Lecture 14: Boosting

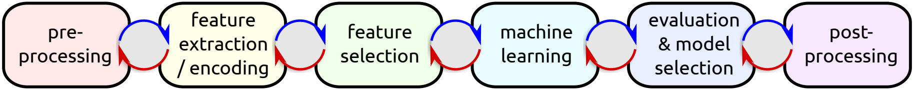
Machine Learning, Summer Term 2019

July 4, 2019

Michael Tangermann Frank Hutter Marius Lindauer

University of Freiburg

The Big Picture



- Lecture 1: overview
- Lecture 2-6: linear methods
- Lecture 7-9: algorithm-independent principles
- Lectures 10-15: nonlinear methods
  - Lecture 10-12: kernel-based methods
  - Lectures 13-14: tree-based methods and ensembles
  - Lecture 15: neural networks

Motivation for Boosting

Winning Methods of 10 Kaggle competitions in 2016 with non-image input (Competitions with images: deep neural networks dominate)

Almost all winners were large ensembles prominently using boosting:

Competition	Winners
Expedia Hotel Recommendations	1st: XGB
Santander Customer Satisfaction	3rd: XGB, RF, AdaBoost & others
Home Depot Product Search Relevance	1st and 3rd: XGB and RF
BNP Paribas Cardif Claims Management	1st and 2nd: XGB, RF & others
March Machine Learning Mania 2016	1st: RF and logarithmic regression
Telstra Network Disruptions	1st: sklearn, XGB, NN
Prudential Life Insurance Assessment	Top 3 used XGB (2nd and 3rd also others)
Airbnb New User Bookings	2nd: XGB; 3rd: XGB, NN, RF, ET
Homesite Quote Conversion	1st and 2nd: XGB & others; 3rd: others

XGB = Extreme Gradient Boosting
Places not listed: choice of method not released

Motivation for Boosting

- Similar to random forests, boosting has often been called the best off-the-shelf model (e.g., by Leo Breiman, inventor of random forests)

Motivation for Boosting

- Similar to random forests, boosting has often been called the best off-the-shelf model (e.g., by Leo Breiman, inventor of random forests)
- Similar advantages as random forests (we'll boost trees)
  - Trees: easy to interpret
  - Directly handle categorical features
  - Scalable to many data points (can be fast)
  - Scalable to many features (automated feature selectors)
  - Robust performance even for small datasets

Lecture Overview

1. Introduction to Boosting
2. AdaBoost
3. Gradient Boosting

Lecture Overview

1. Introduction to Boosting
2. AdaBoost
3. Gradient Boosting

Boosting combines weak learners

- A weak learner is a learning algorithm that does at least slightly better than random (e.g., strictly better than 50% error for binary classification)
  - E.g., email spam filter: occurence of 'buy now' would already let us classify better than random
  - Often, these simple rules already yield reasonable classification performance

Boosting combines weak learners

- A weak learner is a learning algorithm that does at least slightly better than random (e.g., strictly better than 50% error for binary classification)
  - E.g., email spam filter: occurence of 'buy now' would already let us classify better than random
  - Often, these simple rules already yield reasonable classification performance
- Boosting combines many weak learners into a highly accurate decision model.

Boosting combines weak learners

- A weak learner is a learning algorithm that does at least slightly better than random (e.g., strictly better than 50% error for binary classification)
  - E.g., email spam filter: occurence of 'buy now' would already let us classify better than random
  - Often, these simple rules already yield reasonable classification performance
- Boosting combines many weak learners into a highly accurate decision model.
- In this lecture, we use decision trees as our class of weak learners (even single-level trees, 'stumps', are sometimes used)

Boosting: Relationship to Bagging

- Like bagging, boosting is an committee / method that combines many weak learners (submodels) into a highly accurate decision model.

Boosting: Relationship to Bagging

- Like bagging, boosting is an committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels sequentially, with the m-th submodel trained to fix the mistakes of the first $m-1$ submodels

Boosting: Relationship to Bagging

- Like bagging, boosting is an committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels sequentially, with the m-th submodel trained to fix the mistakes of the first $m - 1$ submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging

Boosting: Relationship to Bagging

- Like bagging, boosting is an committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels sequentially, with the m-th submodel trained to fix the mistakes of the first $m-1$ submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally give different weights to its submodels

Boosting: Relationship to Bagging

- Like bagging, boosting is an committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels sequentially, with the m-th submodel trained to fix the mistakes of the first $m-1$ submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally give different weights to its submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging ✓

Boosting: Relationship to Bagging

- Like bagging, boosting is an committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels sequentially, with the m-th submodel trained to fix the mistakes of the first $m-1$ submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally give different weights to its submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally bootstrap samples and use random feature subsets

Boosting: Relationship to Bagging

- Like bagging, boosting is an committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels sequentially, with the m-th submodel trained to fix the mistakes of the first $m-1$ submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally give different weights to its submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally bootstrap samples and use random feature subsets
  - ★ This is exactly like bagging ✓
  - ★ This is different to bagging

Two approaches to boosting

Boosting: Submodels are trained sequentially, with the m-th submodel trained to fix the mistakes of the first $m-1$ submodels

In this lecture, we will consider two algorithms that implement boosting:

Two approaches to boosting

Boosting: Submodels are trained sequentially, with the m-th submodel trained to fix the mistakes of the first $m-1$ submodels

In this lecture, we will consider two algorithms that implement boosting:

- Adaboost, where mistakes are identified by weightings on more "difficult" data points
  - Each submodel also gets a different weight, based on how "good" it is

Two approaches to boosting

Boosting: Submodels are trained sequentially, with the m-th submodel trained to fix the mistakes of the first $m-1$ submodels

In this lecture, we will consider two algorithms that implement boosting:

- Adaboost, where mistakes are identified by weightings on more "difficult" data points
  - Each submodel also gets a different weight, based on how "good" it is
- Gradient Boosting, where mistakes are identified by the gradient of our loss function
  - Each submodel gets the same weight

Lecture Overview

1. Introduction to Boosting
2. AdaBoost
3. Gradient Boosting

AdaBoost [Freund and Schapire, 1995]

Basic idea: iteratively construct submodels G_m that fix previous errors

- We'll have a weight w_i for each data point i
  - w_i measures how hard data point x_i is to predict
  - Start with uniform weights $w_i^{(1)}$, adapt across iterations m: $w_i^{(m)}$
  - In each iteration m,
    exponentially increase weights of data points x_i misclassified by G_m:

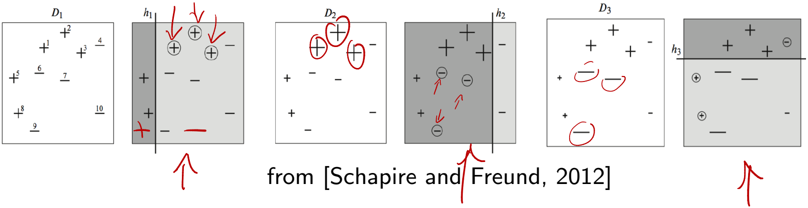


Figure from [Schapire and Freund, 2012]

AdaBoost [Freund and Schapire, 1995]

Basic idea: iteratively construct submodels G_m that fix previous errors

- We'll have a weight w_i for each data point i
  - w_i measures how hard data point x_i is to predict
  - Start with uniform weights $w_i^{(1)}$, adapt across iterations m: $w_i^{(m)}$
  - In each iteration m,
    exponentially increase weights of data points x_i misclassified by G_m:

$$w_i^{(m+1)} := \begin{cases} w_i^{(m)} \times \exp(\alpha_m) & , \quad \text{if } y_i \neq G_m(x_i) \\ w_i^{(m)} & , \quad \text{otherwise} \end{cases}$$

AdaBoost [Freund and Schapire, 1995]

Basic idea: iteratively construct submodels G_m that fix previous errors

- We'll have a weight w_i for each data point i
  - w_i measures how hard data point x_i is to predict
  - Start with uniform weights $w_i^{(1)}$, adapt across iterations m: $w_i^{(m)}$
  - In each iteration m,
    exponentially increase weights of data points x_i misclassified by G_m:

$$
\begin{aligned}
w_i^{(m+1)} &:= \begin{cases} w_i^{(m)} \times \exp(\alpha_m) & , \quad \text{if } y_i \neq G_m(x_i) \\ w_i^{(m)} & , \quad \text{otherwise} \end{cases} \\
&= w_i^{(m)} \exp(\underline{\alpha_m}\, \underline{\mathbb{I}(y_i \neq G_m(x_i))})
\end{aligned}
$$

AdaBoost [Freund and Schapire, 1995]

Basic idea: iteratively construct submodels G_m that fix previous errors

- We'll have a weight w_i for each data point i
  - w_i measures how hard data point x_i is to predict
  - Start with uniform weights $w_i^{(1)}$, adapt across iterations m: $w_i^{(m)}$
  - In each iteration m,
    exponentially increase weights of data points x_i misclassified by G_m:

$$
\begin{aligned}
w_i^{(m+1)} &:= \begin{cases} w_i^{(m)} \times \exp(\alpha_m) &, \quad \text{if } y_i \neq G_m(x_i) \\ w_i^{(m)} &, \quad \text{otherwise} \end{cases} \\
&= w_i^{(m)} \exp(\alpha_m \mathbb{I}(y_i \neq G_m(x_i)))
\end{aligned}
$$

- We'll also compute a weight α_m for each submodel G_m
  - This depends on how good the model is
  - We'll use these weights α_m for a weighted majority vote in the end

AdaBoost [Freund and Schapire, 1995]

- Binary classification problem; class labels $+1$ and -1

AdaBoost [Freund and Schapire, 1995]

- Binary classification problem; class labels $+1$ and -1
- Final model $G(x)$ combines individual submodels $G_1, \ldots, G_M$ through a weighted majority vote with weights $\alpha_1, \ldots, \alpha_M$:

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

AdaBoost [Freund and Schapire, 1995]

- Binary classification problem; class labels $+1$ and -1
- Final model $G(x)$ combines individual submodels $G_1, \ldots, G_M$ through a weighted majority vote with weights $\alpha_1, \ldots, \alpha_M$:

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

- Submodels G_m are weighted depending on their error err_m:

$$\alpha_m := \log \frac{1 - err_m}{err_m}$$

  - E.g., $err_m = 0.1 \rightarrow \alpha_m = 2.197$
  - E.g., $err_m = 0.4 \rightarrow \alpha_m = 0.41$
  - E.g., $err_m = 0.5 \rightarrow \alpha_m = 0$

AdaBoost [Freund and Schapire, 1995]

- Binary classification problem; class labels +1 and -1
- Final model $G(x)$ combines individual submodels $G_1, \ldots, G_M$ through a weighted majority vote with weights $\alpha_1, \ldots, \alpha_M$:

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

- Submodels G_m are weighted depending on their error err_m:

$$\alpha_m := \log \frac{1 - err_m}{err_m}$$

  - E.g., $err_m = 0.1 \rightarrow \alpha_m = 2.197$
  - E.g., $err_m = 0.4 \rightarrow \alpha_m = 0.41$
  - E.g., $err_m = 0.5 \rightarrow \alpha_m = 0$

  - This choice of α_m can be shown to minimize an upper bound of the final hypothesis error (see [Schapire, 2003] for details)

AdaBoost [Freund and Schapire, 1995]

- The (unweighted) training error rate of a submodel G_m is

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(y_i \neq G_m(x_i)).$$

AdaBoost [Freund and Schapire, 1995]

- The (unweighted) training error rate of a submodel G_m is

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(y_i \neq G_m(x_i)).$$

- The weighted training error rate of submodel G_m is

$$err_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

AdaBoost [Freund and Schapire, 1995]

- The (unweighted) training error rate of a submodel G_m is

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i \neq G_m(x_i)).$$

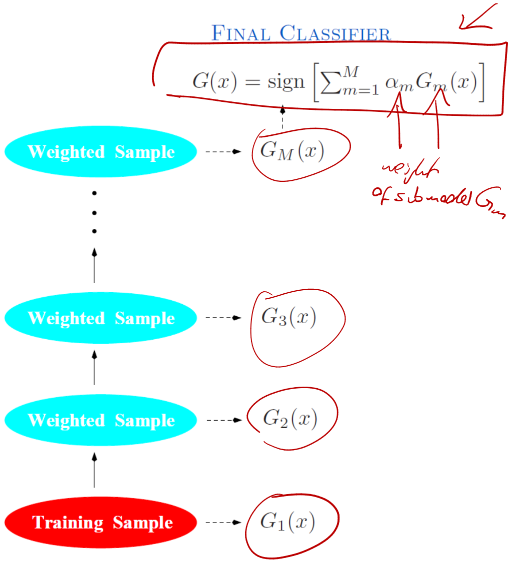
- The weighted training error rate of submodel G_m is

$$err_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

- As mentioned, these weighted training error rates are used for computing the model weights:

$$\alpha_m := \log \frac{1 - err_m}{err_m}$$

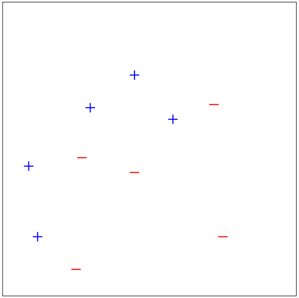
AdaBoost [Freund and Schapire, 1995]

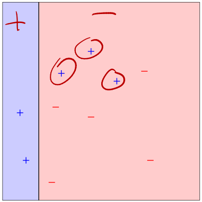


from [Hastie, Tibshirani and Friedman]

AdaBoost Example (step $m = 1$)

Example taken from [Schapire, 2003]
Model class: simple axis-aligned splits (decision stumps)

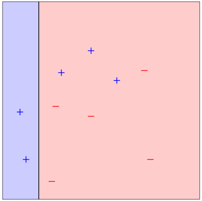




How large is the error err_1 of this first model?

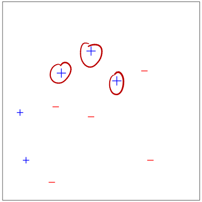
★ 0.3 ★ 0.2 ★ 0.5 ★ 0.7

AdaBoost Example (step $m = 1$)



How large is the error err_1 of this first model?

★ 0.3 ★ 0.2 ★ 0.5 ★ 0.7



AdaBoost Example (step $m = 1$ details)

- Model error and model weight:

$$err_1 = \sum_{i=1}^{N} w_i^{(1)} \, \mathbb{I}(G_1(x_i) \neq y_i) = \frac{1}{10} \times 3 = 0.3$$

$$\underline{\alpha_1} = \log \frac{1 - err_1}{err_1} = \underline{\log \frac{1 - 0.3}{0.3}} \approx \underline{0.847}$$

AdaBoost Example (step $m = 1$ details)

- Model error and model weight:

$$err_1 = \sum_{i=1}^{N} w_i^{(1)} \, \mathbb{I}(G_1(x_i) \neq y_i) = \frac{1}{10} \times 3 = 0.3$$

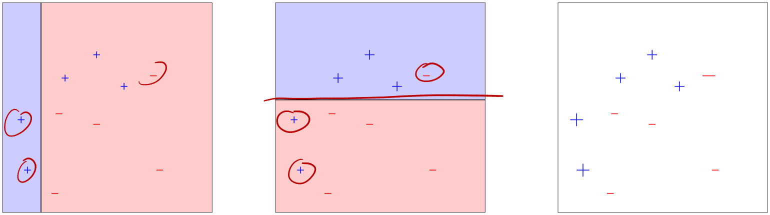
$$\alpha_1 = \log \frac{1 - err_1}{err_1} = \log \frac{1 - 0.3}{0.3} \approx 0.847$$

- Weight adaptation for data points:

$$w_i^{(m+1)} = w_i^{(m)} \exp(\alpha_m \, \mathbb{I}(y_i \neq G_m(x_i)))$$

  - Misclassified data point: $w_i^{(2)} \leftarrow w_i^{(1)} \exp(\alpha_1) \approx 0.1 \exp(0.847) \approx 0.233$
  - Correctly classified data points: $w_i^{(2)} \leftarrow w_i^{(1)} = 0.1$

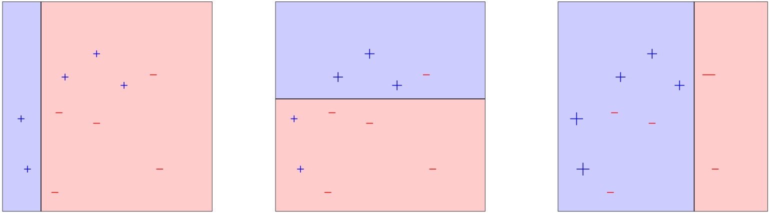
AdaBoost Example (step $m = 2$)



$$err_2 = \frac{\sum_{i=1}^{N} w_i^{(2)} \, \mathbb{I}(G_m(x_i) \neq y_i)}{\sum_{i=1}^{N} w_i^{(2)}} \approx \frac{0.1 + 0.1 + 0.1}{1.4} \approx 0.21$$

$$\alpha_2 = \log \frac{1 - err_2}{err_2} \approx \log \frac{1 - 0.21}{0.21} \approx 1.3$$

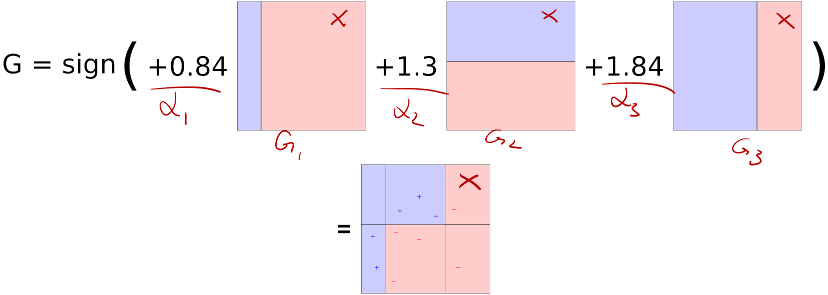
AdaBoost Example (step $m = 3$)



$err_3 \approx 0.14, \alpha_3 \approx 1.84$

AdaBoost Example

Final classifier:

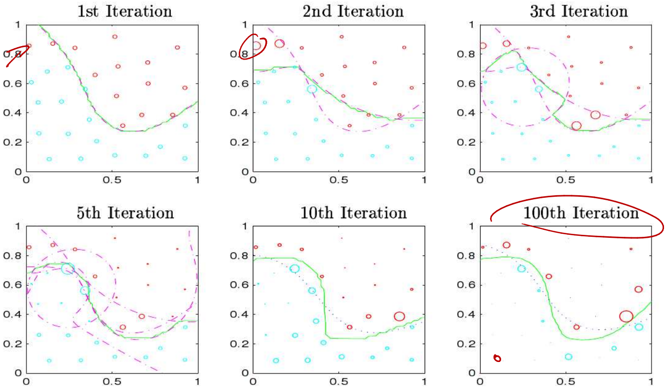


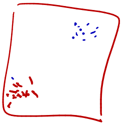
Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

from [Hastie, Tibshirani and Friedman]

AdaBoost: Focus on Hardest Data Points





Taken from [Meir, Raetsch, 2003]: AdaBoost on a 2D toy data set: color indicates the label, diameter is proportional to the weight of the example. Purple dashed lines: decision boundaries of the single classifiers (up to 5th iteration). Solid green line: decision boundary of the combined classifier. In the last two plots the decision line of Bagging is plotted for a comparison.

⇝ Main problem of AdaBoost: if the data is very noisy, it can overfit badly / it it very sensitive to outliers

Lecture Overview

1 Introduction to Boosting

2 AdaBoost

3 Gradient Boosting

Break: Teaching evaluations

- Teaching evaluations are running
  - Evaluations are our only formal reward
  - We redesigned much of the course, so feedback is very valuable

- Let's take a 10-minute break to do the evaluation right now

An alternative formulation of boosting

- With AdaBoost, we took the sign of a weighted majority vote of M submodels, for binary classification:

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

- Let's now generalise to a real-valued prediction, and remove the weights on each submodel:

$$G(x) = \sum_{m=1}^{M} G_m(x)$$

- Again, we will fit these M submodels sequentially.

An alternative formulation of boosting

Suppose that $M-1$ submodels have already been fitted, and our task is to fit the Mth submodel.

$$G(x) = \sum_{m=1}^{M-1} G_m(x) + G_M(x)$$

An alternative formulation of boosting

Suppose that $M-1$ submodels have already been fitted, and our task is to fit the Mth submodel.

$$G(x) \;=\; \sum_{m=1}^{M-1} G_m(x) + G_M(x)$$

As always, our objective is to minimise the loss over all data points:

$$\sum_{i=1}^{N} L(y_i, \boxed{\sum_{m=1}^{M-1} G_m(x_i) + \underline{G_M(x_i)}})$$

An alternative formulation of boosting

Suppose that $M-1$ submodels have already been fitted, and our task is to fit the Mth submodel.

$$y_i - \sum_{m=1}^{M-1} G_m(x)$$

$$G(x) = \sum_{m=1}^{M-1} G_m(x) + G_M(x)$$

As always, our objective is to minimise the loss over all data points:

$$\sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M-1} G_m(x_i) + G_M(x_i))$$

Question: What method have we seen for iteratively minimising the loss of a model's predictions?

★ PCA ★ ICA ★ Talking to domain experts ★ Gradient descent

Gradient Boosting: Idea

- We can use **any** differentiable loss function L, and find the gradient g_i of our model's predictions (with M submodels) with respect to our predictions for each data point i:

$$g_i = \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)}$$

Gradient Boosting: Idea

- We can use **any** differentiable loss function L, and find the gradient g_i of our model's predictions (with M submodels) with respect to our predictions for each data point i:

$$g_i = \frac{\partial L\left(y_i, G(x_i)\right)}{\partial G(x_i)}$$

- We then fit our new weak learner G_{M+1} with parameters θ_{M+1} using a gradient descent step $-\nu \cdot g$ as our target variable y.
  - ν is our gradient descent step size, or learning rate. In the context of gradient boosting, ν is also known as shrinkage.

$$\boxed{\theta_{M+1} = \arg\min_{\theta} \sum_{i=1}^{N} L(\overbrace{-\nu \cdot g_i}^{y_i}, G(x_i; \theta))}$$

Gradient Boosting Algorithm (Simplified)

1. Initialize G = best constant prediction for y.
2. For $m = 1$ to M
   (a) For $i = 1$ to N: $g_i = \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)}$
   (b) Fit weak learner G_m to data $\langle x_i, -\nu \cdot g_i \rangle_{i=1}^{N}$
   (c) Update $G(x) \leftarrow G(x) + G_m(x)$
3. Output final prediction $G(x)$

Gradient Boosting: Plugging in decision trees

1. Initialize $G =$ best constant prediction for y.
2. For $m = 1$ to M
   (a) For $i = 1$ to N: $g_i = \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)}$
   (b) Fit decision tree G_m to data $\langle x_i, -\nu \cdot g_i \rangle_{i=1}^{N}$
   (c) Update $G(x) \leftarrow G(x) + G_m(x)$
3. Output final prediction $G(x)$

Each of these lines is basically one line in Python (using a tree building library)

Gradient Boosting: Plugging in Squared Error Loss

- Plugging in squared error loss to our gradient function, we get a familiar result:

$$
\begin{aligned}
g_i &= \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)} \\
&= \frac{\partial \frac{1}{2}(y_i - G(x_i))^2}{\partial G(x_i)} \\
&= -(y_i - G(x_i))
\end{aligned}
$$

Gradient Boosting: Plugging in Squared Error Loss

- Plugging in squared error loss to our gradient function, we get a familiar result:

$$
\begin{aligned}
g_i &= \frac{\partial L\left(y_i, G(x_i)\right)}{\partial G(x_i)} \\
&= \frac{\partial \frac{1}{2}(y_i - G(x_i))^2}{\partial G(x_i)} \\
&= -(y_i - G(x_i))
\end{aligned}
$$

$\rightarrow$ Doing a step w.r.t. the negative gradient: Fit a submodel with the previous model's residuals $y_i - G(x_i)$ as targets.

- Review: why do we use the *negative* gradient?

Gradient Tree Boosting with L2 Error

1. Initialize $G =$ best constant prediction for y.
2. For $m = 1$ to M
   (a) For $i = 1$ to N: $r_i = y_i - G(x_i)$
   (b) Fit decision tree G_m to data $\langle x_i, -\nu \cdot r_i \rangle_{i=1}^{N}$
   (c) Update $G(x) \leftarrow G(x) + G_m(x)$
3. Output final prediction $G(x)$

- Computationally very simple!
- This is related to forward stagewise additive modelling, which fits an additive model by greedily fitting one component at a time

Gradient Boosting: Plugging in Absolute Error

- Plugging in absolute error loss to our gradient function, we get another effective algorithm:

$$\begin{aligned} g_i &= \frac{\partial L\left(y_i, G(x_i)\right)}{\partial G(x_i)} \\ &= \frac{\partial |y_i - G(x_i)|}{\partial G(x_i)} \\ &= \underline{\text{sign}(y_i - G(x_i))} \end{aligned}$$

Gradient Boosting: Plugging in Absolute Error

- Plugging in absolute error loss to our gradient function, we get another effective algorithm:

$$
\begin{aligned}
g_i &= \frac{\partial L\left(y_i, G(x_i)\right)}{\partial G(x_i)} \\
&= \frac{\partial |y_i - G(x_i)|}{\partial G(x_i)} \\
&= \underline{\text{sign}}(y_i - G(x_i))
\end{aligned}
$$

- Review: when would you expect this to perform better than squared error loss? 

The link between Gradient Boosting and Adaboost

- What if we plug in the exponential loss function $L(y_i, G(x_i)) = \exp(-y_i \times G(x_i))$?

The link between Gradient Boosting and Adaboost

- What if we plug in the exponential loss function $L(y_i, G(x_i)) = \exp(-y_i \times G(x_i))$?
- It turns out that we recover the Adaboost algorithm!
  - The optimal G_m minimizes weighted error:

$$G_m = \arg\min_G \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}(y_i \neq G(x_i))$$

  - Reintroduce a model weight α_m as in Adaboost:

$$\alpha_m = \log \frac{1 - err_m}{err_m}$$

  - The weight update rule is as in Adaboost:

$$w_i^{(m+1)} \propto w_i^{(m)} \cdot \exp(\alpha_m \mathbb{I}(y_i \neq G_m(x_i)))$$

  - (More info on ILIAS; these details will not be examinable)

Regularisation for Tree Complexity

- Recall our formal definition of a tree $G(x_i; \theta)$, where:

$$\theta = \langle \underbrace{\{R_1, \ldots, R_J\}}_{\text{leaf regions}}, \underbrace{\{w_1, \ldots, w_J\}}_{\text{leaf scores}} \rangle$$

Regularisation for Tree Complexity

- Recall our formal definition of a tree $G(x_i; \theta)$, where:

$$\theta = \langle \underbrace{\{R_1, \ldots, R_J\}}_{\text{leaf regions}}, \underbrace{\{w_1, \ldots, w_J\}}_{\text{leaf scores}} \rangle$$

- We can impose a complexity penalty Ω on each submodel G_m
  - hyperparameters γ and λ penalise tree complexity J and leaf score magnitude $\|\mathbf{w}_m\|$ respectively:

$$\Omega(G_m) = \gamma J_m + \frac{1}{2}\lambda \left\|\mathbf{w}_m\right\|_2^2$$

Regularisation for Tree Complexity

- Recall our formal definition of a tree $G(x_i;\theta)$, where:

$$\theta = \langle \underbrace{\{R_1, \ldots, R_J\}}_{\text{leaf regions}}, \underbrace{\{w_1, \ldots, w_J\}}_{\text{leaf scores}} \rangle$$

- We can impose a complexity penalty Ω on each submodel G_m
  - hyperparameters γ and λ penalise tree complexity J and leaf score magnitude $\|\mathbf{w}_m\|$ respectively:

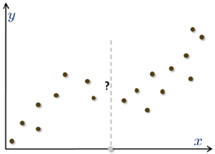
$$\Omega(G_m) = \gamma J_m + \frac{1}{2}\lambda \left\|\mathbf{w}_m\right\|_2^2$$

- Our method for fitting each submodel then becomes:

$$\theta_m = \arg\min_{\theta} \left[\sum_{i=1}^{N} L(-\nu \cdot g_i, G(x_i;\theta)) + \Omega(G_m) \right]$$

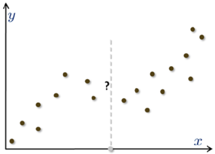
Regularisation for Tree Complexity

- Suppose we have a prior belief that:
  - there should be few divisions along the axes of our input space . . .
  - . . . but that these divisions should separate highly different samples



Regularisation for Tree Complexity

- Suppose we have a prior belief that:
  - there should be few divisions along the axes of our input space ...
  - ... but that these divisions should separate highly different samples



- What would be an appropriate hyperparameter setting?

$$\Omega(G_m) = \gamma J_m + \frac{1}{2}\lambda \|\mathbf{w}_m\|_2^2$$

★ High γ, High λ ★ High γ, Low λ
★ Low γ, High λ ★ Low γ, Low λ

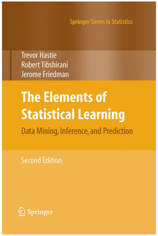
Summary by learning goals

Having heard this lecture, you can now . . .

- explain the principles of boosting, and how it differs from bagging
- describe the steps of the AdaBoost algorithm
- describe the steps of the gradient boosting approach
- Explain the relationship between gradient boosting, gradient descent, and Adaboost
- Describe the role of shrinkage and regularisation in gradient boosting

Further Reading

- Main source: Hastie, Tibshirani and Friedman
  - Chapter 10: Boosting and Additive Trees



- Wikipedia article on boosting
- Specialized literature
  - Schapire: A boosting tutorial
    `www.cs.princeton.edu/~schapire`
  - Meir and Raetsch, 2003: An introduction to boosting
  - Raetsch: Tutorial at MLSS 2003

Preview of Assignment 9

In the 9^{th} assignment, you will

- Implement the AdaBoost algorithm
- Implement the gradient boosting algorithm