CSCI567 Machine Learning (Spring 2021)

Sirisha Rambhatla

University of Southern California

March 10, 2021

Outline

1. Logistics
2. Review of last lecture
3. Boosting

Outline

Logistics

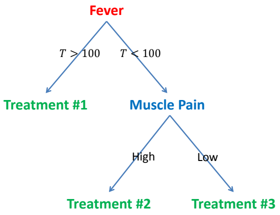
- Checkpoint 1 for project is due today. We will be tracking Kaggle submissions starting March 11, 2021.
- Quiz 1 has been graded. The mean, median, and standard deviation were 60.3, 62.3 and 17.4, respectively.
- March 12, 2021 is a **Wellness Day**, there will be no class.

Outline

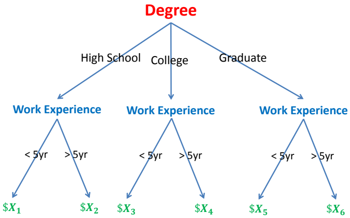
Decision Trees

Many decisions are made based on some tree structure

Medical treatment



Salary in a company



Learning Decision Trees

DecisionTreeLearning(Examples, Features)

- if Examples have the same class, return a leaf with this class
- else if Features is empty, return a leaf with the majority class
- else if Examples is empty, return a leaf with majority class of parent
- else

  find the best feature A to split (e.g. based on conditional entropy)

  Tree $\leftarrow$ a root with test on A

  For each value a of A:

  Child $\leftarrow$ **DecisionTreeLearning**(Examples with $A = a$, Features$\setminus\{A\}$)
  add **Child** to **Tree** as a new branch
- return **Tree**

Outline

Introduction

Boosting

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy

Introduction

Boosting

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)

Introduction

Boosting

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)
- works very well in practice (especially in combination with trees)

Introduction

Boosting

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)
- works very well in practice (especially in combination with trees)
- often is *resistant to overfitting*

Introduction

Boosting

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)
- works very well in practice (especially in combination with trees)
- often is *resistant to overfitting*
- has strong theoretical guarantees

Introduction

Boosting

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)
- works very well in practice (especially in combination with trees)
- often is *resistant to overfitting*
- has strong theoretical guarantees

We again focus on binary classification.

A simple example

Email spam detection:

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" $\Rightarrow$ spam

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" $\Rightarrow$ spam
- reweight the examples so that "difficult" ones get more attention
  - e.g. spam that doesn't contain the word "money"

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" $\Rightarrow$ spam
- reweight the examples so that "difficult" ones get more attention
  - e.g. spam that doesn't contain the word "money"
- obtain another classifier by applying the same base algorithm:
  - e.g. empty "to address" $\Rightarrow$ spam

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" $\Rightarrow$ spam
- reweight the examples so that "difficult" ones get more attention
  - e.g. spam that doesn't contain the word "money"
- obtain another classifier by applying the same base algorithm:
  - e.g. empty "to address" $\Rightarrow$ spam
- repeat ...

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" ⇒ spam
- reweight the examples so that "difficult" ones get more attention
  - e.g. spam that doesn't contain the word "money"
- obtain another classifier by applying the same base algorithm:
  - e.g. empty "to address" ⇒ spam
- repeat ...
- final classifier is the (weighted) majority vote of all weak classifiers

The base algorithm

A **base algorithm** $\mathcal{A}$ (also called weak learning algorithm/oracle) takes a training set S weighted by D as input, and outputs classifier $h \leftarrow \mathcal{A}(S, D)$

The base algorithm

A **base algorithm** $\mathcal{A}$ (also called weak learning algorithm/oracle) takes a training set S weighted by D as input, and outputs classifier $h \leftarrow \mathcal{A}(S, D)$

- this can be any off-the-shelf classification algorithm (e.g. decision trees, logistic regression, neural nets, etc)

The base algorithm

A **base algorithm** $\mathcal{A}$ (also called weak learning algorithm/oracle) takes a training set S weighted by D as input, and outputs classifier $h \leftarrow \mathcal{A}(S, D)$

- this can be any off-the-shelf classification algorithm (e.g. decision trees, logistic regression, neural nets, etc)
- many algorithms can deal with a weighted training set (e.g. for algorithm that minimizes some loss, we can simply replace "total loss" by "weighted total loss")

The base algorithm

A **base algorithm** $\mathcal{A}$ (also called weak learning algorithm/oracle) takes a training set S weighted by D as input, and outputs classifier $h \leftarrow \mathcal{A}(S, D)$

- this can be any off-the-shelf classification algorithm (e.g. decision trees, logistic regression, neural nets, etc)
- many algorithms can deal with a weighted training set (e.g. for algorithm that minimizes some loss, we can simply replace "total loss" by "weighted total loss")
- even if it's not obvious how to deal with weight directly, we can always resample according to D to create a new unweighted dataset

Boosting Algorithms

Given:

- a training set S
- a base algorithm $\mathcal{A}$

Boosting Algorithms

Given:

- a training set S
- a base algorithm $\mathcal{A}$

Two things to specify a boosting algorithm:

- how to **reweight** the examples?
- how to **combine** all the weak classifiers?

Boosting Algorithms

Given:

- a training set S
- a base algorithm $\mathcal{A}$

Two things to specify a boosting algorithm:

- how to **reweight** the examples?
- how to **combine** all the weak classifiers?

AdaBoost is one of the most successful boosting algorithms.

The AdaBoost Algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

The AdaBoost Algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

For $t = 1, \ldots, T$

- obtain a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$

The AdaBoost Algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

For $t = 1, \ldots, T$

- obtain a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$
- calculate the importance of h_t as

$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \qquad (\beta_t > 0 \Leftrightarrow \epsilon_t < 0.5)$$

where $\epsilon_t = \sum_{n: h_t(\boldsymbol{x}_n) \neq y_n} D_t(n)$ is the weighted error of h_t.

The AdaBoost Algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

For $t = 1, \ldots, T$

- obtain a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$
- calculate the importance of h_t as

$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \qquad (\beta_t > 0 \Leftrightarrow \epsilon_t < 0.5)$$

  where $\epsilon_t = \sum_{n: h_t(\boldsymbol{x}_n) \neq y_n} D_t(n)$ is the weighted error of h_t.

- update distributions

$$D_{t+1}(n) \propto D_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)} = \begin{cases} D_t(n) e^{-\beta_t} & \text{if } h_t(x_n) = y_n \\ D_t(n) e^{\beta_t} & \text{else} \end{cases}$$

The AdaBoost Algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

For $t = 1, \ldots, T$

- obtain a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$
- calculate the importance of h_t as

$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \qquad (\beta_t > 0 \Leftrightarrow \epsilon_t < 0.5)$$

  where $\epsilon_t = \sum_{n: h_t(\boldsymbol{x}_n) \neq y_n} D_t(n)$ is the weighted error of h_t.

- update distributions

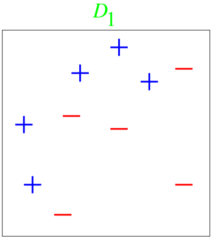
$$D_{t+1}(n) \propto D_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)} = \begin{cases} D_t(n) e^{-\beta_t} & \text{if } h_t(x_n) = y_n \\ D_t(n) e^{\beta_t} & \text{else} \end{cases}$$

Output the final classifier $H(\boldsymbol{x}) = \text{sgn}\left(\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x}) \right)$

Example

10 data points in $\mathbb{R}^2$

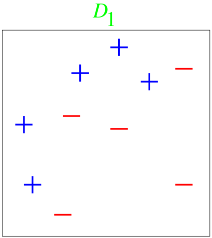
The size of $+$ or - indicates the weight, which starts from uniform D_1



Example

10 data points in $\mathbb{R}^2$

The size of + or - indicates the weight, which starts from uniform D_1

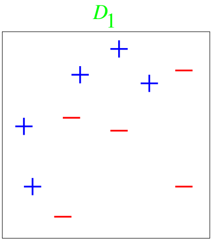


Base algorithm is decision stump:

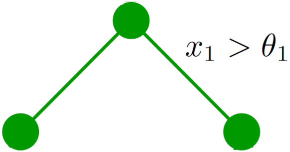
Example

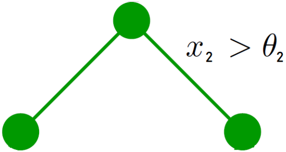
10 data points in $\mathbb{R}^2$

The size of $+$ or - indicates the weight, which starts from uniform D_1



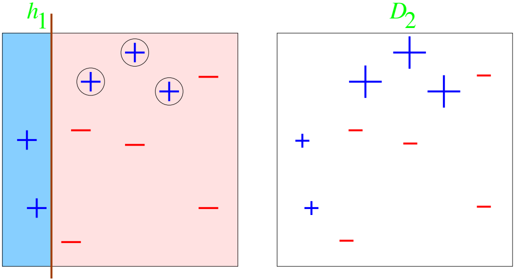
Base algorithm is decision stump:





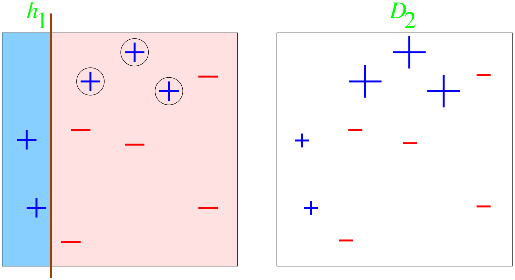
Observe that *no stump can predict very accurately for this dataset*

Round 1: $t = 1$



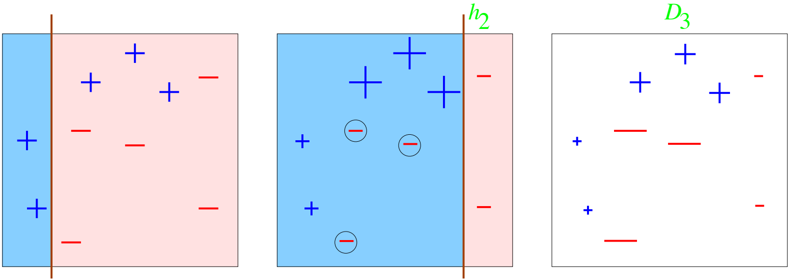
- 3 misclassified (circled): $\epsilon_1 = 0.3 \rightarrow \beta_1 = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right) \approx 0.42.$

Round 1: $t = 1$



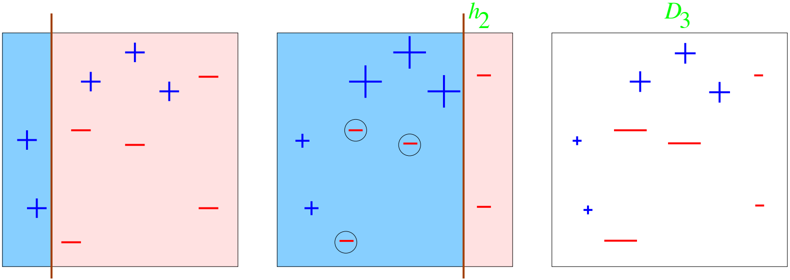
- 3 misclassified (circled): $\epsilon_1 = 0.3 \rightarrow \beta_1 = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \approx 0.42.$
- D_2 puts more weights on those examples

Round 2: $t = 2$



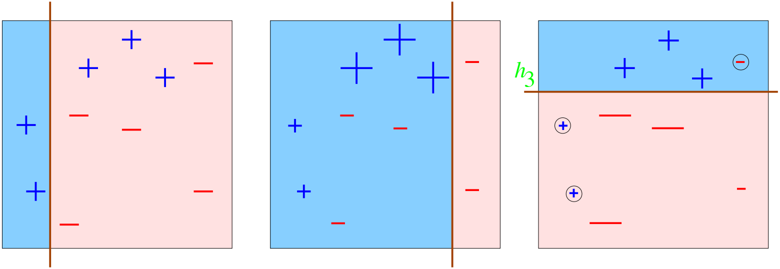
- 3 misclassified (circled): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.

Round 2: $t = 2$



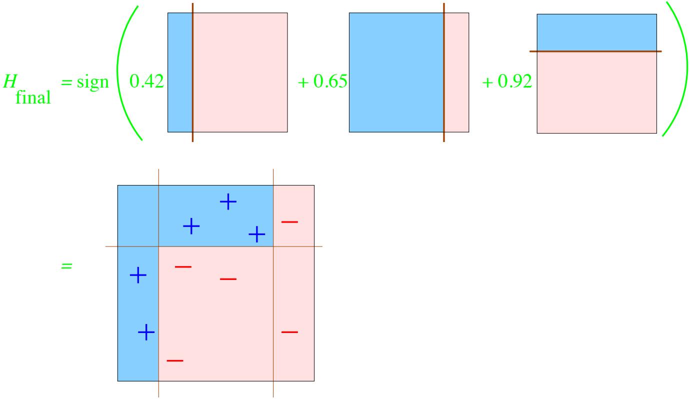
- 3 misclassified (circled): $\epsilon_2 = 0.21 \to \beta_2 = 0.65$.
- D_3 puts more weights on those examples

Round 3: $t = 3$

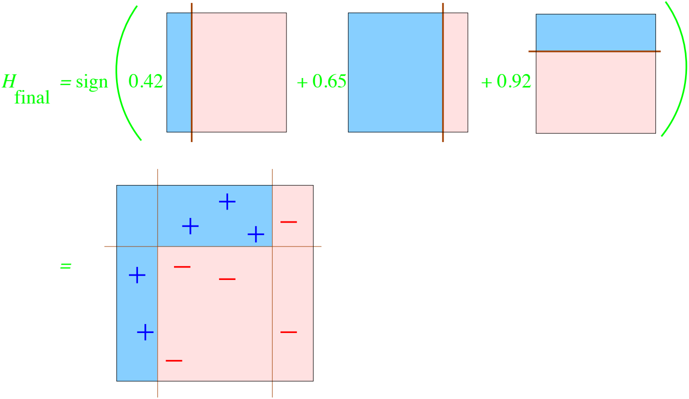


- again 3 misclassified (circled): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.

Final classifier: combining 3 classifiers



Final classifier: combining 3 classifiers



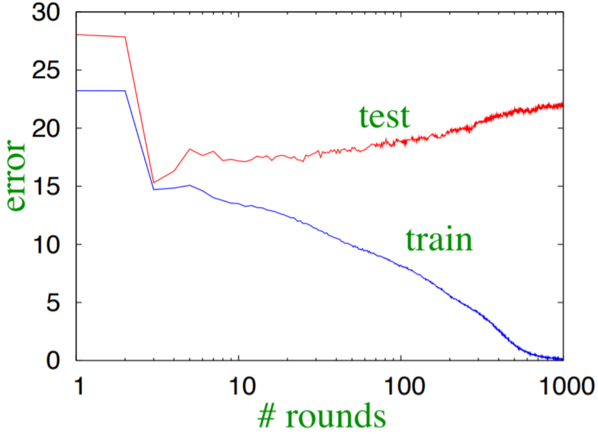
All data points are now classified correctly, even though each weak classifier makes 3 mistakes.

Overfitting

When T is large, the model is very complicated and overfitting can happen

Overfitting

When T is large, the model is very complicated and overfitting can happen



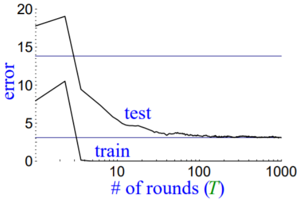
(boosting "stumps" on heart-disease dataset)

Resistance to overfitting

However, *very often AdaBoost is resistant to overfitting*

Resistance to overfitting

However, *very often AdaBoost is resistant to overfitting*



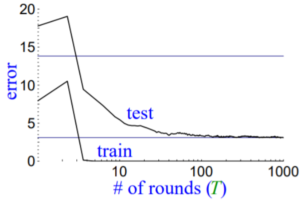
(boosting C4.5 on "letter" dataset)

- test error does not increase, even after 1000 rounds
  - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

Resistance to overfitting

However, *very often AdaBoost is resistant to overfitting*



(boosting C4.5 on "letter" dataset)

- test error does not increase, even after 1000 rounds
  - (total size $>$ 2,000,000 nodes)
- test error continues to drop even after training error is zero!

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

Used to be a mystery, but by now rigorous theory has been developed to explain this phenomenon.

Why AdaBoost works?

In fact, *AdaBoost also follows the general framework of minimizing some surrogate loss*.

Why AdaBoost works?

In fact, *AdaBoost also follows the general framework of minimizing some surrogate loss*.

Step 1: the model that AdaBoost considers is

$$\left\{ \text{sgn}\left(f(\cdot)\right) \;\middle|\; f(\cdot) = \sum_{t=1}^{T} \beta_t h_t(\cdot) \text{ for some } \beta_t \geq 0 \text{ and } h_t \in \mathcal{H} \right\}$$

where $\mathcal{H}$ is the set of models considered by the base algorithm

Why AdaBoost works?

In fact, *AdaBoost also follows the general framework of minimizing some surrogate loss*.

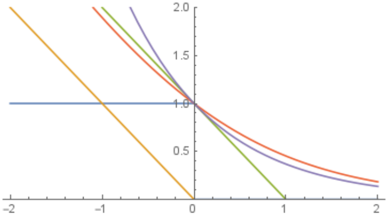
Step 1: the model that AdaBoost considers is

$$\left\{ \text{sgn}\left(f(\cdot)\right) \;\middle|\; f(\cdot) = \sum_{t=1}^{T} \beta_t h_t(\cdot) \text{ for some } \beta_t \geq 0 \text{ and } h_t \in \mathcal{H} \right\}$$

where $\mathcal{H}$ is the set of models considered by the base algorithm

Step 2: the loss that AdaBoost minimizes is the exponential loss

$$\sum_{n=1}^{N} \exp\left(-y_n f(\boldsymbol{x}_n)\right)$$



Greedy minimization

Step 3: the way that AdaBoost minimizes exponential loss is by a greedy approach, that is, find β_t, h_t one by one for $t = 1, \ldots, T$.

Greedy minimization

Step 3: the way that AdaBoost minimizes exponential loss is by a greedy approach, that is, find β_t, h_t one by one for $t = 1, \ldots, T$.

Specifically, let $f_t = \sum_{\tau=1}^{t} \beta_\tau h_\tau$. Suppose we have found f_{t-1}, *what should f_t be?*

Greedy minimization

Step 3: the way that AdaBoost minimizes exponential loss is by a greedy approach, that is, find β_t, h_t one by one for $t = 1, \ldots, T$.

Specifically, let $f_t = \sum_{\tau=1}^{t} \beta_\tau h_\tau$. Suppose we have found f_{t-1}, *what should f_t be?* Greedily, we want to find β_t, h_t to minimize

$$\sum_{n=1}^{\mathsf{N}} \exp\left(-y_n f_t(\boldsymbol{x}_n)\right)$$

Greedy minimization

Step 3: the way that AdaBoost minimizes exponential loss is by a greedy approach, that is, find β_t, h_t one by one for $t = 1, \ldots, T$.

Specifically, let $f_t = \sum_{\tau=1}^{t} \beta_\tau h_\tau$. Suppose we have found f_{t-1}, *what should f_t be?* Greedily, we want to find β_t, h_t to minimize

$$\sum_{n=1}^{\mathsf{N}} \exp\left(-y_n f_t(\boldsymbol{x}_n)\right) = \sum_{n=1}^{\mathsf{N}} \exp\left(-y_n f_{t-1}(\boldsymbol{x}_n)\right) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right)$$

Greedy minimization

Step 3: the way that AdaBoost minimizes exponential loss is by a greedy approach, that is, find β_t, h_t one by one for $t = 1, \ldots, T$.

Specifically, let $f_t = \sum_{\tau=1}^{t} \beta_\tau h_\tau$. Suppose we have found f_{t-1}, *what should f_t be?* Greedily, we want to find β_t, h_t to minimize

$$\sum_{n=1}^{\mathsf{N}} \exp\left(-y_n f_t(\boldsymbol{x}_n)\right) = \sum_{n=1}^{\mathsf{N}} \exp\left(-y_n f_{t-1}(\boldsymbol{x}_n)\right) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right)$$

$$\propto \sum_{n=1}^{\mathsf{N}} D_t(n) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right)$$

Greedy minimization

Step 3: the way that AdaBoost minimizes exponential loss is by a greedy approach, that is, find β_t, h_t one by one for $t = 1, \ldots, T$.

Specifically, let $f_t = \sum_{\tau=1}^{t} \beta_\tau h_\tau$. Suppose we have found f_{t-1}, *what should f_t be?* Greedily, we want to find β_t, h_t to minimize

$$\begin{aligned}\sum_{n=1}^{\mathsf{N}} \exp\left(-y_n f_t(\boldsymbol{x}_n)\right) &= \sum_{n=1}^{\mathsf{N}} \exp\left(-y_n f_{t-1}(\boldsymbol{x}_n)\right) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right) \\ &\propto \sum_{n=1}^{\mathsf{N}} D_t(n) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right)\end{aligned}$$

where the last step is by the definition of weights

$$D_t(n) \propto D_{t-1}(n) \exp\left(-y_n \beta_{t-1} h_{t-1}(\boldsymbol{x}_n)\right) \propto \cdots \propto \exp\left(-y_n f_{t-1}(\boldsymbol{x}_n)\right)$$

Greedy minimization

So the goal becomes finding $\beta_t, h_t \in \mathcal{H}$ that minimize

$$\sum_{n=1}^{\mathsf{N}} D_t(n) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right)$$

Greedy minimization

So the goal becomes finding $\beta_t, h_t \in \mathcal{H}$ that minimize

$$\sum_{n=1}^{N} D_t(n) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right)$$

$$= \sum_{n: y_n \neq h_t(\boldsymbol{x}_n)} D_t(n) e^{\beta_t} + \sum_{n: y_n = h_t(\boldsymbol{x}_n)} D_t(n) e^{-\beta_t}$$

Greedy minimization

So the goal becomes finding $\beta_t, h_t \in \mathcal{H}$ that minimize

$$\sum_{n=1}^{\mathsf{N}} D_t(n) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right)$$

$$= \sum_{n: y_n \neq h_t(\boldsymbol{x}_n)} D_t(n) e^{\beta_t} + \sum_{n: y_n = h_t(\boldsymbol{x}_n)} D_t(n) e^{-\beta_t}$$

$$= \epsilon_t e^{\beta_t} + (1 - \epsilon_t) e^{-\beta_t}$$

(recall $\epsilon_t = \sum_{n: y_n \neq h_t(\boldsymbol{x}_n)} D_t(n)$)

Greedy minimization

So the goal becomes finding $\beta_t, h_t \in \mathcal{H}$ that minimize

$$
\begin{aligned}
&\sum_{n=1}^{\mathsf{N}} D_t(n) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right) \\
&= \sum_{n: y_n \neq h_t(\boldsymbol{x}_n)} D_t(n) e^{\beta_t} + \sum_{n: y_n = h_t(\boldsymbol{x}_n)} D_t(n) e^{-\beta_t} \\
&= \epsilon_t e^{\beta_t} + (1-\epsilon_t) e^{-\beta_t} \qquad \text{(recall } \epsilon_t = \textstyle\sum_{n: y_n \neq h_t(\boldsymbol{x}_n)} D_t(n)\text{)} \\
&= \epsilon_t (e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t}
\end{aligned}
$$

Greedy minimization

So the goal becomes finding $\beta_t, h_t \in \mathcal{H}$ that minimize

$$\sum_{n=1}^{\mathsf{N}} D_t(n) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right)$$

$$= \sum_{n: y_n \neq h_t(\boldsymbol{x}_n)} D_t(n) e^{\beta_t} + \sum_{n: y_n = h_t(\boldsymbol{x}_n)} D_t(n) e^{-\beta_t}$$

$$= \epsilon_t e^{\beta_t} + (1 - \epsilon_t) e^{-\beta_t} \qquad \text{(recall } \epsilon_t = \textstyle\sum_{n: y_n \neq h_t(\boldsymbol{x}_n)} D_t(n)\text{)}$$

$$= \epsilon_t (e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t}$$

It is now clear we should find h_t to minimize the weighted classification error ϵ_t, *exactly what the base algorithm should do intuitively!*

Greedy minimization

So the goal becomes finding $\beta_t, h_t \in \mathcal{H}$ that minimize

$$\sum_{n=1}^{\mathsf{N}} D_t(n) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right)$$

$$= \sum_{n: y_n \neq h_t(\boldsymbol{x}_n)} D_t(n) e^{\beta_t} + \sum_{n: y_n = h_t(\boldsymbol{x}_n)} D_t(n) e^{-\beta_t}$$

$$= \epsilon_t e^{\beta_t} + (1 - \epsilon_t) e^{-\beta_t} \qquad \text{(recall } \epsilon_t = \textstyle\sum_{n: y_n \neq h_t(\boldsymbol{x}_n)} D_t(n))$$

$$= \epsilon_t (e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t}$$

It is now clear we should find h_t to minimize the weighted classification error ϵ_t, *exactly what the base algorithm should do intuitively!*

This greedy step is abstracted out through a base algorithm.

Greedy minimization

When h_t (and thus ϵ_t) is fixed, we then find β_t to minimize

$$\epsilon_t(e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t}$$

Greedy minimization

When h_t (and thus ϵ_t) is fixed, we then find β_t to minimize

$$\epsilon_t(e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t}$$

This gives the following (*verify!*):

$$\beta_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

Greedy minimization

When h_t (and thus ϵ_t) is fixed, we then find β_t to minimize

$$\epsilon_t(e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t}$$

This gives the following (*verify!*):

$$\beta_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

Keep doing this greedy minimization gives the AdaBoost algorithm.

Summary for boosting

Key idea of boosting is to **combine weak predictors into a strong one**.

Summary for boosting

Key idea of boosting is to **combine weak predictors into a strong one**.

There are many boosting algorithms; AdaBoost is the most classic one.

Summary for boosting

Key idea of boosting is to **combine weak predictors into a strong one**.

There are many boosting algorithms; AdaBoost is the most classic one.

AdaBoost is **greedily minimizing the exponential loss**.

Summary for boosting

Key idea of boosting is to **combine weak predictors into a strong one**.

There are many boosting algorithms; AdaBoost is the most classic one.

AdaBoost is **greedily minimizing the exponential loss**.

AdaBoost tends to **not overfit**.