Boosting

Chris Cornwell

May 8, 2025

Outline

Outline

Boosting - General Idea

Boosting is a technique that trains models which each have high Bias – low Variance – and it creates a particular linear combination of these models which is a better fit for the data (this “boosted” model having increased Variance).

Boosting - General Idea

Boosting is a technique that trains models which each have high Bias – low Variance – and it creates a particular linear combination of these models which is a better fit for the data (this “boosted” model having increased Variance).

- Ideally, the simple models, that have high Bias, allow for very efficient training, keeping computation times small.

Boosting - General Idea

Boosting is a technique that trains models which each have high Bias – low Variance – and it creates a particular linear combination of these models which is a better fit for the data (this "boosted" model having increased Variance).

- Ideally, the simple models, that have high Bias, allow for very efficient training, keeping computation times small.
- Two popular algorithms for this general idea are AdaBoost and "Gradient Boosted Decision Trees" – which are used in a tool called XGBoost.

The 0-1 Loss Function and Learning Algorithms

For our assumptions with boosting, we consider the setting of binary classification. There is a clear generalization to multi-label Classification tasks. Additionally, one can adapt it to a Regression task (using a different loss function).

The 0-1 Loss Function and Learning Algorithms

For our assumptions with boosting, we consider the setting of binary classification. There is a clear generalization to multi-label Classification tasks. Additionally, one can adapt it to a Regression task (using a different loss function).

For binary classification, write the empirical 0-1 **loss function** as $\mathcal{L}_{\mathcal{S}}^{01}$; it is defined as follows. Given a prediction function $f : \mathbb{R}^d \to \{1, -1\}$, the loss function on data $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is the probability that $f(\mathbf{x}_i) \neq y_i$. That is,

$$\mathcal{L}_{\mathcal{S}}^{01}(f) = \frac{|\{i \ : \ f(\mathbf{x}_i) \neq y_i\}|}{n}.$$

The 0-1 Loss Function and Learning Algorithms

For our assumptions with boosting, we consider the setting of binary classification. There is a clear generalization to multi-label Classification tasks. Additionally, one can adapt it to a Regression task (using a different loss function).

For binary classification, write the empirical 0-1 **loss function** as $\mathcal{L}_S^{01}$; it is defined as follows. Given a prediction function $f : \mathbb{R}^d \to \{1, -1\}$, the loss function on data $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is the probability that $f(\mathbf{x}_i) \neq y_i$. That is,

$$\mathcal{L}_S^{01}(f) = \frac{|\{i \,:\, f(\mathbf{x}_i) \neq y_i\}|}{n}.$$

Population loss: probability that $f(\mathbf{x}) \neq y$ when $(\mathbf{x}, y)$ is drawn from the population.

The 0-1 Loss Function and Learning Algorithms

For our assumptions with boosting, we consider the setting of binary classification. There is a clear generalization to multi-label Classification tasks. Additionally, one can adapt it to a Regression task (using a different loss function).

For binary classification, write the empirical 0-1 **loss function** as $\mathcal{L}_{\mathcal{S}}^{01}$; it is defined as follows. Given a prediction function $f : \mathbb{R}^d \to \{1, -1\}$, the loss function on data $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ is the probability that $f(\mathbf{x}_i) \neq y_i$. That is,

$$\mathcal{L}_{\mathcal{S}}^{01}(f) = \frac{|\{i \ : \ f(\mathbf{x}_i) \neq y_i\}|}{n}.$$

Population loss: probability that $f(\mathbf{x}) \neq y$ when $(\mathbf{x}, y)$ is drawn from the population.

A **learner** or **learning algorithm** is a procedure or rule that finds (or trains) $f_{\mathcal{S}}$ from given training data $\mathcal{S}$.

Loss with weights

Recall how each empirical loss function $\mathcal{L}_S$ is the average of a "per-example" loss over the set S. In the 0-1 case, a per-example loss of

$$\ell^{01}(f, (\mathbf{x}, y)) = \begin{cases} 0, & \text{if } f(\mathbf{x}) = y \\ 1, & \text{else} \end{cases}$$

gives us that $\mathcal{L}_S^{01}(f)$ is the average over S;

[1] The standard one corresponding to $\mathbf{p} = (1/n, 1/n, \ldots, 1/n)$.

Loss with weights

Recall how each empirical loss function $\mathcal{L}_S$ is the average of a "per-example" loss over the set S. In the 0-1 case, a per-example loss of

$$\ell^{01}(f, (\mathbf{x}, y)) = \begin{cases} 0, & \text{if } f(\mathbf{x}) = y \\ 1, & \text{else} \end{cases}$$

gives us that $\mathcal{L}_S^{01}(f)$ is the average over S; that is

$$\mathcal{L}_S^{01}(f) = \frac{1}{n}\ell^{01}(f, (\mathbf{x}_1, y_1)) + \frac{1}{n}\ell^{01}(f, (\mathbf{x}_2, y_2)) + \ldots + \frac{1}{n}\ell^{01}(f, (\mathbf{x}_n, y_n)).$$

[1] The standard one corresponding to $\mathbf{p} = (1/n, 1/n, \ldots, 1/n)$.

Loss with weights

Recall how each empirical loss function $\mathcal{L}_S$ is the average of a "per-example" loss over the set S. In the 0-1 case, a per-example loss of

$$\ell^{01}(f, (\mathbf{x}, y)) = \begin{cases} 0, & \text{if } f(\mathbf{x}) = y \\ 1, & \text{else} \end{cases}$$

gives us that $\mathcal{L}_S^{01}(f)$ is the average over S; that is

$$\mathcal{L}_S^{01}(f) = \frac{1}{n}\ell^{01}(f, (\mathbf{x}_1, y_1)) + \frac{1}{n}\ell^{01}(f, (\mathbf{x}_2, y_2)) + \ldots + \frac{1}{n}\ell^{01}(f, (\mathbf{x}_n, y_n)).$$

In this average, each point in S is *weighted* equally. However, given a probability vector $\mathbf{p} = (p_1, p_2, \ldots, p_n)$, we could weight points in S according to $\mathbf{p}$:

$$p_1\ell^{01}(f, (\mathbf{x}_1, y_1)) + p_2\ell^{01}(f, (\mathbf{x}_2, y_2)) + \ldots + p_n\ell^{01}(f, (\mathbf{x}_n, y_n)).$$

[1] The standard one corresponding to $\mathbf{p} = (1/n, 1/n, \ldots, 1/n)$.

Loss with weights

Recall how each empirical loss function $\mathcal{L}_{\mathcal{S}}$ is the average of a "per-example" loss over the set $\mathcal{S}$. In the 0-1 case, a per-example loss of

$$\ell^{01}(f, (\mathbf{x}, y)) = \begin{cases} 0, & \text{if } f(\mathbf{x}) = y \\ 1, & \text{else} \end{cases}$$

gives us that $\mathcal{L}_{\mathcal{S}}^{01}(f)$ is the average over $\mathcal{S}$; that is

$$\mathcal{L}_{\mathcal{S}}^{01}(f) = \frac{1}{n}\ell^{01}(f, (\mathbf{x}_1, y_1)) + \frac{1}{n}\ell^{01}(f, (\mathbf{x}_2, y_2)) + \ldots + \frac{1}{n}\ell^{01}(f, (\mathbf{x}_n, y_n)).$$

In this average, each point in $\mathcal{S}$ is *weighted* equally. However, given a probability vector $\mathbf{p} = (p_1, p_2, \ldots, p_n)$, we could weight points in $\mathcal{S}$ according to $\mathbf{p}$:

$$p_1\ell^{01}(f, (\mathbf{x}_1, y_1)) + p_2\ell^{01}(f, (\mathbf{x}_2, y_2)) + \ldots + p_n\ell^{01}(f, (\mathbf{x}_n, y_n)).$$

We'll write $\mathcal{L}_{\mathcal{S}}^{01}(f, \mathbf{p})$ for this weighted loss function. Note that a similar weighted loss exists for any "standard" empirical loss function.[1]

[1] The standard one corresponding to $\mathbf{p} = (1/n, 1/n, \ldots, 1/n)$.

Loss with weights

Recall how each empirical loss function $\mathcal{L}_{\mathcal{S}}$ is the average of a "per-example" loss over the set $\mathcal{S}$. In the 0-1 case, a per-example loss of

$$\ell^{01}(f, (\mathbf{x}, y)) = \begin{cases} 0, & \text{if } f(\mathbf{x}) = y \\ 1, & \text{else} \end{cases}$$

gives us that $\mathcal{L}_{\mathcal{S}}^{01}(f)$ is the average over $\mathcal{S}$; that is

$$\mathcal{L}_{\mathcal{S}}^{01}(f) = \frac{1}{n}\ell^{01}(f, (\mathbf{x}_1, y_1)) + \frac{1}{n}\ell^{01}(f, (\mathbf{x}_2, y_2)) + \ldots + \frac{1}{n}\ell^{01}(f, (\mathbf{x}_n, y_n)).$$

In this average, each point in $\mathcal{S}$ is *weighted* equally. However, given a probability vector $\mathbf{p} = (p_1, p_2, \ldots, p_n)$, we could weight points in $\mathcal{S}$ according to $\mathbf{p}$:

$$p_1\ell^{01}(f, (\mathbf{x}_1, y_1)) + p_2\ell^{01}(f, (\mathbf{x}_2, y_2)) + \ldots + p_n\ell^{01}(f, (\mathbf{x}_n, y_n)).$$

We'll write $\mathcal{L}_{\mathcal{S}}^{01}(f, \mathbf{p})$ for this weighted loss function. Note that a similar weighted loss exists for any "standard" empirical loss function.[1]

Below, we begin with a learner that does okay, but $\mathcal{L}_{\mathcal{S}}^{01}(f_{\mathcal{S}})$ may not be that "small"; however, through weighted loss and adaptively changing $\mathbf{p}$, we build one with much better 0-1 loss.

[1] The standard one corresponding to $\mathbf{p} = (1/n, 1/n, \ldots, 1/n)$.

Weak Learners

For γ, with $0 < \gamma < \frac{1}{2}$, we call a learning algorithm a γ-**weak learner** if, for any $0 < \delta < 1$, there is a positive integer N so that whenever $|\mathcal{S}| \geq N$ then the learning algorithm determines $f_{\mathcal{S}}$ so that $\mathcal{L}^{01}_{\mathcal{S}}(f_{\mathcal{S}}) \leq \frac{1}{2} - \gamma$.

Weak Learners

For γ, with $0 < \gamma < \frac{1}{2}$, we call a learning algorithm a γ-**weak learner** if, for any $0 < \delta < 1$, there is a positive integer N so that whenever $|\mathcal{S}| \geq N$ then the learning algorithm determines $f_{\mathcal{S}}$ so that $\mathcal{L}^{01}_{\mathcal{S}}(f_{\mathcal{S}}) \leq \frac{1}{2} - \gamma$.

Note that, for a sufficiently large set of training data $\mathcal{S}$, a γ-weak learner will produce a prediction function $f_{\mathcal{S}}$ that is incorrect less than half of the time. That is, for small γ, this is a bit better than using a coin flip.

Weak Learners

For γ, with $0 < \gamma < \frac{1}{2}$, we call a learning algorithm a γ-**weak learner** if, for any $0 < \delta < 1$, there is a positive integer N so that whenever $|\mathcal{S}| \geq N$ then the learning algorithm determines $f_{\mathcal{S}}$ so that $\mathcal{L}^{01}_{\mathcal{S}}(f_{\mathcal{S}}) \leq \frac{1}{2} - \gamma$.

Note that, for a sufficiently large set of training data $\mathcal{S}$, a γ-weak learner will produce a prediction function $f_{\mathcal{S}}$ that is incorrect less than half of the time. That is, for small γ, this is a bit better than using a coin flip.
Decision stumps can be viewed as weak learners. More precisely, for data where the label should have a single sign ($+1$ or -1) for all $\mathbf{x}$ that have some coordinate x_j that is in a fixed interval, decision stumps are γ-weak learners for some $0 < \gamma < \frac{1}{6}$.

Weak Learners

For γ, with $0 < \gamma < \frac{1}{2}$, we call a learning algorithm a γ-**weak learner** if, for any $0 < \delta < 1$, there is a positive integer N so that whenever $|\mathcal{S}| \geq N$ then the learning algorithm determines $f_{\mathcal{S}}$ so that $\mathcal{L}^{01}_{\mathcal{S}}(f_{\mathcal{S}}) \leq \frac{1}{2} - \gamma$.

Note that, for a sufficiently large set of training data $\mathcal{S}$, a γ-weak learner will produce a prediction function $f_{\mathcal{S}}$ that is incorrect less than half of the time. That is, for small γ, this is a bit better than using a coin flip.
Decision stumps can be viewed as weak learners. More precisely, for data where the label should have a single sign ($+1$ or -1) for all $\mathbf{x}$ that have some coordinate x_j that is in a fixed interval, decision stumps are γ-weak learners for some $0 < \gamma < \frac{1}{6}$.
Since decision stumps are decision trees that have only depth 1, and they are weak learners for such a classification problem, decision trees are also weak learners.

Weak Learners

For γ, with $0 < \gamma < \frac{1}{2}$, we call a learning algorithm a γ-**weak learner** if, for any $0 < \delta < 1$, there is a positive integer N so that whenever $|\mathcal{S}| \geq N$ then the learning algorithm determines $f_{\mathcal{S}}$ so that $\mathcal{L}^{01}_{\mathcal{S}}(f_{\mathcal{S}}) \leq \frac{1}{2} - \gamma$.

Note that, for a sufficiently large set of training data $\mathcal{S}$, a γ-weak learner will produce a prediction function $f_{\mathcal{S}}$ that is incorrect less than half of the time. That is, for small γ, this is a bit better than using a coin flip.
Decision stumps can be viewed as weak learners. More precisely, for data where the label should have a single sign ($+1$ or -1) for all $\mathbf{x}$ that have some coordinate x_j that is in a fixed interval, decision stumps are γ-weak learners for some $0 < \gamma < \frac{1}{6}$.
Since decision stumps are decision trees that have only depth 1, and they are weak learners for such a classification problem, decision trees are also weak learners.

- Hence, it is common to use decision trees with small max depth as the base model in a boosting algorithm.

Outline

Description of AdaBoost

One chooses a number T of *rounds* for the AdaBoost algorithm. In round t, with $1 \leq t \leq T$, say that f_t is the basic model that is trained in that round. That round will also have a corresponding coefficient w_t. The final (boosted) model is given by

$$f(\mathbf{x}) = \operatorname{sign}(w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \ldots + w_T f_T(\mathbf{x})).$$

Description of AdaBoost

One chooses a number T of *rounds* for the AdaBoost algorithm. In round t, with $1 \leq t \leq T$, say that f_t is the basic model that is trained in that round. That round will also have a corresponding coefficient w_t. The final (boosted) model is given by

$$f(\mathbf{x}) = \operatorname{sign}(w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \ldots + w_T f_T(\mathbf{x})).$$

The procedure for finding f_t and w_t is the following.
In each round t, there is a probability vector $\mathbf{p}^{(t)} \in \mathbb{R}^n$. In round 1, we give $\mathbf{p}^{(1)}$ uniform weights; that is, $\mathbf{p}^{(1)} = (1/n, 1/n, \ldots, 1/n)$.

Description of AdaBoost

One chooses a number T of *rounds* for the AdaBoost algorithm. In round t, with $1 \leq t \leq T$, say that f_t is the basic model that is trained in that round. That round will also have a corresponding coefficient w_t. The final (boosted) model is given by

$$f(\mathbf{x}) = \operatorname{sign}(w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \ldots + w_T f_T(\mathbf{x})).$$

The procedure for finding f_t and w_t is the following.
In each round t, there is a probability vector $\mathbf{p}^{(t)} \in \mathbb{R}^n$. In round 1, we give $\mathbf{p}^{(1)}$ uniform weights; that is, $\mathbf{p}^{(1)} = (1/n, 1/n, \ldots, 1/n)$.
Now, in round t of AdaBoost:

1. Use your learner to minimize weighted loss $\mathcal{L}_S^{01}(f, \mathbf{p}^{(t)})$, getting f_t.

Description of AdaBoost

One chooses a number T of *rounds* for the AdaBoost algorithm. In round t, with $1 \leq t \leq T$, say that f_t is the basic model that is trained in that round. That round will also have a corresponding coefficient w_t. The final (boosted) model is given by

$$f(\mathbf{x}) = \operatorname{sign}(w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \ldots + w_T f_T(\mathbf{x})).$$

The procedure for finding f_t and w_t is the following.
In each round t, there is a probability vector $\mathbf{p}^{(t)} \in \mathbb{R}^n$. In round 1, we give $\mathbf{p}^{(1)}$ uniform weights; that is, $\mathbf{p}^{(1)} = (1/n, 1/n, \ldots, 1/n)$.
Now, in round t of AdaBoost:

1. Use your learner to minimize weighted loss $\mathcal{L}_S^{01}(f, \mathbf{p}^{(t)})$, getting f_t.
2. Compute the weighted empirical loss $\varepsilon_t = \mathcal{L}_S(f_t, \mathbf{p}^{(t)})$.

Description of AdaBoost

One chooses a number T of *rounds* for the AdaBoost algorithm. In round t, with $1 \leq t \leq T$, say that f_t is the basic model that is trained in that round. That round will also have a corresponding coefficient w_t. The final (boosted) model is given by

$$f(\mathbf{x}) = \operatorname{sign}(w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \ldots + w_T f_T(\mathbf{x})).$$

The procedure for finding f_t and w_t is the following.
In each round t, there is a probability vector $\mathbf{p}^{(t)} \in \mathbb{R}^n$. In round 1, we give $\mathbf{p}^{(1)}$ uniform weights; that is, $\mathbf{p}^{(1)} = (1/n, 1/n, \ldots, 1/n)$.
Now, in round t of AdaBoost:

1. Use your learner to minimize weighted loss $\mathcal{L}_S^{01}(f, \mathbf{p}^{(t)})$, getting f_t.
2. Compute the weighted empirical loss $\varepsilon_t = \mathcal{L}_S(f_t, \mathbf{p}^{(t)})$.
3. Define $w_t = \ln\left(\frac{1}{\varepsilon_t} - 1\right)$.

Description of AdaBoost

One chooses a number T of *rounds* for the AdaBoost algorithm. In round t, with $1 \leq t \leq T$, say that f_t is the basic model that is trained in that round. That round will also have a corresponding coefficient w_t. The final (boosted) model is given by

$$f(\mathbf{x}) = \operatorname{sign}(w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \ldots + w_T f_T(\mathbf{x})).$$

The procedure for finding f_t and w_t is the following.
In each round t, there is a probability vector $\mathbf{p}^{(t)} \in \mathbb{R}^n$. In round 1, we give $\mathbf{p}^{(1)}$ uniform weights; that is, $\mathbf{p}^{(1)} = (1/n, 1/n, \ldots, 1/n)$.
Now, in round t of AdaBoost:

1. Use your learner to minimize weighted loss $\mathcal{L}_S^{01}(f, \mathbf{p}^{(t)})$, getting f_t.
2. Compute the weighted empirical loss $\varepsilon_t = \mathcal{L}_S(f_t, \mathbf{p}^{(t)})$.
3. Define $w_t = \ln\left(\frac{1}{\varepsilon_t} - 1\right)$.
4. Find $\mathbf{p}^{(t+1)} = (p_1^{(t+1)}, \ldots, p_n^{(t+1)})$ by setting

$$p_i^{(t+1)} = \frac{p_i^{(t)} e^{-w_t y_i f_t(\mathbf{x}_i)}}{\sum_{j=1}^n p_j^{(t)} e^{-w_t y_j f_t(\mathbf{x}_j)}}.$$

Description of AdaBoost

One chooses a number T of *rounds* for the AdaBoost algorithm. In round t, with $1 \leq t \leq T$, say that f_t is the basic model that is trained in that round. That round will also have a corresponding coefficient w_t. The final (boosted) model is given by

$$f(\mathbf{x}) = \operatorname{sign}(w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \ldots + w_T f_T(\mathbf{x})).$$

The procedure for finding f_t and w_t is the following.
In each round t, there is a probability vector $\mathbf{p}^{(t)} \in \mathbb{R}^n$. In round 1, we give $\mathbf{p}^{(1)}$ uniform weights; that is, $\mathbf{p}^{(1)} = (1/n, 1/n, \ldots, 1/n)$.
Now, in round t of AdaBoost:

1. Use your learner to minimize weighted loss $\mathcal{L}_S^{01}(f, \mathbf{p}^{(t)})$, getting f_t.
2. Compute the weighted empirical loss $\varepsilon_t = \mathcal{L}_S(f_t, \mathbf{p}^{(t)})$.
3. Define $w_t = \ln\left(\frac{1}{\varepsilon_t} - 1\right)$.
4. Find $\mathbf{p}^{(t+1)} = (p_1^{(t+1)}, \ldots, p_n^{(t+1)})$ by setting

$$p_i^{(t+1)} = \frac{p_i^{(t)} e^{-w_t y_i f_t(\mathbf{x}_i)}}{\sum_{j=1}^n p_j^{(t)} e^{-w_t y_j f_t(\mathbf{x}_j)}}.$$

Once done with T rounds, the boosted model is the one given by linear combination above.