Learning Algorithms From Game Theory II: Boosting

Aaron Roth

University of Pennsylvania

April 30 2024

Overview

- In this lecture, we'll continue our exploration into how one can derive powerful machine learning algorithms from game theoretic principles.

Overview

- In this lecture, we'll continue our exploration into how one can derive powerful machine learning algorithms from game theoretic principles.
- We'll focus on the general and empirically successful paradigm of *boosting*.

Overview

- In this lecture, we'll continue our exploration into how one can derive powerful machine learning algorithms from game theoretic principles.
- We'll focus on the general and empirically successful paradigm of *boosting*.
- Boosting addresses the question of how one can combine classifiers that individually do (just) a little bit better than random guessing, into powerful predictive models.

Setting

Definition

A labeled *datapoint* is a pair $(x, y) \in X \times Y$, where X is some space of *features* and Y is some space of *labels*: for example, a common case is $X = \mathbb{R}^d$, and $Y = \{0, 1\}$.
A dataset $D \in (X \times Y)^n$ is a collection of n labeled datapoints.

Setting

Definition

A labeled *datapoint* is a pair $(x, y) \in X \times Y$, where X is some space of *features* and Y is some space of *labels*: for example, a common case is $X = \mathbb{R}^d$, and $Y = \{0, 1\}$.
A dataset $D \in (X \times Y)^n$ is a collection of n labeled datapoints.

Our goal: find some function $f : X \to Y$ for predicting labels from their features that has high accuracy.

Setting

Definition

Given a predictor $f : X \to Y$, its prediction accuracy on a dataset D is:

$$acc(f, D) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[f(x_i) = y_i]$$

The prediction accuracy as defined uniformly weights all of the points in the dataset. But we can also define *weighted* prediction accuracy relative to any other weighting $w \in \Delta[n]$ of the n points:

$$acc(f, D, w) = \sum_{i=1}^{n} w_i \mathbb{1}[f(x_i) = y_i]$$

Note that $acc(f, D)$ is simply the special case of $acc(f, D, w)$ in which $w_i = 1/n$ for all i.

An Aside

1. We're ignoring an important statistical aspect of machine learning!

An Aside

1. We're ignoring an important statistical aspect of machine learning!
2. The goal is not to predict the labels of points in our dataset D (we already know them!) but to predict well on new points drawn from the same distribution.

An Aside

1. We're ignoring an important statistical aspect of machine learning!
2. The goal is not to predict the labels of points in our dataset D (we already know them!) but to predict well on new points drawn from the same distribution.
3. Informally, to do this it suffices to predict the labels in D accurately with "simple" hypotheses.

An Aside

1. We're ignoring an important statistical aspect of machine learning!
2. The goal is not to predict the labels of points in our dataset D (we already know them!) but to predict well on new points drawn from the same distribution.
3. Informally, to do this it suffices to predict the labels in D accurately with "simple" hypotheses.
4. The boosting approach in this lecture does this, but we'll just focus on the algorithmic aspects.

Setting

Definition

A hypothesis class H is a collection of predictors or *hypotheses* $h : X \to Y$. A weighted learning algorithm A with range H is a mapping from datasets and weight vectors to hypotheses in H. $A : (X \times Y)^n \times [0, 1]^n \to H$.

Setting

Definition

A hypothesis class H is a collection of predictors or *hypotheses* $h : X \to Y$. A weighted learning algorithm A with range H is a mapping from datasets and weight vectors to hypotheses in H. $A : (X \times Y)^n \times [0,1]^n \to H$.

1. If $Y = \{0, 1\}$ then it is uninteresting to find a hypothesis h with $acc(h, D) \leq 1/2$

Setting

Definition

A hypothesis class H is a collection of predictors or *hypotheses* $h : X \to Y$. A weighted learning algorithm A with range H is a mapping from datasets and weight vectors to hypotheses in H. $A : (X \times Y)^n \times [0,1]^n \to H$.

1. If $Y = \{0, 1\}$ then it is uninteresting to find a hypothesis h with $acc(h, D) \leq 1/2$
2. Could have done this by random guessing!

Setting

Definition

A hypothesis class H is a collection of predictors or *hypotheses* $h : X \to Y$. A weighted learning algorithm A with range H is a mapping from datasets and weight vectors to hypotheses in H. $A : (X \times Y)^n \times [0,1]^n \to H$.

1. If $Y = \{0,1\}$ then it is uninteresting to find a hypothesis h with $acc(h, D) \leq 1/2$
2. Could have done this by random guessing!
3. Want accuracy more like 0.99...

Setting

Definition

A hypothesis class H is a collection of predictors or *hypotheses* $h : X \to Y$. A weighted learning algorithm A with range H is a mapping from datasets and weight vectors to hypotheses in H. $A : (X \times Y)^n \times [0,1]^n \to H$.

1. If $Y = \{0, 1\}$ then it is uninteresting to find a hypothesis h with $acc(h, D) \leq 1/2$
2. Could have done this by random guessing!
3. Want accuracy more like 0.99...
4. But what if you can reliably get accuracy 0.51?

Setting

Definition

A hypothesis class H is a collection of predictors or *hypotheses* $h : X \to Y$. A weighted learning algorithm A with range H is a mapping from datasets and weight vectors to hypotheses in H. $A : (X \times Y)^n \times [0,1]^n \to H$.

1. If $Y = \{0,1\}$ then it is uninteresting to find a hypothesis h with $acc(h, D) \leq 1/2$
2. Could have done this by random guessing!
3. Want accuracy more like 0.99...
4. But what if you can reliably get accuracy 0.51?
5. An algorithm that can guarantee this is a *weak learner*

Weak Learning

Definition

A weighted learning algorithm A is a weak learning algorithm for D if for every distribution $w \in \Delta[n]$, $A(D, w) = h$ such that:

$$acc(h, D, w) \geq 0.51$$

Weak Learning

Definition

A weighted learning algorithm A is a weak learning algorithm for D if for every distribution $w \in \Delta[n]$, $A(D, w) = h$ such that:

$$acc(h, D, w) \geq 0.51$$

1. *Weighted* learning algorithm?

Weak Learning

Definition

A weighted learning algorithm A is a weak learning algorithm for D if for every distribution $w \in \Delta[n]$, $A(D, w) = h$ such that:

$$acc(h, D, w) \geq 0.51$$

1. *Weighted* learning algorithm?
2. Most learning algorithms can handle weights — just weight points in the objective function.

Weak Learning

Definition

A weighted learning algorithm A is a weak learning algorithm for D if for every distribution $w \in \Delta[n]$, $A(D, w) = h$ such that:

$$acc(h, D, w) \geq 0.51$$

1. *Weighted* learning algorithm?
2. Most learning algorithms can handle weights — just weight points in the objective function.
3. If yours can't, construct a new dataset D' by sampling from D under the probability distribution specified by w, and then run your algorithm on D'.

Weak Learning

Definition

A weighted learning algorithm A is a weak learning algorithm for D if for every distribution $w \in \Delta[n]$, $A(D, w) = h$ such that:

$$acc(h, D, w) \geq 0.51$$

1. *Weighted* learning algorithm?
2. Most learning algorithms can handle weights — just weight points in the objective function.
3. If yours can't, construct a new dataset D' by sampling from D under the probability distribution specified by w, and then run your algorithm on D'.
4. So weights are without loss of generality.

Weak Learning

1. Weak learning algorithms seem weak!

Weak Learning

1. Weak learning algorithms seem weak!
2. We want more like 99% accuracy! Strong learning algorithms!

Weak Learning

1. Weak learning algorithms seem weak!
2. We want more like 99% accuracy! Strong learning algorithms!
3. Can we get strong learning from weak learning?

Weak Learning

1. Weak learning algorithms seem weak!
2. We want more like 99% accuracy! Strong learning algorithms!
3. Can we get strong learning from weak learning?

Definition

A is a strong learning algorithm for D if $A(D) = h$ such that $acc(h, D) = 1$.

Weak Learning and Strong Learning are Equivalent

Theorem

For any dataset D, if there exists an efficient (polynomial time) weak learning algorithm A for D, then there exists an efficient strong learning algorithm A' for D.

Weak Learning and Strong Learning are Equivalent

Theorem

For any dataset D, if there exists an efficient (polynomial time) weak learning algorithm A for D, then there exists an efficient strong learning algorithm A' for D.

Proof Idea: Study the appropriately defined zero sum game. Then compute the equilibrium strategy in that game.

Proof

1. Let H be the hypothesis class used by the weak learning algorithm A.

Proof

1. Let H be the hypothesis class used by the weak learning algorithm A.
2. Define the following 2-player zero-sum game:

Proof

1. Let H be the hypothesis class used by the weak learning algorithm A.
2. Define the following 2-player zero-sum game:
   2.1 The action space for the minimization player (the "Data Player") is the set of datapoints in the dataset: $A_1 = D$.

Proof

1. Let H be the hypothesis class used by the weak learning algorithm A.
2. Define the following 2-player zero-sum game:
   2.1 The action space for the minimization player (the "Data Player") is the set of datapoints in the dataset: $A_1 = D$.
   2.2 The action space for the maximization player (the "Learner") is $A_2 = H$.

Proof

1. Let H be the hypothesis class used by the weak learning algorithm A.
2. Define the following 2-player zero-sum game:
    2.1 The action space for the minimization player (the "Data Player") is the set of datapoints in the dataset: $A_1 = D$.
    2.2 The action space for the maximization player (the "Learner") is $A_2 = H$.
    2.3 The cost function is C is defined as $C((x_i, y_i), h) = \mathbb{1}[h(x_i) = y_i]$.

Proof

1. Let H be the hypothesis class used by the weak learning algorithm A.
2. Define the following 2-player zero-sum game:
   2.1 The action space for the minimization player (the "Data Player") is the set of datapoints in the dataset: $A_1 = D$.
   2.2 The action space for the maximization player (the "Learner") is $A_2 = H$.
   2.3 The cost function is C is defined as $C((x_i, y_i), h) = \mathbb{1}[h(x_i) = y_i]$.
3. What is the value of this game?

Proof

1. The existence of a weak learning algorithm implies the value of the game is at least 0.51!

Proof

1. The existence of a weak learning algorithm implies the value of the game is at least 0.51!

$$\min\max(C) \quad = \quad \min_{w \in \Delta[n]} \max_{h \in H} \sum_{i=1}^{n} w_i C((x_i, y_i), h)$$

Proof

1. The existence of a weak learning algorithm implies the value of the game is at least 0.51!

$$\begin{aligned}
\min\max(C) &= \min_{w\in\Delta[n]} \max_{h\in H} \sum_{i=1}^{n} w_i C((x_i, y_i), h) \\
&= \min_{w\in\Delta[n]} \max_{h\in H} w_i \mathbb{1}[h(x_i) = y_i]
\end{aligned}$$

Proof

1. The existence of a weak learning algorithm implies the value of the game is at least 0.51!

$$
\begin{aligned}
\min\max(C) &= \min_{w \in \Delta[n]} \max_{h \in H} \sum_{i=1}^{n} w_i C((x_i, y_i), h) \\
&= \min_{w \in \Delta[n]} \max_{h \in H} w_i \mathbb{1}[h(x_i) = y_i] \\
&= \min_{w \in \Delta[n]} \max_{h \in H} acc(h, D, w)
\end{aligned}
$$

Proof

1. The existence of a weak learning algorithm implies the value of the game is at least 0.51!

$$
\begin{aligned}
\min\max(C) &= \min_{w \in \Delta[n]} \max_{h \in H} \sum_{i=1}^{n} w_i C((x_i, y_i), h) \\
&= \min_{w \in \Delta[n]} \max_{h \in H} w_i \mathbb{1}[h(x_i) = y_i] \\
&= \min_{w \in \Delta[n]} \max_{h \in H} acc(h, D, w) \\
&\geq 0.51
\end{aligned}
$$

Proof

1. The minimax theorem implies the Learner can do just as well, even if she is forced to commit to her strategy first:

$$\min\max(C) = \max\min(C) = \max_{p\in\Delta H}\min_{i\in[n]}\sum_{h\in H} p_h \mathbb{1}[h(x_i) = y_i] \geq 0.51$$

Proof

1. The minimax theorem implies the Learner can do just as well, even if she is forced to commit to her strategy first:

$$\min\max(C) = \max\min(C) = \max_{p \in \Delta H} \min_{i \in [n]} \sum_{h \in H} p_h \mathbb{1}[h(x_i) = y_i] \geq 0.51$$

2. i.e. there is a *fixed* distribution p^* over hypotheses $h \in H$ such that for *every* data point $(x_i, y_i) \in D$, at least 51% of the probability mass under p is on hypotheses that correctly label (x_i, y_i).

Proof

1. The minimax theorem implies the Learner can do just as well, even if she is forced to commit to her strategy first:

$$\min\max(C) = \max\min(C) = \max_{p \in \Delta H} \min_{i \in [n]} \sum_{h \in H} p_h \mathbb{1}[h(x_i) = y_i] \geq 0.51$$

2. i.e. there is a *fixed* distribution p^* over hypotheses $h \in H$ such that for *every* data point $(x_i, y_i) \in D$, at least 51% of the probability mass under p is on hypotheses that correctly label (x_i, y_i).
3. So consider the following "majority vote" classification rule f_{p^*}:

$$f_{p^*}(x) = \mathbb{1}\left[\sum_{h:h(x)=1} p^*_h \geq 0.5\right]$$

Proof

1. The minimax theorem implies the Learner can do just as well, even if she is forced to commit to her strategy first:

$$\min\max(C) = \max\min(C) = \max_{p \in \Delta H} \min_{i \in [n]} \sum_{h \in H} p_h \mathbb{1}[h(x_i) = y_i] \geq 0.51$$

2. i.e. there is a *fixed* distribution p^* over hypotheses $h \in H$ such that for *every* data point $(x_i, y_i) \in D$, at least 51% of the probability mass under p is on hypotheses that correctly label (x_i, y_i).
3. So consider the following "majority vote" classification rule f_{p^*}:

$$f_{p^*}(x) = \mathbb{1}\left[\sum_{h:h(x)=1} p^*_h \geq 0.5\right]$$

4. f_{p^*} must have perfect accuracy...

Proof

Lemma

For the distribution $p^* = \max_{p \in \Delta H} \min_{i \in [n]} \sum_{h \in H} p_h \mathbb{1}[h(x_i) = y_i]$*, the hypothesis* f_{p^*} *satisfies* $acc(f_{p^*}, D) = 1$

Proof:

Proof

Lemma

For the distribution $p^* = \max_{p \in \Delta H} \min_{i \in [n]} \sum_{h \in H} p_h \mathbb{1}[h(x_i) = y_i]$*, the hypothesis* f_{p^*} *satisfies* $acc(f_{p^*}, D) = 1$

Proof:

1. Fix any $(x_i, y_i) \in D$. We must show $f_{p^*}(x_i) = y_i$.

Proof

Lemma

For the distribution $p^* = \max_{p \in \Delta H} \min_{i \in [n]} \sum_{h \in H} p_h \mathbb{1}[h(x_i) = y_i]$, *the hypothesis* f_{p^*} *satisfies* $acc(f_{p^*}, D) = 1$

Proof:

1. Fix any $(x_i, y_i) \in D$. We must show $f_{p^*}(x_i) = y_i$.
2. From our minimax calculation, we know that for every i, $\sum_{h \in H} p^*_h \mathbb{1}[h(x_i) = y_i] \geq 0.51$.

Proof

Lemma

For the distribution $p^* = \max_{p \in \Delta H} \min_{i \in [n]} \sum_{h \in H} p_h \mathbb{1}[h(x_i) = y_i]$*, the hypothesis* f_{p^*} *satisfies* $acc(f_{p^*}, D) = 1$

Proof:

1. Fix any $(x_i, y_i) \in D$. We must show $f_{p^*}(x_i) = y_i$.
2. From our minimax calculation, we know that for every i, $\sum_{h \in H} p^*_h \mathbb{1}[h(x_i) = y_i] \geq 0.51$.
3. If $y_i = 1$, we have that

$$\sum_{h \in H} p^*_h \mathbb{1}[h(x_i) = 1] = \sum_{h : h(x) = 1} p^*_h \geq 0.51$$

and hence by definition $f_{p^*}(x_i) = 1$.

Proof

Lemma

For the distribution $p^* = \max_{p \in \Delta H} \min_{i \in [n]} \sum_{h \in H} p_h \mathbb{1}[h(x_i) = y_i]$, *the hypothesis* f_{p^*} *satisfies* $acc(f_{p^*}, D) = 1$

Proof:

1. Fix any $(x_i, y_i) \in D$. We must show $f_{p^*}(x_i) = y_i$.
2. From our minimax calculation, we know that for every i, $\sum_{h \in H} p_h^* \mathbb{1}[h(x_i) = y_i] \geq 0.51$.
3. If $y_i = 1$, we have that

$$\sum_{h \in H} p_h^* \mathbb{1}[h(x_i) = 1] = \sum_{h : h(x) = 1} p_h^* \geq 0.51$$

   and hence by definition $f_{p^*}(x_i) = 1$.
4. Similarly, if $y_i = 0$, we know that $\sum_{h : h(x) = 1} p_h^* < 0.49$ and hence by definition $f_{p^*}(x_i) = 0$.

Proof

Lemma

For the distribution $p^* = \max_{p \in \Delta H} \min_{i \in [n]} \sum_{h \in H} p_h \mathbb{1}[h(x_i) = y_i]$, *the hypothesis* f_{p^*} *satisfies* $acc(f_{p^*}, D) = 1$

Proof:

1. Fix any $(x_i, y_i) \in D$. We must show $f_{p^*}(x_i) = y_i$.
2. From our minimax calculation, we know that for every i, $\sum_{h \in H} p^*_h \mathbb{1}[h(x_i) = y_i] \geq 0.51$.
3. If $y_i = 1$, we have that

$$\sum_{h \in H} p^*_h \mathbb{1}[h(x_i) = 1] = \sum_{h: h(x)=1} p^*_h \geq 0.51$$

   and hence by definition $f_{p^*}(x_i) = 1$.
4. Similarly, if $y_i = 0$, we know that $\sum_{h:h(x)=1} p^*_h < 0.49$ and hence by definition $f_{p^*}(x_i) = 0$.
5. Tada!

Proof

1. But what is the efficient algorithm?

Proof

1. But what is the efficient algorithm?
2. Note that $f_{p^*}(x_i)$ is easy to compute if we know p^*.

Proof

1. But what is the efficient algorithm?
2. Note that $f_{p^*}(x_i)$ is easy to compute if we know p^*.
3. So the algorithmic problem is to compute an equilibrium strategy p^* for our learning game, that makes only a polynomial number of calls to our weak learning algorithm.

Proof

1. But what is the efficient algorithm?
2. Note that $f_{p^*}(x_i)$ is easy to compute if we know p^*.
3. So the algorithmic problem is to compute an equilibrium strategy p^* for our learning game, that makes only a polynomial number of calls to our weak learning algorithm.
4. In fact, we only need an ϵ-approximate equilibrium for $\epsilon < 0.01$...

Proof

1. But what is the efficient algorithm?
2. Note that $f_{p^*}(x_i)$ is easy to compute if we know p^*.
3. So the algorithmic problem is to compute an equilibrium strategy p^* for our learning game, that makes only a polynomial number of calls to our weak learning algorithm.
4. In fact, we only need an ϵ-approximate equilibrium for $\epsilon < 0.01$...
5. We know how to do that by having the data player play polynomial weights over the data points, and the learner best respond.

Proof

1. But what is the efficient algorithm?
2. Note that $f_{p^*}(x_i)$ is easy to compute if we know p^*.
3. So the algorithmic problem is to compute an equilibrium strategy p^* for our learning game, that makes only a polynomial number of calls to our weak learning algorithm.
4. In fact, we only need an ϵ-approximate equilibrium for $\epsilon < 0.01$...
5. We know how to do that by having the data player play polynomial weights over the data points, and the learner best respond.
6. And we can implement best responses using the weak learning algorithm...

Proof

Algorithm 1 Boost(D, A)

Let $T \leftarrow \frac{4 \log n}{\epsilon^2}$ for $\epsilon < 0.01$.
Initialize a copy of polynomial weights to run over $w^t \in \Delta^n$.
for $t = 1$ to T **do**
  Let $h^t = A(D, w^t)$
  Let $\ell^t \in [0,1]^m$ be such that $\ell_i^t = \mathbb{1}[h^t(x_i) = y_i]$.
  Pass ℓ^t to the PW algorithm.
end for
Let $\hat{p} = \frac{1}{T}\sum_{t=1}^{T} e_{h^t}$. (Note that this is concisely representable even though H is large, because $\hat{p}$ has support over only the T models h^t.)
Return $f_{\hat{p}}(x)$.

Proof

1. Since ϵ is a constant, on a dataset of size n, the algorithm runs for only $O(\log n)$ many iterations.

Proof

1. Since ϵ is a constant, on a dataset of size n, the algorithm runs for only $O(\log n)$ many iterations.
2. At each iteration it makes a single call to our weak learning algorithm A.

Proof

1. Since ϵ is a constant, on a dataset of size n, the algorithm runs for only $O(\log n)$ many iterations.
2. At each iteration it makes a single call to our weak learning algorithm A.
3. It then has to update the polynomial weights distribution over the n datapoints, which takes time $O(n)$.

Proof

1. Since ϵ is a constant, on a dataset of size n, the algorithm runs for only $O(\log n)$ many iterations.
2. At each iteration it makes a single call to our weak learning algorithm A.
3. It then has to update the polynomial weights distribution over the n datapoints, which takes time $O(n)$.
4. Total running time is $O(\log n(n + R(A)))$, where $R(A)$ is the running time of our weak learning algorithm.

Briefly: Statistical Aspects

1. Our final hypothesis is a majority vote over $O(\log n)$ hypotheses from H.

Briefly: Statistical Aspects

1. Our final hypothesis is a majority vote over $O(\log n)$ hypotheses from H.
2. So the complexity is not much larger than the complexity of individual weak hypotheses in H.

Briefly: Statistical Aspects

1. Our final hypothesis is a majority vote over $O(\log n)$ hypotheses from H.
2. So the complexity is not much larger than the complexity of individual weak hypotheses in H.
3. Hence (informally), the amount of data needed for the strong learner to generalize "out of sample" is not much larger than was needed for the weak learner.

Briefly: Statistical Aspects

1. Our final hypothesis is a majority vote over $O(\log n)$ hypotheses from H.
2. So the complexity is not much larger than the complexity of individual weak hypotheses in H.
3. Hence (informally), the amount of data needed for the strong learner to generalize "out of sample" is not much larger than was needed for the weak learner.
4. This can be formalized in various ways, including with a measure called "VC-Dimension".

Thanks!

See you next class — stay healthy!

Thanks!

~~See you next class~~ — stay healthy!

Thanks!

Have a great summer!