A Brief Insight Into Bagging & Boosting

Presenters:
Siba Smarak Panigrahi & Sohan Patnaik

Special Session
Kharagpur Data Analytics Group
IIT Kharagpur

May 02, 2021

Outline

Introduction to Bagging: Ensemble Paradigms

There are two ways to ensemble:

- **Parallel:**
  - Base learners are generated in parallel
  - Exploits the independence between base learners
- **Sequential:**
  - Base learners are generated sequentially
  - Exploits the dependence between base learners

Bagging

Consider a binary classification problem on classes {-1, 1}. Suppose the ground truth function is f, and each base classifier has an independent generalization error ϵ, i.e., for base classifier h_i,

$$P(h_i(x) \neq f(x)) = \epsilon \tag{1}$$

After combining T number of such base classifiers according to

$$H(x) = sign\Big(\sum_{i=1}^{T} h_i(x)\Big), \tag{2}$$

the ensemble H makes an error only when at least half of its base classifiers make errors.

Bagging

Therefore, by **Hoeffding Inequality**, the generalization error of ensemble is

$$P(H(x) \neq f(x)) = \sum_{k=0}^{\lfloor T/2 \rfloor} \binom{T}{k} (1-\epsilon)^k \epsilon^{T-k} \leq exp\left(-\frac{1}{2}T(2\epsilon - 1)^2\right) \quad (3)$$

Equation 3 shows the generalization error reduces exponentially w.r.t ensemble size T (approaches to zero as T $\rightarrow \infty$)

Note: Another benefit of parallel ensemble methods is that they are inherently favourable to parallel computing, and training speed can be accelerated.

Bagging (Bootstrap Aggregating)

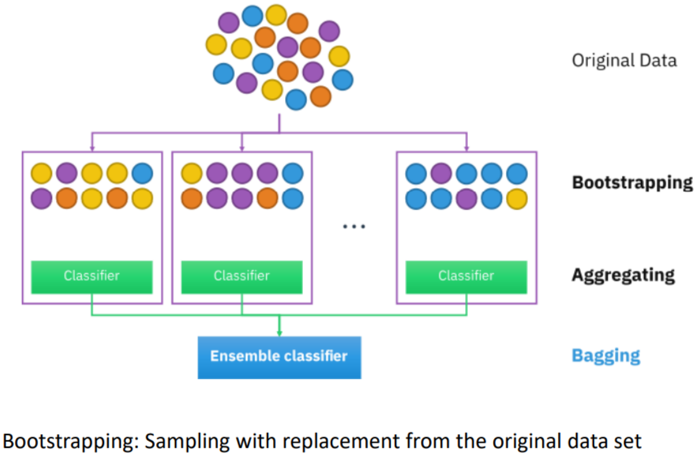

Figure: Bagging (Bootstrap Aggregating)

Bagging Algorithm

Bagging = Bootstrap AGGregatING
We try to obtain base learners as independent as possible so as to minimize the error. The possible solutions:

- design non-overlapping data subsets: but we do not have infinite training instances
- bootstrap sampling: sampling with replacement. Applying this process say T times we obtain T samples of m training examples.

Aggregating Outputs: *Voting* for classification and *Averaging* for regression.
Bagging feeds all the instances to its base classifiers, collects the outputs, uses aggregation methods (with ties broken arbitrarily) to provide the output. Note that, bagging can handle both binary classification and multi-class classification.

Bagging Algorithm

Algorithm 1: Bagging Algorithm

```
Input: Data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$;
       Base learning algorithm $\mathfrak{L}$;
       Number of base learners $T$.
Result: $H(x) = argmax_{y \in Y} \sum_{t=1}^{T} \mathbb{I}(h_t(x) = y)$
1 for $t = 1, \dots\ T$ do
2     $h_t = \mathfrak{L}(D, D_{bs})$ where $D_{bs}$ is the bootstrap distribution;
3 end
```

The goodness of a base learner can be found from **out-of-bag** examples and thus generalization error can be estimated.

Out Of Bag Examples

Denote H^{oob} as out-of-bag prediction on x, where only learners not trained on x are involved, i.e.,

$$H^{oob} = argmax_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(x) = y).\mathbb{I}(x \notin D_t) \tag{4}$$

The out of bag estimate of the generalization error of Bagging is

$$err^{oob} = \frac{1}{|D|} \sum_{(x,y) \in D} \mathbb{I}(H^{oob}(x) \neq y) \tag{5}$$

Random Forests

Algorithm 2: Random Tree Algorithm

```
Input: Data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$;
       Feature Subset size $K$.
Result: A random decision tree
$N \longleftarrow$ create a tree node based on $D$;
if all instances in same class then return $N$;
$\mathcal{F} \longleftarrow$ set of features that can be split further;
if $\mathcal{F}$ is empty then return $N$;
$\tilde{\mathcal{F}} \longleftarrow$ select $K$ features from $\mathcal{F}$ randomly;
$N.f \longleftarrow$ the feature which has the best split point in $\tilde{\mathcal{F}}$;
$N.p \longleftarrow$ the best split on $N.f$;
$D_l \longleftarrow$ subset of $D$ with values on $N.f$ smaller than $N.p$;
$D_r \longleftarrow$ subset of $D$ with values on $N.f$ no smaller than $N.p$;
$N_l \longleftarrow$ call the process with parameters $(D_l, K)$;
$N_r \longleftarrow$ call the process with parameters $(D_r, K)$;
return $N$;
```

Special Notes on Random Forest

- Extension of bagging with the addition of randomized feature selection. At each step of split selection, RF first randomly selects a subset of features and then carries the conventional split selection procedure on the selected features.
- Parameter K determines the degree of randomness. With $K = 1$, we have a single feature selected randomly, but with $K =$ total number of features, we have traditional deterministic decision tree.
- Suggested K value is logarithm of number of features, but it depends on the task.
- Simple bagging would have used all the features in the creation of traditional decision tree, while RF needs only a random subset of features to be evaluated.

Boosting

- Take a weak learning algorithm (any learning algorithm that gives a classifier that is slightly better than random). Then transform it into a strong classifier, which does a lot better than random.
- We define a vector $p = (p^{(1)}, \cdots, p^{(m)})$ on a training set $(x^{(1)}, y^{(1)}), \cdots (x^{(m)}, y^{(m)})$, as a distribution on the examples if $p^{(i)} \geq 0 \ \forall$ i *and* $\sum_{i=1}^{m} p^{(i)} = 1$
- Consider ϕ_j as weak hypotheses. Then, we say that there is a weak learner with margin $\gamma > 0$ if

$$\sum_{i=1}^{m} p^{(i)} 1\{y^{(i)} \neq \phi_j(x^{(i)})\} \leq \frac{1}{2} - \gamma \tag{6}$$

Thus we assume that there is some classifier that does slightly better than random guessing on the dataset.

Boosting

```
Algorithm 3: Weak Learning Algorithm
Input: A distribution $p^{(1)}, ..., p^{(m)}$ and training set $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$
       with $\sum_{i=1}^{m} p^{(i)} = 1$ and $p^{(i)} > 0$;
Result: A weak classifier $\phi_j : \mathbb{R}^n \longrightarrow \{-1, 1\}$ such that
       $$\sum_{i=1}^{m} p^{(i)} \mathbb{I}\{y^{(i)} \neq \phi_j(x^{(i)})\} \leq \frac{1}{2} - \gamma$$
```

Boosting algorithm can be summarized as:

- Assign each training example equal weight in the dataset
- Obtain a weak-hypothesis that does well on the current weights on training examples. Incorporate it into current classification model
- Re-weight the training examples - where mistakes were made receive higher weight and other receive lower weight.
- Perform repeated re-weighting of the training data.

Adaboost

AdaBoost algorithm is one of the most influential boosting algorithm. First let us look at the algorithm.

Algorithm 4: AdaBoost Algorithm

```
Input: Data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$;
       Base learning algorithm $\mathfrak{L}$;
       Number of base learners $T$.
Result: $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$
1 for $t = 1, \ldots\ T$ do
2     $h_t = \mathfrak{L}(D, D_t)$ % Train a classifier $h_t$ from $D$ under distribution $D_t$;
3     $\epsilon_t = P_{x \sim D_t}(h_t(x) \neq f(x))$; % Evaluate the error of $h_t$;
4     if $\epsilon_t > 0.5$ then break
        $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$; % Determine the weight of $h_t$
        $D_{t+1}(x) = \frac{D_t(x) \exp(-\alpha_t f(x) h_t(x))}{Z_t}$ % Update the distribution, $Z_t$ is
        the normalization factor about which we will discuss later.
5 end
```

Why Exponential Loss?

We have a binary classification problem with class labels {-1, 1}. We define the exponential loss as

$$l_{exp}(h|D) = \mathbb{E}_{x \sim D}[e^{-f(x)h(x)}] \tag{7}$$

The exponential loss is used since it gives an elegant and simple update rule and it is consistent with the goal of minimizing classification error and can be justified by its relationship to the standard log likelihood.
For minimizing loss, partial derivative of loss w.r.t. to H for every x should be zero.

$$\frac{\partial e^{-f(x)H(x)}}{\partial H(x)} = -f(x)e^{-f(x)H(x)} = 0 \tag{8}$$

$$\frac{\partial e^{-f(x)H(x)}}{\partial H(x)} = -e^{-H(x)}P(f(x) = 1|x) + e^{H(x)}P(f(x) = -1|x) = 0 \tag{9}$$

Why Exponential Loss?

$$H(x) = \frac{1}{2} \ln \left(\frac{P(f(x) = 1|x)}{P(f(x) = -1|x)} \right) \tag{10}$$

$$sign(H(x)) = argmax_{y \in \{-1,1\}} P(f(x) = y|x) \tag{11}$$

The above derivation shows that when exponential loss is minimized, the classification error is also minimized, thus exponential loss is a proper optimization target for replacing the non-differentiable loss.

How to find α_t

H as described above is produced by iteratively generating h_t and α_t. The first weak classifier h_1 is obtained by invoking the weak learning algorithm on original distribution. The aim is to find $\alpha_t h_t$ such that the exponential loss is minimized.

$$\begin{aligned} l_{exp}(\alpha_t h_t | D_t) &= \mathbb{E}_{x \sim D_t}[e^{-f(x)\alpha_t h_t(x)}] \\ &= e^{-\alpha_t} P_{x \sim D_t}(f(x) = h_t(x)) + \\ &\quad e^{\alpha_t} P_{x \sim D_t}(f(x) \neq h_t(x)) \\ &= e^{-\alpha_t}(1 - \epsilon_t) + e^{\alpha_t}\epsilon_t \end{aligned}$$

Now to find α_t we have to differentiate the above equation w.r.t α_t and equate it to 0. We get,

$$\alpha_t = \frac{1}{2} ln\Big(\frac{1 - \epsilon_t}{\epsilon_t}\Big) \tag{12}$$

How to find h_t

Consider exponential loss again, the ideal classifier h_t corrects all the mistakes of H_{t-1}.

$$l_{exp}(H_{t-1} + h_t|D) = \mathbb{E}_{x \sim D}[e^{-f(x)H_{t-1}(x)} e^{-f(x)h_t(x)}] \tag{13}$$

Using Taylor expansion of $e^{-f(x)h_t(x)}$ till two degree, the exponential loss is approximated as

$$l_{exp}(H_{t-1} + h_t|D) = \mathbb{E}_{x \sim D}\left[e^{-f(x)H_{t-1}(x)}\left(1 - f(x)h_t(x) + \frac{1}{2}\right)\right] \tag{14}$$

Note that the term $f^2(x) = 1$ and $h_t^2(x) = 1$.

How to find h_t

Therefore the ideal classifier h_t is

$$\begin{aligned}
h_t(x) &= argmin_h l_{exp}(H_{t-1} + h|D) \\
&= argmin_h \mathbb{E}_{x\sim D}\left[e^{-f(x)H_{t-1}(x)}\left(1 - f(x)h(x) + \frac{1}{2}\right)\right] \\
&= argmax_h \mathbb{E}_{x\sim D}\left[e^{-f(x)H_{t-1}(x)} f(x)h(x)\right] \\
&= argmax_h \mathbb{E}_{x\sim D}\left[\frac{e^{-f(x)H_{t-1}(x)}}{\mathbb{E}_{x\sim D}[e^{-f(x)H_{t-1}(x)}]} f(x)h(x)\right]
\end{aligned}$$

Notice that we introduced the term $\mathbb{E}_{x\sim D}[e^{-f(x)H_{t-1}(x)}]$ in the denominator which acts as the normalizing constant we had seen earlier in the algorithm as Z_t. Basically, this makes the term a probability distribution.

How to find h_t

Denote a distribution D_t as

$$D_t(x) = \frac{D(x)e^{-f(x)H_{t-1}(x)}}{\mathbb{E}_{x\sim D}[e^{-f(x)H_{t-1}(x)}]} \tag{15}$$

Now, $h_t(x)$ can be written as

$$\begin{aligned} h_t(x) &= argmax_h \mathbb{E}_{x\sim D}\left[\frac{e^{-f(x)H_{t-1}(x)}}{\mathbb{E}_{x\sim D}[e^{-f(x)H_{t-1}(x)}]} f(x)h(x)\right] \\ &= argmax_h \mathbb{E}_{x\sim D_t}[f(x)h(x)] \end{aligned}$$

Notice that $f(x)h_t(x) = 1 - 2\mathbb{I}(f(x) \neq h_t(x))$, then

$$h_t(x) = argmin_h \mathbb{E}_{x\sim D_t}[\mathbb{I}(f(x) \neq h(x))] \tag{16}$$

So, we see that the weak learner $h_t(x)$ is to be trained under the distribution $D_t(x)$. Now let us look at the relation D_t and D_{t+1}

Relation between D_{t+1} and D_t

$$\begin{aligned} D_{t+1}(x) &= \frac{D(x)e^{-f(x)H_t(x)}}{\mathbb{E}_{x\sim D}[e^{-f(x)H_t(x)}]} \\ &= \frac{D(x)e^{-f(x)H_{t-1}(x)}e^{-f(x)\alpha_t h_t(x)}}{\mathbb{E}_{x\sim D}[e^{-f(x)H_t(x)}]} \\ &= D_t(x)e^{-f(x)\alpha_t h_t(x)}\frac{\mathbb{E}_{x\sim D}[e^{-f(x)H_{t-1}(x)}]}{\mathbb{E}_{x\sim D}[e^{-f(x)H_t(x)}]} \end{aligned}$$

This completes the derivation of relation between D_t and D_{t+1}

Gradient Boost

Steps:

- Given any approximator $F_{m-1}(x)$, the function $\beta_m h(x; a_m)$ is to be viewed as the best greedy step towards the data-based estimate of $F^*(x)$.
- We know, the best steepest-descent step direction is $-\mathbf{g}_m = \{-g_m(x_i)\}_1^N$ in the N - dimensional data space at $F_{m-1}(x)$. Thus we have the direction at each of the input training instances x_i. But we need an approximator to generalize to any other x-values.
- That approximator is $h(x; a_m)$ which produces $\mathbf{h}_m = \{h(x_i; a_m)\}_1^N$ most parallel to $-\mathbf{g}_m \in \mathbb{R}^N$

$$-g_m(x_i) = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \tag{17}$$

- The $h(x; a)$ is most highly correlated with $-g_m(x)$ over the data distribution. It can thus be obtained from the solution

$$a_m = argmin_{a,\beta} \sum_{i=1}^{N} [-g_m(x_i) - \beta h(x_i; a)]^2 \tag{18}$$

  This constrained negative gradient $h(x, a_m)$ is used in place ofthe unconstrained one $g_m(x)$ in the steepest-descent strategy.
- Final steps of updation are:

$$\rho_m = argmin_{\rho} \sum_{i=1}^{N} [L(y_i, F_{m-1} + \rho h(x_i, a_m))] \tag{19}$$

$$F_m(x) = F_{m-1}(x) + \rho_m h(x; a_m) \tag{20}$$

Gradient Boost Algorithm

Algorithm 5: Gradient Boost Algorithm

$$F_0(x) = argmin_\rho \sum_{i=1}^{N} L(y_i, \rho)$$

for $m = 1$ to M do:

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}, i = 1 \cdots N$$

$$a_m = argmin_{a,\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(x_i; a)]^2$$

$$\rho_m = argmin_\rho \sum_{i=1}^{N} [L(y_i, F_{m-1} + \rho h(x_i, a_m))]$$

$$F_m(x) = F_{m-1}(x) + \rho_m h(x; a_m)$$

References

- Zhi-Hua Zhou, Ensemble Methods Foundations and Algorithms **[Link]**
- CS229 Course **[Link]**
- Jerome H. Friedman, Greedy Function Approximation : A Gradient Boosting Machine. **[Link]**

Thank You!