Statistical Learning, Fall 2024-5

Class notes 9

Bagging and Random Forest

Instead of a single tree being a model, combine many trees into a model:

1. Bagging and Random Forest: Fit different trees to the data and average them
2. Boosting: Adaptively build a model from adding more and more trees

We will focus first on Random Forest (also Bagging), later discuss boosting. Main idea of Random Forest: Take advantage of the instability and high variance of the trees, which are unstable and greedy: if we change the data a little bit, the tree can change a lot. Now we intentionally change (randomize) the data to get a different tree every time, and average them.

The value of averaging: This is captured through different things we know: CLT, LLN, variance of the average... Assume $z_i \sim F$ has some distribution with mean μ and variance σ^2. If $z_1, ..., z_m \sim F$ are independent, then $Var(\bar{z}) = \sigma^2/m$, so $\bar{z}$ is close to μ for large m.

Slightly more complex setting: assume $z_1, ..., z_m$ are *somewhat* dependent $Cov(z_i, z_j) = \rho\sigma^2,\ \rho < 1$. Now we still get some variance reduction from averaging:

$$Var(\bar{z}) = \rho\sigma^2 + (1-\rho)\sigma^2/m \to \rho\sigma^2.$$

This is exactly the intuition behind Bagging and Random forest.

Random forest algorithm:

1. Repeat many times:
   (a) Randomize the data (by taking a subsample or a *bootstrap* sample)
   (b) Build a tree on the randomized data, also randomize tree building, by randomly choosing variables to consider at each node: typically choosing $k \approx p/3$ variables in regression and $k \approx \sqrt{p}$ for classification.
2. Final model is average of all models: To predict at new x_0, apply each tree and average their predictions

Intuition: tree predictions are different because of randomization, they are like $z_1, ...z_n \sim \mathbb{P}(\hat{y}_0|x_0, T)$

1. Related ($\rho > 0$) because it's the same training set T

2. Still different from each other ($\rho < 1$) because of randomization and instability of trees

Hence we expect (and indeed see!) that Random Forest gives more accurate predictions of $E(y|x)$ or $P(y = 1|x)$ than single trees.

We take care of the bias by building very big trees, assuring that they are roughly unbiased (flexible, have low appx. error) but high variance, and we take care of the variance with the Random Forest averaging.

Bagging is the same as RF, applying only data randomization, without the randomization of variables in each split. Historically, Breiman first invented Bagging around 1994, and extended it to RF in 2001.

Boosted trees

Intuitive idea: We gradually and iteratively build the overall model as a sum of smaller models called **weak learners**. Each weak learner seeks to improve the model we have so far. Weak learners can be any predictive model, most widely used: trees. How do we capture the notion of *improve the model we have so far?*.

Boosting: overall scheme

1. Initialize $F^{(0)}(x) = 0,\ \forall x$

2. At stage $t \geq 1$:

   (a) Calculate $Y^{(t)} = (y_1^{(t)}, \ldots, y_n^{(t)})$ capturing what the model $F^{(t-1)}$ *has not yet explained*

   (b) Fit a weak learner $\hat{f}^{(t)}$ to $T^{(t)} = (X, Y^{(t)})$

   (c) Update $F^{(t)} = F^{(t-1)} + \epsilon \hat{f}^{(t)}$

Details: How to determine $Y^{(t)}$? Which weak learner to use? What is ϵ?

Example: Tree boosting for regression
Defining $Y^{(t)}$ as $y_i^{(t)} = (y_i - F^{(t-1)}(x_i))$ the current residual (what the model does not explain)
Weak learner: trees, usually small — two- or three-level trees as $\hat{f}^{(t)}$
Make ϵ as small as possible (ϵ-boosting): tradeoff between accuracy and computation.

For regression, taking the residual as $y_i^{(t)}$ makes sense. What is an analogy for classification? What about a more rigorous mathematical explanation of what we are doing? There are several approaches of varying mathematical complexity for describing and analyzing boosting.

The additive model view

Start with a very large (possibly infinite) set of q candidate "weak learners": $h_1(x), ..., h_q(x)$. We are looking for a "linear" model of the form $\hat{f}(x) = \sum_{k=1}^{q} \hat{\beta}_k h_k(x)$. In boosted trees example the

h_k's are *all possible trees of the given depth.* Since q is huge we cannot directly find a good $\hat{\beta} \in \mathbb{R}^q$.

Additive model via boosting:
At each iteration t we find a "good" candidate h_{k_t} and add ϵh_{k_t} to the current model. After T iterations we have a model where $\hat{\beta}_k = \epsilon \times \#\{k_t = k\}$ (the number of times k was chosen). How do we define a good h_{k_t} to update its coefficient? One option: given the current model $F^{(t-1)}$, which h_k improves the model fit the *fastest* when we add it to the model? This can be captured by using the derivative of the loss which measures the fit. Derivative of the RSS (squared loss):

$$\left.\frac{\partial RSS(F^{(t-1)})}{\partial \hat{y}_i}\right|_{\hat{y}_i = F^{(t-1)}(x_i)} = -2(y_i - F^{(t-1)}(x_i)).$$

The gradient boosting paradigm

- Choose a loss function for modeling (like RSS for regression)
- At each iteration: calculate the (negative) gradient of the loss function at the current model, use that as $Y^{(t)}$ for the next weak learner
- Interpretation: trying to find a weak learner h_{k_t} which "behaves like" the negative gradient, which is the direction of fastest decrease of the loss
- Can be applied with different loss functions for regression or classification

For 2-class classification we denote $y_i \in \{0, 1\}$ (sometimes $y_i \in \{\pm 1\}$ but we stick here with 0/1). A common loss function, which we also presented in the context of logistic regression is the (negative) Bernoulli log likelihood:

$$L(y_i, \hat{y}_i) = -y_i \log(\frac{\exp(\hat{y}_i)}{1 + \exp(\hat{y}_i)}) - (1 - y_i) \log(\frac{1}{1 + \exp(\hat{y}_i)}).$$

For simplicity denote as in logistic regression using the inverse logit transformation:

$$\hat{p}_i = \frac{\exp(\hat{y}_i)}{1 + \exp(\hat{y}_i)} = \frac{\exp(F^{(t-1)}(x_i))}{1 + \exp(F^{(t-1)}(x_i))},$$

where the last equality refers to already applying $\hat{y}_i = F^{(t-1)}(x_i)$.

If we use this loss function in a gradient boosting algorithm, after some scary differentiation we get that simply:

$$y_i^{(t)} = y_i(1 - \hat{p}_i) - (1 - y_i)\hat{p}_i.$$

Note that since $y_i \in \{0, 1\}$, only one of the two expressions is non-zero.

Formal gradient boosting description

In the gradient boosting paradigm, define:

- Training loss function per observation: $L(y, \hat{y})$, and total: $\mathcal{L} = \sum_i L(Y_i, \hat{Y}_i)$.
- Family of q weak learners: $h_k : \mathcal{X} \to \mathbb{R}$, $k = 1, \ldots, q$, possibly $q = \infty$.

Now define $F^{(0)} \equiv 0$, and for $t = 1, \ldots, T$:

1. Set $\nabla \mathcal{L} = \left\{ \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}\Big|_{\hat{y}_i = F^{(t-1)}(x_i)} \right\}_{i=1}^{n}$
2. Solve (exactly or approximately) $k_t = \arg\min_k \langle \nabla \mathcal{L}, h_k(\mathbb{X}) \rangle$
3. Find coefficient $\alpha_t = \begin{cases} \arg\min_\alpha \mathcal{L}(F^{(t-1)} + \alpha h_k) & \text{Line search boosting} \\ \epsilon \text{ (small)} & \epsilon\text{-boosting} \end{cases}$
4. Update $F^{(t)} = F^{(t-1)} + \alpha_t h_{k_t}$.

The optimization problem in the second step is often replaced with:

$$k_t = \arg\min_k \|(-\nabla \mathcal{L}) - h_k(\mathbb{X})\|^2$$

which is very similar, except it also penalizes $\|h_k(\mathbb{X})\|^2$, controlling the norm of the model.

AdaBoost as gradient boosting

The famous AdaBoost algorithm (Freund and Schapire 1992) was developed in the machine learning community, with a different theory, but we can discuss it as gradient boosting. The algorithm for two-class classification initializes $\hat{f}_0 = 0, w_i \equiv 1$, then for $t = 1...T$ updates:

1. Fit a classification tree with response y and weights w on the observations, getting tree h_t
2. Denote by Err_t the (weighted) misclassification error of h_t
3. Set $\alpha_t = 0.5 \log\left((1 - Err_t)/Err_t\right)$
4. Update weights: $w_i \leftarrow w_i \exp\left(-\alpha_t (y_i h_t(x_i))\right)$

This is a gradient boosting algorithm with $L(y, \hat{y}) = \exp(-y\hat{y})$, (where $y \in \{\pm 1\}$ and $\hat{y} \in \mathbb{R}$). The weights w are the gradient, and step 3 is the solution to line search in this setting.