Ensembles

Muchang Bahng

Spring 2025

Contents

The bias variance noise decomposition gives us a very nice way of explaining overfitttting. That is, the bias (expectation of the squared difference between the true $\mathbb{E}[Y \mid X]$ and the expected trained hypothesis function $h_{\boldsymbol{\theta};\mathcal{D}}$) reduces, but the variance in this overfitted model increases. Therefore, if we had a slightly different dataset $\mathcal{D}$ sampled from $(X \times Y)^N$, then we might have a very different trained hypothesis since it's so sensitive to the data.

A way to treat this is through **ensemble learning**, where we train *multiple* models over slightly different datasets, and then average their predictions to get a better model. What do we mean by a better model? Well, we know that a too complex model has low bias but large variance, and a too simple model has high bias but low variance.

1. *Bagging* refers to taking a complex model and decreasing its variance. Even though each model is trained over a smaller dataset, resulting it being more noisy, the average of all these slightly more noisy models will hopefully bring down the variance more than what we have added.[1]
2. *Boosting* refers to taking a simple model and decreasing its bias. Each simple model, usually a weak learner, has relatively small search space, but by taking the aggregate of them, we can hopefully increase it whilst bounding the variance in some way. Usually, the dataset is reweighted such that the weak learner in the next iteration will correct the previous weak learner.

[1] This is why random forests have underlying trees that are somewhat as large as possible.

1 Ensemble Models

2 The Jackknife

The Jackknife is a resampling technique first introduced by Quenouille in 1949 [Que49].

3 Bootstrapping

4 Boosting

Now we delve more into the applied and computational aspects of machine learning. It's had a lot of empirical success and is more interesting from a theoretical perspective. It starts off with the *weak learning assumption*, which we introduce in the context of classification with the misclassification loss function. It is actually a specific case of PAC learners.

Definition 4.1 (Probability Approximately Correct Learner)

A **PAC learning** is an algorithm that learns a function class $\mathcal{H}$ with parameter $\delta > 0$ if there exists an $\epsilon > 0$ and the algorithm can find a $f \in \mathcal{H}$ with probability at least $1 - \delta$ s.t.

$$R(f) \leq \epsilon \tag{1}$$

i.e.

$$\mathbb{P}[R(f) \leq \epsilon] \geq 1 - \delta \tag{2}$$

This is quite a strong requirement, since it says that with probability at least $1 - \delta$ you must find an model f that is correct with a probability of $1 - \epsilon$, i.e. ϵ-good.

Definition 4.2 (Weak Learner)

A **weak learner** is an algorithm that learns a function class $\mathcal{H}$ with parameter $\delta > 0$ if there exists an $\gamma > 0$ and the algorithm can find a $f \in \mathcal{H}$ s.t.

$$\mathbb{P}[R(f) < 1/2 - \gamma] \geq 1 - \delta \tag{3}$$

for some $\delta > 0$, where γ is considered our edge. Another way to write it is that with probability of at least $1 - \delta$, we can find a function f s.t.

$$\mathbb{P}_{x,y \sim \mathcal{X} \times \mathcal{Y}}[f(x) \neq y] < 1/2 - \gamma \tag{4}$$

This essentially means that given some γ that measures how good our target predictor is compared to random guessing, the probability that we can find such a predictor with this edge is $1-\delta$. Furthermore, this case must hold true for all distributions $P \sim \mathcal{X} \times \mathcal{Y}$.

Therefore, a weak learner just means some algorithm that learns a model that is a bit better than random. For example, learning decision stumps may be a weak learner. Colloquially, a weak learner can be thought of as an algorithm that cannot get your training error to 0 and a strong learner can. The question is then, can we make a strong learner out of a bunch of weak learners? The general idea is that we want to iteratively find a bunch of weak learners and slowly add them up to get a strong learner.

$$f = \sum_{i=1}^{n} f_i \tag{5}$$

where f is strong, f_i weak.

4.1 Adaptive Boosting (AdaBoost)

Let's start with Adaboost for binary classification.

Definition 4.3 (Adaboost for Binary Classification)

Given data $\{(x_i, y_i)\} \in \mathcal{X} \times \mathcal{Y}$, with $\mathcal{Y} = \{-1, +1\}$, we implement the following greedy algorithm.

1. You first set an n-vector weighting your samples, where the weight of the ith sample is $W_t(i)$.

$$W_1 = (\frac{1}{n}, \dots, \frac{1}{n}) \tag{6}$$

2. For $t = 1, \dots, T$, we do the following.
   (a) You run your weak learning algorithm, which will return your hypothesis h_t with probability $1 - \delta$ which is slightly better than random. We define its empirical error over the distribution W_t to be

$$\epsilon_t = R_{W_t}(h_t) = \mathbb{P}_{x_i \sim W_t}[h_t(x_i) \neq y_i] = \sum_{i=1}^{n} W_t(i) \cdot \mathbb{1}_{h_t(x_i) \neq y_i} \tag{7}$$

   This may be done differently by actually sampling n samples from this distribution and then computing proportion of misclassifications.
   (b) This new weak learner provides some information on the new weighted distribution. We would like to weight this weak learner h_t with some scale α_t to determine how important its vote is in the ensemble. We define this weighting to be

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{8}$$

   Note the following important properties. If $0 < \epsilon_t < 0.5$, then this does indeed mean that h_t is slightly better than random, so it would have a positive weighting. If $\epsilon_t = 0.5$, then it is random so no weighting. Finally, if $0.5 < \epsilon < 1.0$, then it is an extremely poor classifier and we are better off looking at the opposite of its prediction, meaning that α_t will be negative. This is also seen with the facts that as $\epsilon_t \to 0, 1$, then $\alpha_t \to +\infty, -\infty$.[a]
   (c) Then we set

$$W_{t+1}(i) \propto W_t(i) \exp\{-\alpha_t y_i h_t(x_i)\} = \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{+\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \tag{9}$$

   meaning that the new weights go up for incorrect labels and down for correct labels. We show proportional to since it is not normalized, but we can normalize it with the constant Z_t.
3. Your final strong classifier is then

$$f(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right) \tag{10}$$

   which indeed is a sequential sum of these classifiers.

[a] In practice, ϵ cannot be 0 or 1 due to numerical reasons, so a small constant is added to prevent this from happening.

In this way, by weighting the incorrect labels higher, I am telling successive weak learner to give me a new weak hypothesis that tells me something new. This makes it so that the actual sequence of learned weak models are important, since the next h_{t+1} tries to fix the errors that the h_t makes.

Algorithm 4.1 (AdaBoost Algorithm)

The full algorithm for brevity is shown below.

Require: Training data $\{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$
Require: Number of iterations T
Require: Weak learning algorithm $\mathcal{A}$
Initialize weights $W_1(i) = \frac{1}{n}$ for $i = 1, \dots, n$
for $t = 1$ to T **do**
    Train weak learner $h_t = \mathcal{A}(\{(x_i, y_i)\}, W_t)$
    Calculate weighted error:
    $\epsilon_t = \sum_{i=1}^n W_t(i) \cdot \mathbb{1}_{h_t(x_i) \neq y_i}$
    if $\epsilon_t \geq \frac{1}{2}$ **then**
        break
    end if
    Calculate importance weight:
    $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
    Update sample weights:
    for $i = 1$ to n **do**
        $W_{t+1}(i) = W_t(i) \cdot \exp(-\alpha_t y_i h_t(x_i))$
    end for
    Normalize weights:
    $Z_t = \sum_{i=1}^n W_{t+1}(i)$
    $W_{t+1}(i) = \frac{W_{t+1}(i)}{Z_t}$ for all i
end for
return Final classifier $f(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$

We now actually show that this is a strong learner by showing that the training error goes to 0.

Theorem 4.1 (Exponential Decay of Training Error in AdaBoost)

Support that $\gamma \leq (1/2) - \epsilon_t$ for all t. Then our empirical risk decays exponentially with T.

$$\hat{R}(h) \leq e^{-2\gamma^2 T} \tag{11}$$

and hence, the training error goes to 0 quickly.

Proof.

Can be shown with the lemma.

$$Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)} \tag{12}$$

Sure, the training error goes to 0, but what we really care about is the generalization error. It turns out that we can prove things about this, but omitted for now.

Example 4.1 (AdaBoost with Stumps)

We can define our weak learning algorithm to be a decision stump with only 1 split. Doing adaboost gives something similar to a random forest (but not quite since its a bagging algorithm) with great generalization error.

Surprisingly, Adaboost has a tendency not to overfit, i.e. the variance does not explode. There is a lot of theory that tries to explain why this is the case, such as margin theory.

There are a lot of different ways to analyze AdaBoost. For many years, researchers did not think of it as having any connection to gradient descent or loss functions, but it actually does. AdaBoost can be viewed

as optimizing the **exponential loss**

$$L(\mathbf{x}, y) = e^{-yf(\mathbf{x})} \tag{13}$$

so that the full empirical objective function is

$$L = \sum_i \exp\left(-\frac{1}{2} y_i \sum_{t=1}^{T} \alpha_t f_t(\mathbf{x}_i) \right) \tag{14}$$

which must be optimize w.r.t. the weights α_t and the parameters of each weak classifier f_t. This stepwise optimization is greedy and sequential, where we add one weak classifier at a time, choosing its parameters and α_t to be optimal w.r.t. L and then never change it again. It turns out that if we actually do keep things constant and solve the optimal parameters, it must be the case that $\alpha_t = \ln \frac{1-\epsilon_t}{\epsilon_t}$, which is why it is in the algorithm.[2] Furthermore, the exponential loss is an upper-bound on the misclassification loss, so if an exponential loss of 0 is achieved, then all training points are correctly classified.

4.2 Gradient Boosting

Gradient boosting can deal with both regression and classification problems, and so we will present it in full generality.

Definition 4.4 (Gradient Boosting)

Let us have data $\{(x_i, y_i)\} \in \mathcal{X} \times \mathcal{Y}$ and a differentiable loss function

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{i=1}^{n} L(y_i, \hat{y}_i) \tag{15}$$

with also a *constant* stepsize η.

1. We first initialize the model with a constant value that minimizes the loss. So we have a single leaf as in our decision tree ensemble.

$$F_0 = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma) \tag{16}$$

   If we're doing regression with the MSE loss, then $\gamma = \bar{y}$, the average. This is our first predictor, which predicts $F_0(x) = \gamma$ for all x, and so F_0 is really just the constant n-vector $(\bar{y}, \dots, \bar{y})$. If we're doing binary classification, we can focus on the logits and initialize γ as the log-odds $\log(\frac{C_+}{C_-})$

2. For $t = 1, \dots, T$, we do the following.
   (a) We have the predicted values $F_{t-1}(x_i)$ for each sample x_i. We compute the negative gradient of the loss function w.r.t. the predicted value.

$$\mathbf{r}_t = -\frac{\partial L(\mathbf{y}, \hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}}\bigg|_{\hat{y}=F_{t-1}(\mathbf{x})} = -\left(\frac{\partial L(y_1, \hat{y}_1)}{\partial \hat{y}_1}\bigg|_{\hat{y}_1=F_{t-1}(x_1)}, \dots, \frac{\partial L(y_n, \hat{y}_n)}{\partial y_n}\bigg|_{\hat{y}_n=F_{t-1}(x_n)} \right) \tag{17}$$

   Note that the vector above is a n-vector, and when we use the MSE loss, then the gradient just simplifies to the residual.
   (b) We use our weak learning algorithm to train a weak model f_t on the residual values $\mathbf{r}_t$.
   (c) We update

$$F_t = F_{t-1} + \eta \cdot f_t \tag{18}$$

3. In the end, we have

$$F_t = F_0 + \eta f_1 + \eta f_2 + \dots + \eta f_T \tag{19}$$

   consisting of a bunch of weak learners to make a strong learner.

[2] Derivation here

In a way, we can think of this as an optimization problem in $\mathbb{R}^n$ (not $\mathbb{R}^d$!). We can think of $\hat{\mathbf{y}}$ representing the actual function f, and we're really doing gradient descent on the "function space" $\mathbb{R}^n$ by updating F_t.

Algorithm 4.2 (Gradient Boosting)

Here is the full algorithm for brevity.

Require: Training data $\{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$
Require: Differentiable loss function $L(y, \hat{y})$
Require: Number of iterations T
Require: Learning rate η
Require: Weak learning algorithm $\mathcal{A}$
// Initialize model with optimal constant value
$F_0 = \text{argmin}_\gamma \sum_{i=1}^n L(y_i, \gamma)$
// For regression (MSE): $F_0 = \frac{1}{n}\sum_{i=1}^n y_i$
// For binary classification: $F_0 = \log(\frac{C_+}{C_-})$
for $t = 1$ to T **do**
    // Compute negative gradient vector
    for $i = 1$ to n **do**
        $r_{t,i} = -\frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i}\big|_{\hat{y}_i = F_{t-1}(x_i)}$
    end for
    // Train weak learner on pseudo-residuals
    $f_t = \mathcal{A}(\{(x_i, r_{t,i})\}_{i=1}^n)$
    // Update model with scaled weak learner
    for $i = 1$ to n **do**
        $F_t(x_i) = F_{t-1}(x_i) + \eta \cdot f_t(x_i)$
    end for
end for
return Final model $F_T(x) = F_0(x) + \eta \sum_{t=1}^T f_t(x)$
// Special cases for common loss functions:
// For MSE: $r_{t,i} = y_i - F_{t-1}(x_i)$ (actual residual)
// For LogLoss: $r_{t,i} = y_i - \sigma(F_{t-1}(x_i))$ where σ is sigmoid

---

Example 4.2 (Regression Trees)

If we have regression trees as our weak learners (pratically the max depth is 8 to 32 rather than stumps) with the L2 loss function.

1. The initial model will just constantly predict the average of the y_i's.
2. The r_t are just the pseudoresiduals

$$r_t = -\big(y_1 - f_{t-1}(x_1), \dots, y_n - f_{t-1}(x_n)\big) \tag{20}$$

3. In case where there are multiple samples running to the same leaf node, the predicted values of the terminal nodes are the average of the y's of those samples.

Example 4.3 (Gradient Boosting Classification)

If we have classification trees as our weak learners, then

1. The initial model will just constantly predict the log odds $\log(C_+/C_-)$, where $C_\pm$ is the number of ones and zeros in the whole dataset. For multiclass there is probably a softmax variant of this.

2. In case where there are multiple samples running to the same leaf node, the predicted values of the terminal nodes are decided by majority.

The general ideas are pretty much the same between AdaBoost and gradient boost. We iteratively build a strong learner from weak learners. A few differences, however,

1. AdaBoost dynamically weighs the importance of each weak model, while gradient boost weak learners are equally weighted by η.
2. AdaBoost actively focuses on the samples where the previous weak learner got wrong, but gradient boost reduces the whole loss in general.
3. Gradient boost usually uses trees larger than stumps.

4.3 XGBoost

The final mainstream boosting algorithm is XGBoost. In regression, XGBoost fits to the residuals just like gradient boosting, but it uses unique regression trees. It is designed for large, complex datasets.

Definition 4.5 (XGBoost for Regression)

Let us have the same data $\{(x_i, y_i)\} \in \mathcal{X} \times \mathcal{Y}$ and the MSE loss

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{21}$$

with a constant stepsize ε (by default 3).

1. We first initialize the model with a constant value that minimizes the loss, which is just the average. So we have a single leaf as in our decision tree ensemble.

$$F_0 = \bar{y} \tag{22}$$

2. For $t = 1, \dots, T$, we do the following.
   (a) We have the predicted values $F_{t-1}(x_i)$ for each sample. We first compute the residuals, denoted $\mathbf{r}_0$. To build our next tree, we start off with a single node "containing" this set of residuals representing each data point.
   (b) We want to grow the decision tree, and we do this by splitting on the maximum gain in *similarity score*, defined for a collection of residuals $\mathbf{r}$ to be

$$s(\mathbf{r}) = \frac{\sum r_i}{\dim(\mathbf{r}) + \lambda} \tag{23}$$

   This score determines how well the set is clustered, and we would like well clustered residuals to be close together.λ is a regularization parameter used to decrease the score's sensitivity when splitting. Therefore, we first compute the score for the root node, and let us define the score of a tree as the sum of the scores of all its leaves. We want to split greedily on this metric. We can keep on splitting until it reaches a certain number of levels (6), and then we can prune it based on whether the increase in score surpasses a threshold, called the *gain*. Note that as λ increases, it is easier to prune the tree.
   (c) With our trained tree f_t, we add it to our cluster to iteratively build our final predictor.

$$F_t = F_{t-1} + \varepsilon \cdot f_t \tag{24}$$

5 Bagging

Let's start off with the simpler of the two.

Definition 5.1 (Bootstrap Aggregating)

Given a dataset $\mathcal{D}$ of N samples and a model $\mathcal{M}$, **bagging** is an ensemble method done with two steps:

1. *Bootstrap*. Sample $\tilde{N}$ data points with replacement from $\mathcal{D}$ to get a dataset $\mathcal{D}_1$, and do this M times to get
$$\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_M \subset \mathcal{D}$$
2. *Aggregate*. For each sub dataset $\mathcal{D}_m$, train our model to get the optimal hypothesis $h^*_{\mathcal{D}_m}$. We should have M different hypothesis functions, each trained on each sub dataset.
$$h^*_{\mathcal{D}_1}, h^*_{\mathcal{D}_2}, \dots, h^*_{\mathcal{D}_M}$$

To predict the output on a new value x, we can evaluate all the $h^*_{\mathcal{D}_m}(x)$ and average/vote them.

Since the whole point of this algorithm is to reduce variance, bagging does not really overfit. Here is a nice example providing intuition on why this works.

Example 5.1 (Theory of Bagging on Toy Dataset)

To get some intuition about why bagging is useful, consider this example from Buhlmann and Yu (2002). Suppose that we have a 1-dimensional dataset $X_1, \dots, X_n \in \mathbb{R}$ that we are trying to split. Consider the simple decision rule.
$$\hat{\theta}(x) = \mathbb{1}(\bar{X}_n \leq x) \tag{25}$$
Now its clear that if we have a different dataset, then we will get a different function, and so given an x, let's try to investigate the behavior of $\hat{\theta}(x)$. Let $\mu = \mathbb{E}[X_i]$ and for simplicity assume that $\text{Var}(X_i) = 1$. If x is really positivce or really negative (just away from μ), then it is clear that $\hat{\theta}(x)$ is almost always 0 or 1, so that's not really interesting. We want to see the variability around μ, where $\hat{\theta}(x)$ could be 0 or 1.

1. So suppose that x is close to μ relative to the sample size. To make this a bit more precise, we can consider a sequence defined $(x_n = \mu + \frac{c}{\sqrt{n}})$. We can use the CLT to approximate $\bar{X} \approx N(\mu, \frac{\sigma^2}{n})$, i.e. $\bar{Y} = \mu + \frac{\sigma}{\sqrt{n}} Z$ for $Z \sim N(0,1)$. Substituting this into our definition of $\hat{\theta}$, we get
$$\begin{aligned} \hat{\theta}(x_n) &= \mathbb{1}(\bar{Y} < x_n) && (26)\\ &= \mathbb{1}\left(\mu + \frac{\sigma}{\sqrt{n}} Z < \mu + \frac{c\sigma}{\sqrt{n}}\right) && (27)\\ &= \mathbb{1}(Z \leq c) && (28)\end{aligned}$$
This is a random variable that can be 0 or 1, and we can compute the mean and variance of this.
$$\mathbb{E}[\hat{\theta}] = \Phi(c), \qquad \text{Var}[\hat{\theta}] = \Phi(c)(1 - \Phi(c)) \tag{29}$$
2. Now, let's look at the bagged version, which we call
$$\hat{\theta}^*(x) = \mathbb{1}(\bar{Y}^* \leq x_n), \qquad \bar{Y}^* = \frac{1}{n} \sum Y_i^* \tag{30}$$
where each $Y_i^* \sim P_n$ is sampled from the empirical distribution function that puts mass $1/n$ on each data point. So conditioned on the original data, we can still use the CLT.
$$\bar{Y}^* \sim N(\bar{Y}, \frac{s^2}{n}) \tag{31}$$

Since s^2 converges in probability to σ^2, we can write

$$\bar{Y}^* \approx \bar{Y} + \frac{\sigma}{\sqrt{n}} Z \tag{32}$$

So substituting this again, we get

$$\hat{\theta}(x) = \mathbb{1}\left(\bar{Y} + \frac{\sigma}{\sqrt{n}} Z \leq x_n\right) = \mathbb{1}\left(Z \leq \frac{\sqrt{n}(x_n - \bar{Y})}{\sigma}\right) \tag{33}$$

But this is just a Bernoulli random variable with probability equal to the CDF of the normal distribution, so by taking the randomness over the bootstrap samples, we have the random variable in Z

$$\mathbb{E}[\hat{\theta}(x)] = \Phi\left(\frac{\sqrt{n}(x_n - \bar{Y})}{\sigma}\right) = \Phi(c - Z) \tag{34}$$

In other words, we bootstrap and take this indicator function, then bootstrap and take another indicator, and keep doing this. Then I take an average of these indicator functions. This average of a bunch of step functions ends up looking like a normal CDF. Since Z is a standard normal, $\Phi(Z) \sim \text{Uniform}(0, 1)$. Computing the mean and variance over the randomness of the original data is

$$\mathbb{E}[\Phi(-Z)] = \mathbb{E}[\Phi(Z)] = \frac{1}{2}, \qquad \text{Var}[\Phi(Z)] = \frac{1}{12} \tag{35}$$

To summarize, the unbagged version satisfies $\hat{\theta}_n \approx \mathbb{1}(Z \leq c)$ while the bagged version has $\hat{\theta}^* \approx \Phi(c + Z)$ which is a smoothed version of $\mathbb{1}(Z \leq c)$. In other words, bagging is a smoothing operator. If we take $c = 0$, then $\hat{\theta}$ converges to a Bernoulli with mean $1/2$ and variance $1/4$. The bagged estimator converges to a uniform with mean $1/2$ and variance $1/12$, which is a reduction in variance.

5.1 Random Forests

In random forests, you are doing baggging with decision trees but with a twist.

Definition 5.2 (Random Forests)

A **random forest** is an ensemble of trees, where each tree $\mathcal{T}_i$ is trained as such:

1. Take a bootstrap sample $\mathcal{D}_i$ to train the tree.
2. Every time you split, choose the splitting variable from a random subset of the d covariates to split on.[a]
3. Then average the predictions.

[a] A heuristic is to take about $\sqrt{d}$ features.

In a sense, this works better because the extra subsampling of the features make each tree less correlated. It's similar to dropout in deep learning.

5.2 Pasting

Definition 5.3 (Pasting)

If random subsets (without replacement) are sampled from the original dataset $\mathcal{D}$, then this method is known as **pasting**.

Bibliography

[Que49] M. H. Quenouille. Approximate tests of correlation in time-series. *Journal of the Royal Statistical Society. Series B (Methodological)*, 11(1):68–84, 1949.