bagging or **bootstrap aggregation** (section 8.7) is a technique for reducing the variance of an estimated prediction function. Bagging seems to work especially well for high-variance, low-bias procedures, such as trees. **Boosting** in Chapter 10 was initially proposed as a committee method as well, although unlike bagging, the committee of weak learners evolves over time, and the members cast a weighted vote. Boosting appears to dominate bagging on most problems, and became the preferred choice.

Random forests (Breiman, 2001) is a substantial modification of bagging that builds a large collection of de-correlated trees, and then averages them. On many problems the performance of random forests is very similar to boosting, and they are simpler to train and tune. As a consequence, random forests are popular,

3 15.2 Definition of Random Forests

The essential idea in bagging (Section 8.7) is to average many noisy but approximately unbiased models, and hence reduce the variance. Trees are ideal candidates for bagging, since they can capture complex interaction structures in the data, and if grown sufficiently deep, have relatively low bias. Since trees are notoriously noisy (unstable), they benefit greatly from the averaging.

Algorithm 15.1 Random Forest for Regression or Classification.

```
1. For $b = 1$ to $B$:
   (a) Draw a bootstrap sample $Z^*$ of size $N$ from the training data.
   (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps
   for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
       i. Select $m$ variables at random from the $p$ variables.
       ii. Pick the best variable/split-point among the $m$.
       iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

Moreover, since each tree generated in bagging is identically distributed (i.d.), the expectation of an average of B such trees is the same as the expectation of any one of them. This means the bias of bagged trees is the same as that of the individual trees. This is in contrast to boosting, where the trees are grown in an adaptive way to remove bias, and hence are not i.d.

An average of B i.i.d. random variables, each with variance σ^2 , has variance σ^2/B. If the variables are simply i.d. (identically distributed, but not necessarily independent) with positive pairwise correlation ρ, the variance of the average is (Exercise 15.1):

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

As B increases, $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \to \rho\sigma^2$, and hence the correlation ρ of pairs of bagged trees limits the benefits of averaging.

The idea in random forests (Algorithm 15.1) is to improve the variance reduction of bagging by reducing the correlation ρ between the trees, without increasing the variance σ^2 too much. This is achieved in the tree-growing process through **random selection of the input variables**.

As for the terminal node size, we set the minimum number of observations in each terminal node as 5 for regression and 1 for classification.

3 15.3 Details of Random Forests

For classification, the default value for m is $\lfloor \sqrt{p} \rfloor$ and the minimum node size is one.
For regression, the default value for m is $\lfloor p/3 \rfloor$ and the minimum node size is five.

4 15.3.1 Out of Bag Samples

An important feature of random forests is its use of out-of-bag (oob) samples. An oob error estimate is almost identical to that obtained by N-fold cross-validation. Once the oob error stabilizes, the training can be terminated.

4 15.3.2 Variable Importance

There are two popular metrics to measure variable importance.

1. Mean Decrease in Impurity (MDI).

   At each split in each tree, the improvement in the split-criterion is the importance measure attributed to the splitting variable, and is accumulated over all the trees in the forest separately for each variable.

   $$M\hat{D}I(x_j) = \frac{1}{B}\sum_{b=1}^{B}\sum_{\text{nonterminal nodes}}(\text{Decrease in Impurity from spliting the j-th variable}).$$

2. Mean Decrease in Accuracy (MDA).

   After we computed the OOB error, we randomly permute the values for the j^{th} variable ($j = 1, \ldots, p$), and then compute the OOB error again. (see *8.7 Bagging* for details of OOB error)

   $M\hat{D}A(x_j) =$ Difference in the OOB error before and after (randomly) permuting the values of j^{th} variable x_j in the out-of-bag samples.

   This metric is more computationally expensive and accurate.

4 15.3.4 Random Forests and Overfitting

Random Forests is **robust**. When the number of relevant variables increases, the performance of random forests is surprisingly robust to an increase in the number of noise variables. For example, with 6 relevant and 100 noise variables, the probability p of a relevant variable being selected at any split is 0.46.

Random forests “cannot overfit” the data. It is certainly true that increasing B does not cause the random forest sequence to overfit.

2 Other Topics

3 Variables Selection

Usually we need $N > e^p$.

4 Permutation on Variables

Permute the values of a variable and fit the model with it and all other variables, then record the test error of this model. Permutation on the important variable leads to a large increase of test error.

2 References

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. No. 10. New York: Springer series in statistics, 2001.

Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016.