Machine Learning & Pattern Recognition

SONG Xuemeng

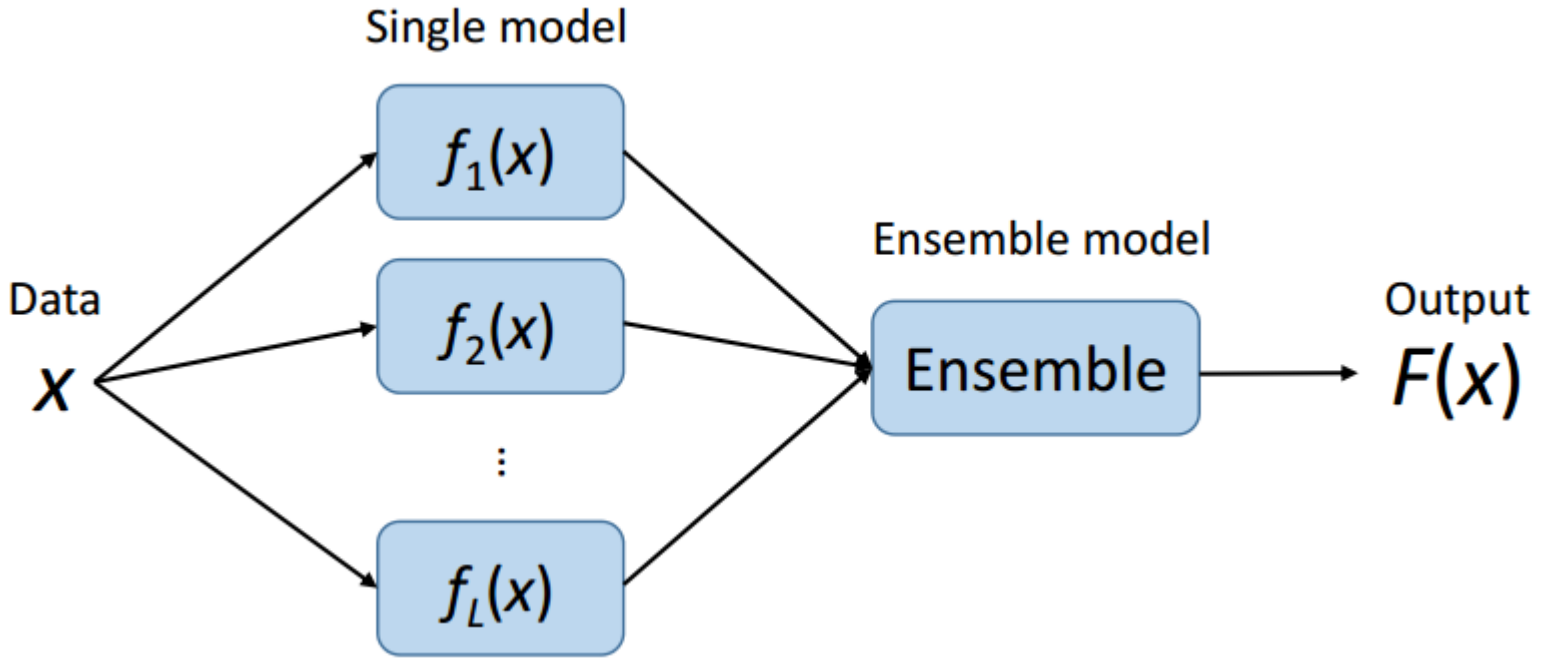
songxuemeng@sdu.edu.cn

http://xuemeng.bitcron.com/

Ensemble Learning

Aggregation of predictions of multiple (*Heterogenous/Homogeneous*) classifiers with the goal of improving accuracy.

IDEA: Consider a set of predictors, $f_1, \ldots, f_L$. Construct a $F(x)$ (*called committee*) that combines the individual decisions of $f_1, \ldots, f_L$.



Ensemble Learning

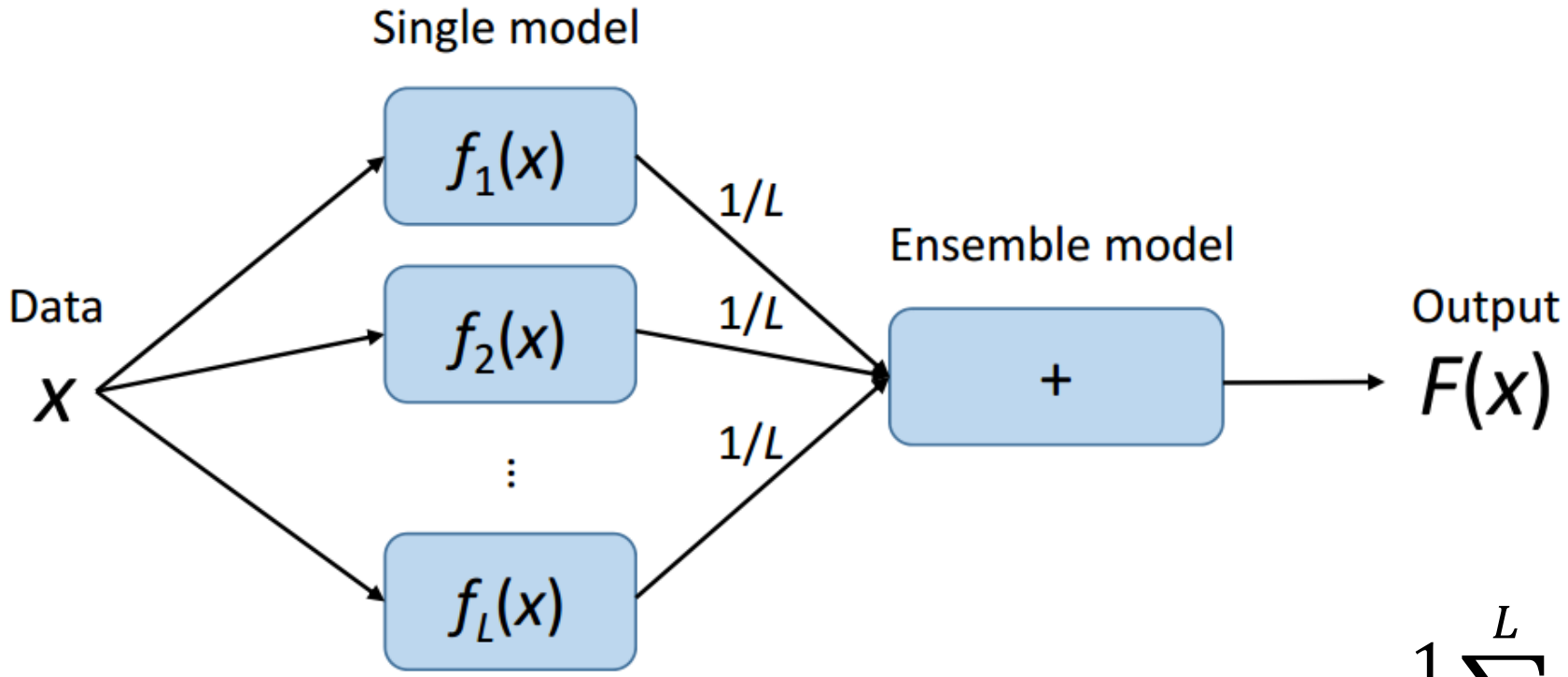
- **Heterogenous (Hybrid) Ensemble:**
  - A heterogeneous ensemble is an ensemble with a set of base classifiers that consist of models created using different algorithms.
  - E.g. The outputs of decision tree classifiers could be combined with the outputs of SVM.

Ensemble Learning

- **Heterogenous (Hybrid) Ensemble:**
  - A heterogeneous ensemble is an ensemble with a set of base classifiers that consist of models created using different algorithms.
  - E.g. The outputs of decision tree classifiers could be combined with the outputs of SVM.

- **Homogeneous Ensemble:**
  - Homogeneous ensemble refers to the fact that all of the base classifiers are of a single type (e.g., decision trees), differing by model parameters, the data used for training, or a combination of the two.
  - E.g., bagging and boosting.

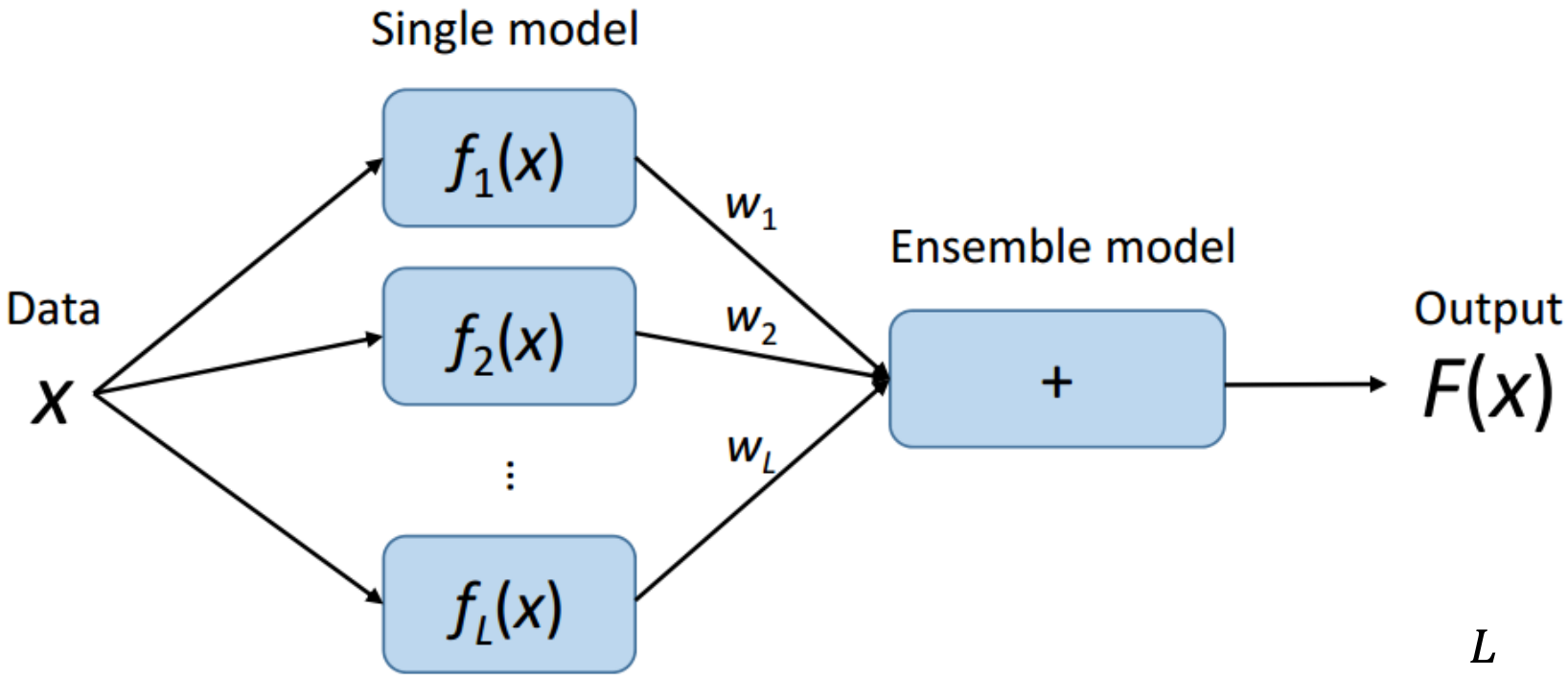
Combining Predictors: Averaging



$$F(\boldsymbol{x}) = \frac{1}{L}\sum_{i=1}^{L} f_i(\boldsymbol{x})$$

- Averaging for regression; voting for classification.

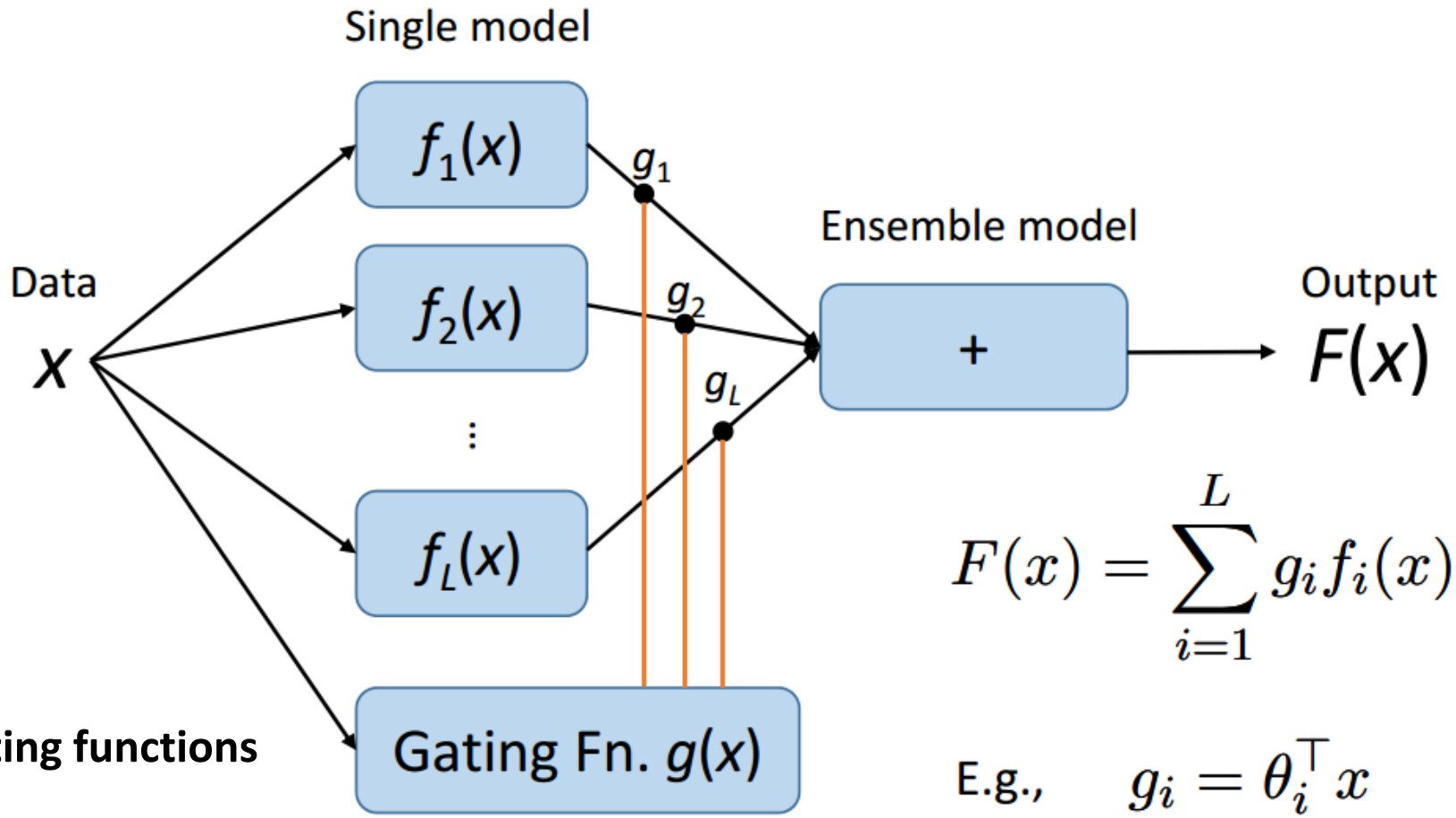
Combining Predictors: Weighted Averaging



$$F(\boldsymbol{x}) = \sum_{i=1}^{L} w_i f_i(\boldsymbol{x})$$

- Just like linear regression or classification
- Note: single model will not be updated when training ensemble model

Combining Predictors: Gating



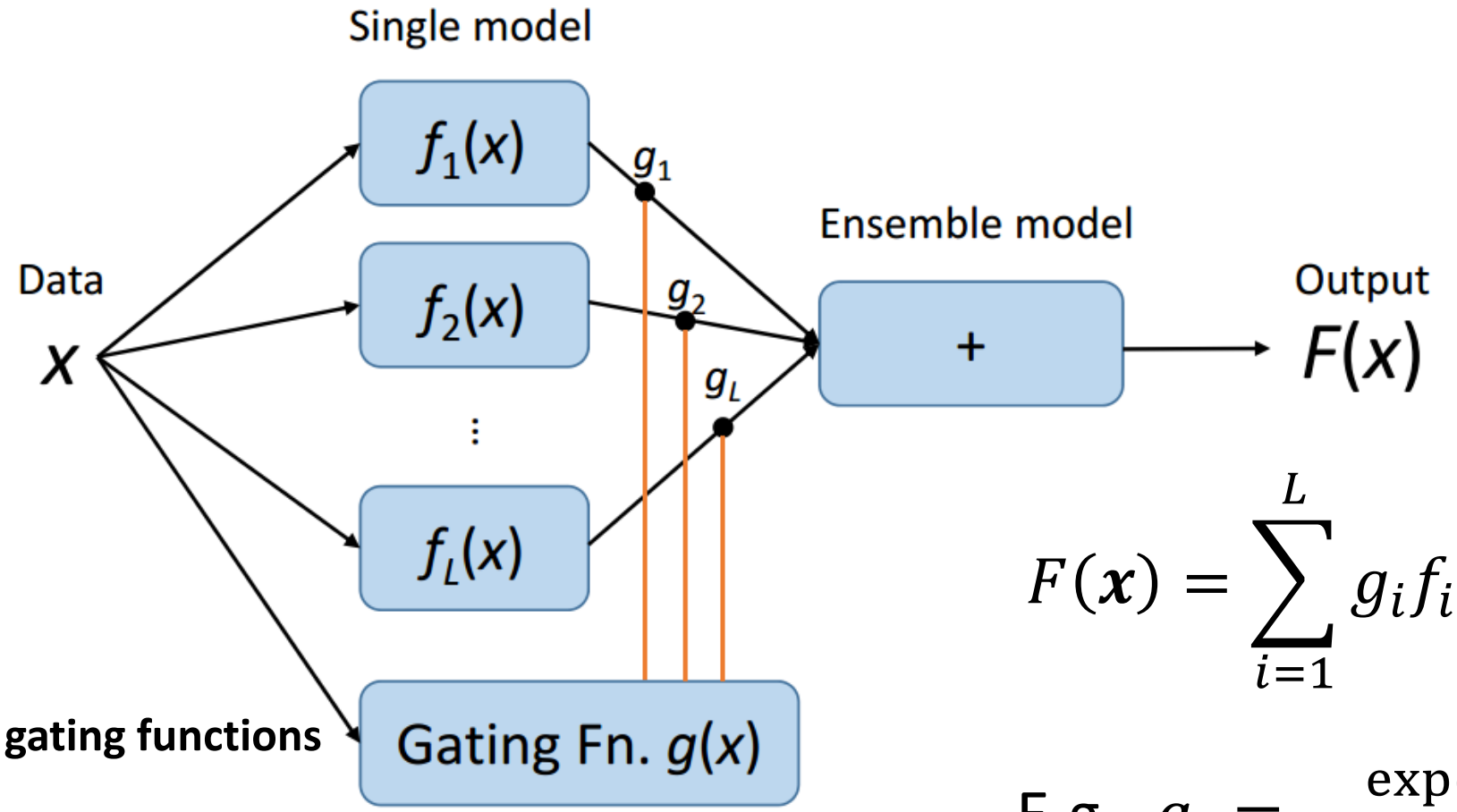
$$F(x) = \sum_{i=1}^{L} g_i f_i(x)$$

Learnable gating functions

E.g., $g_i = \theta_i^\top x$

- Just like linear regression or classification
- Note: single model will not be updated when training ensemble model

Combining Predictors: Gating



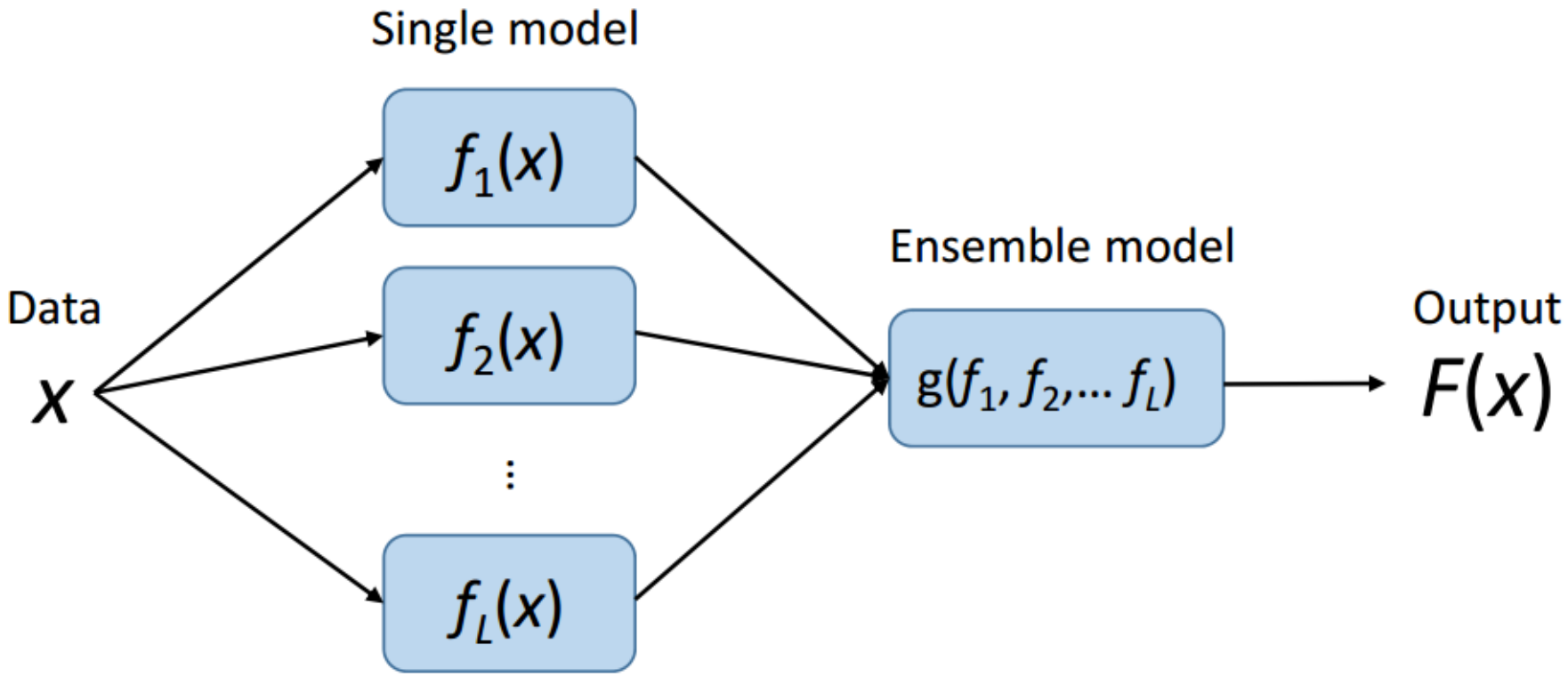
Learnable gating functions

$$F(\boldsymbol{x}) = \sum_{i=1}^{L} g_i f_i(\boldsymbol{x})$$

E.g., $g_i = \frac{\exp(\boldsymbol{\theta_i}^T \boldsymbol{x})}{\sum_{j=1}^{L} \exp(\boldsymbol{\theta_j}^T \boldsymbol{x})}$

- Just like linear regression or classification
- Note: single model will not be updated when training ensemble model

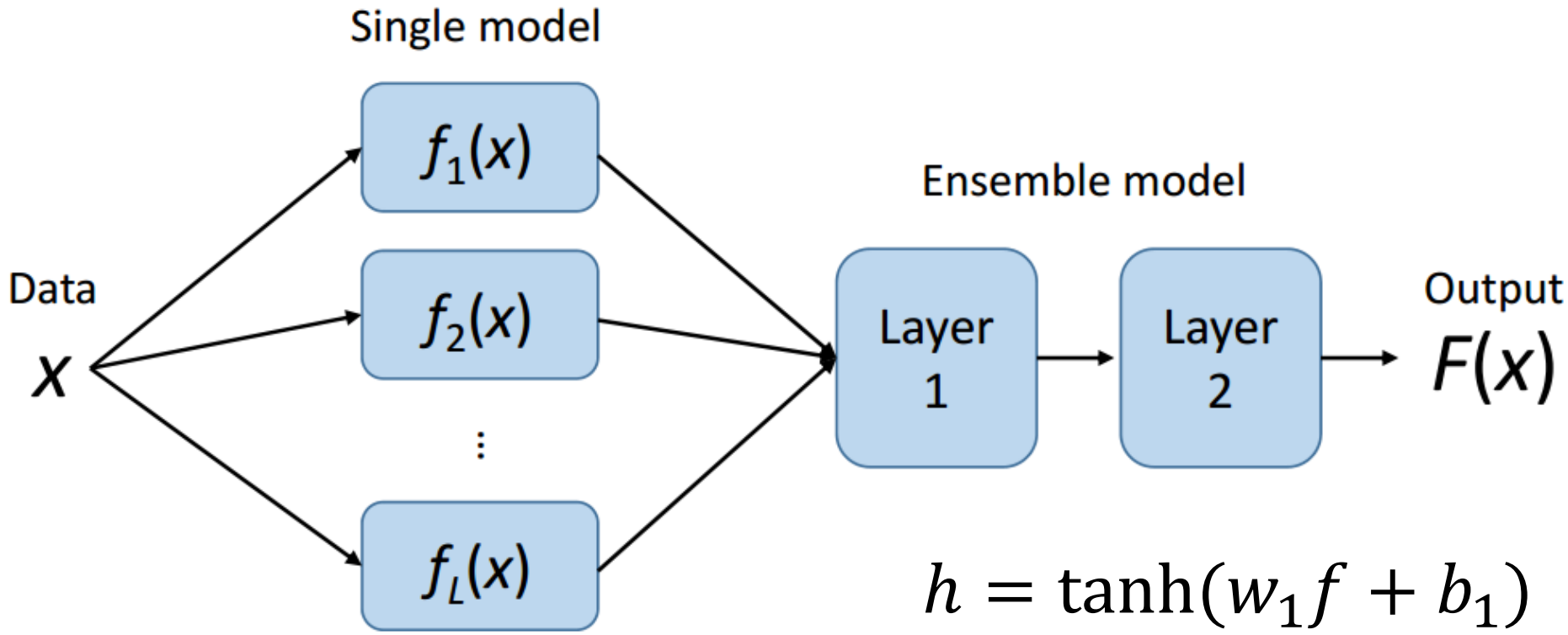
Combining Predictors: Stacking



$$F(x) = g(f_1(x), f_2(x), \ldots, f_L(x))$$

- This is the general formulation of an ensemble.

Combining Predictors: Multi-Layer

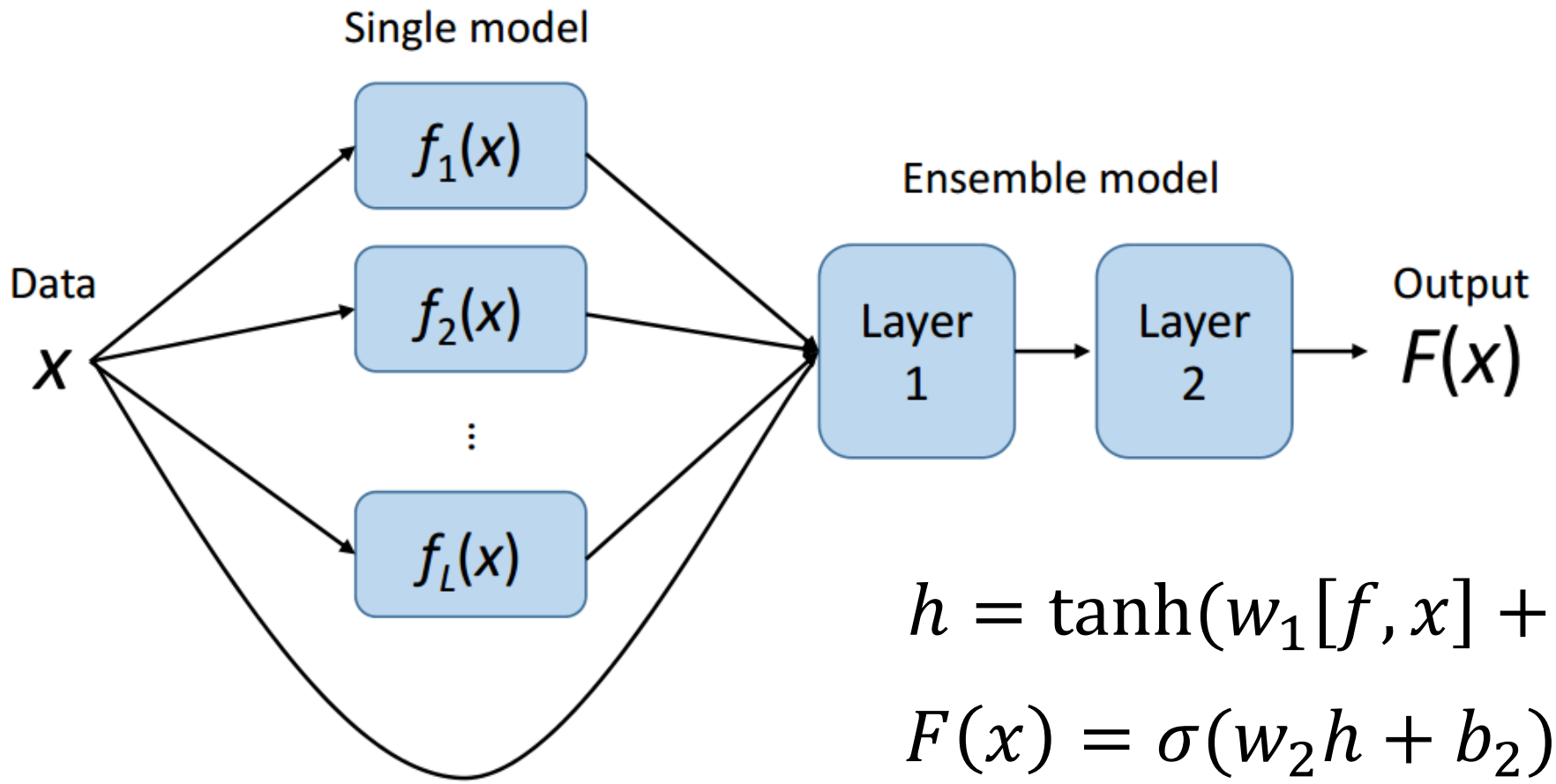


$$h = \tanh(w_1 f + b_1)$$

$$F(x) = \sigma(w_2 h + b_2)$$

- Use the neural network as the ensemble model.

Combining Predictors: Multi-Layer

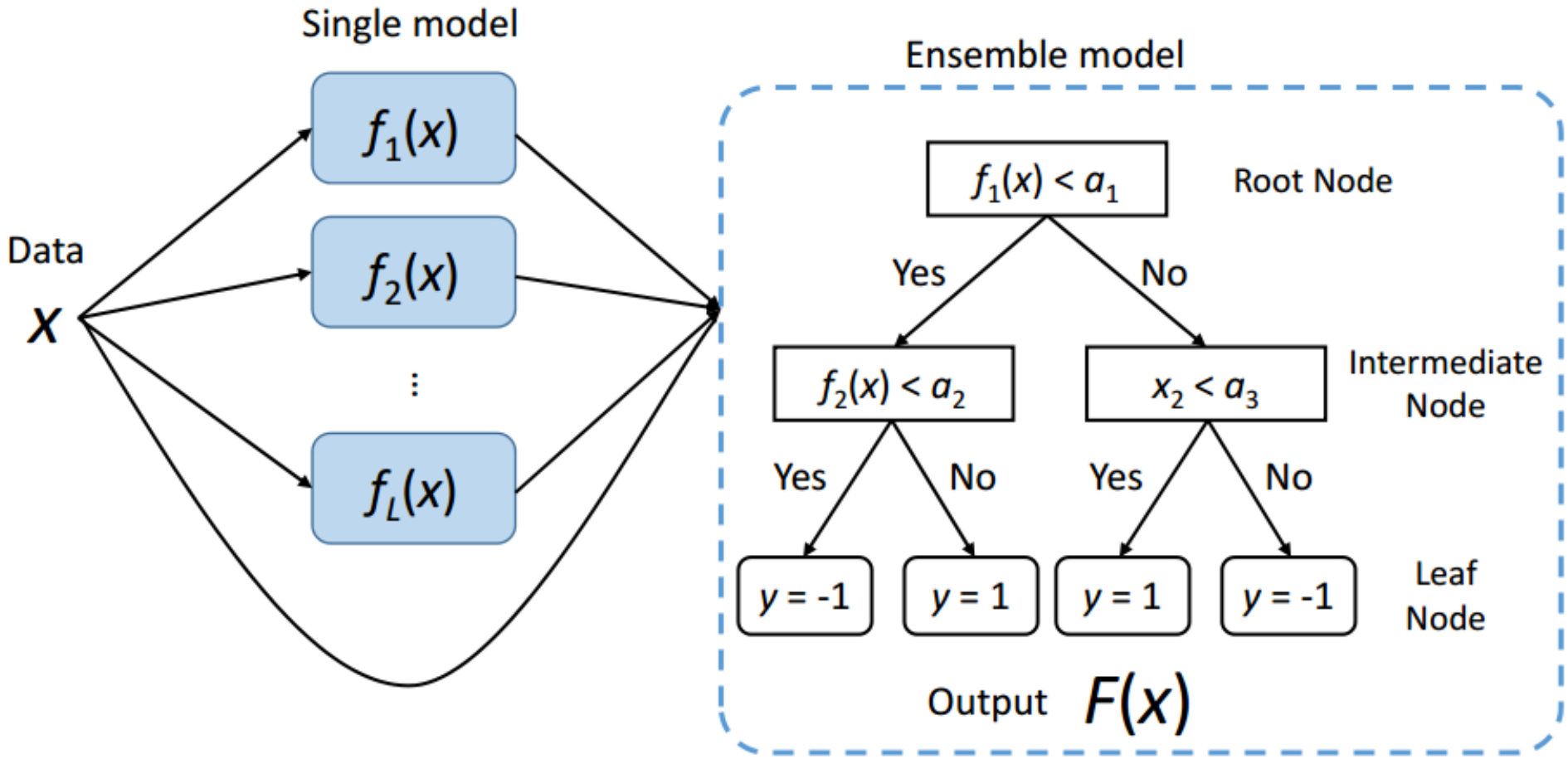


$$h = \tanh(w_1[f, x] + b_1)$$

$$F(x) = \sigma(w_2 h + b_2)$$

- Use the neural network as the ensemble model.
- Incorporate $\boldsymbol{x}$ into the first hidden layer (as gating).

Combining Predictors: Tree Models



- Use the decision trees as the ensemble model.
- Splitting according to the value of f's and $\boldsymbol{x}$.

Some Intuitions on Why Ensemble Methods Work…

Intuitions

Utility of combining diverse, independent opinions in human decision-making.

Majority Vote

Suppose we have 5 completely independent classifiers

- If accuracy is 70% for each
  - ✓ How about the majority vote accuracy?

Intuitions

Utility of combining diverse, independent opinions in human decision-making.

Majority Vote

Suppose we have 5 completely independent classifiers

- If accuracy is 70% for each
  - ✓ $10 \times (0.7^3)\,(0.3^2) + 5 \times (0.7^4)(0.3) + \ (0.7^5)$
  - ✓ **83.7% majority vote accuracy**

Intuitions

Utility of combining diverse, independent opinions in human decision-making.

Majority Vote

Suppose we have 5 completely independent classifiers

- If accuracy is 70% for each
  - ✓ $10 \times (0.7^3)\,(0.3^2) + 5 \times (0.7^4)(0.3) + \ (0.7^5)$
  - ✓ **83.7% majority vote accuracy**
- 101 such classifiers
  - ✓ **99.9% majority vote accuracy**

How good are ensemble methods?

How good are ensemble methods?

Let's look at some Competitions…

Netflix Prize Competition

Supervised Learning Task:

- Training data is a set of users and ratings 1, 2, 3, 4, 5 stars) those users have given to movies.
- Construct a classifier that given a user and an unrated movie, correctly classifies that movie as either 1, 2, 3, 4, or 5 stars

$1 million prize for a 10% improvement over Netflix's current movie recommender/classifier (MSE = 0.9514)

Netflix Prize Competition

Leaderboard

Display top 20 leaders.

An ensemble of more than 800 predictors

Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	The Ensemble	0.8553	10.10	2009-07-26 18:38:22
2	BellKor's Pragmatic Chaos	0.8554	10.09	2009-07-26 18:18:28
	Grand Prize - RMSE <= 0.8563			
3	Grand Prize Team	0.8571	9.91	2009-07-24 13:07:49
4	Opera Solutions and Vandelay United	0.8573	9.89	2009-07-25 20:05:52
5	Vandelay Industries !	0.8579	9.83	2009-07-26 02:49:53
6	PragmaticTheory	0.8582	9.80	2009-07-12 15:09:53
7	BellKor in BigChaos	0.8590	9.71	2009-07-26 12:57:25
8	Dace	0.8603	9.58	2009-07-24 17:18:43
9	Opera Solutions	0.8611	9.49	2009-07-26 18:02:08
10	BellKor	0.8612	9.48	2009-07-26 17:19:11
11	BigChaos	0.8613	9.47	2009-06-23 23:06:52
12	Feeds2	0.8613	9.47	2009-07-24 20:06:46
	Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos			
13	xiangliang	0.8633	9.26	2009-07-21 02:04:40
14	Gravity	0.8634	9.25	2009-07-26 15:58:34
15	Ces	0.8642	9.17	2009-07-25 17:42:38
16	Invisible Ideas	0.8644	9.14	2009-07-20 03:26:12
17	Just a guy in a garage	0.8650	9.08	2009-07-22 14:10:42
18	Craig Carmichael	0.8656	9.02	2009-07-25 16:00:54
19	J Dennis Su	0.8658	9.00	2009-03-11 09:41:54
20	acmehill	0.8659	8.99	2009-04-16 06:29:35
	Progress Prize 2007 - RMSE = 0.8712 - Winning Team: KorBell			
	Cinematch score on quiz subset - RMSE = 0.9514			

KDD-Cup 2011 Yahoo!Music Recommendation

Supervised Learning Task:

- Training data is the Yahoo! Music Dataset
- Predict the user's rating on a music, given some users' ratings on some music
  - With music information (e.g., album, artist, genre IDs)

Winner

From a graduate course of National Taiwan University—an ensemble of 221 predictors.

A Linear Ensemble of Individual and Blended Models for Music Rating Prediction

Po-Lung Chen, Chen-Tse Tsai, Yao-Nan Chen, Ku-Chun Chou, Chun-Liang Li, Cheng-Hao Tsai, Kuan-Wei Wu, Yu-Cheng Chou, Chung-Yi Li, Wei-Shih Lin, Shu-Hao Yu, Rong-Bing Chiu, Chieh-Yen Lin, Chien-Chih Wang, Po-Wei Wang, Wei-Lun Su, Chen-Hung Wu, Tsung-Ting Kuo, Todd G. McKenzie, Ya-Hsuan Chang, Chun-Sung Ferng, Chia-Mau Ni, Hsuan-Tien Lin, Chih-Jen Lin, Shou-De Lin

{R99922038, R98922028, R99922008, R99922095, B97018, B97705004, B96018, B96115, B96069, B96113, B95076, B97114, B97042, D98922007, B97058, B96110, B96055, D97944007, D97041, B96025, R99922054, B96092, HTLIN, CJLIN, SDLIN}@CSIE.NTU.EDU.TW

Department of Computer Science and Information Engineering,
National Taiwan University

Machine learning contests are usually won by methods using model averaging over dozens of models.

Can Any Ensemble Boost the Performance?

Can Any Ensemble Boost the Performance?

Different Ensemble

	Test 1	Test 2	Test 3
h_1	√	√	×
h_2	×	√	√
h_3	√	×	√
Ensemble	√	√	√

(a) Ensemble: Positive Effect.

Different Ensemble

	Test 1	Test 2	Test 3
h_1	√	√	×
h_2	×	√	√
h_3	√	×	√
Ensemble	√	√	√

(a) Ensemble: Positive Effect.

	Test 1	Test 2	Test 3
h_1	√	√	×
h_2	√	√	×
h_3	√	√	×
Ensemble	√	√	×

(b) Ensemble: No Effect.

Different Ensemble

	Test 1	Test 2	Test 3
h_1	√	√	×
h_2	×	√	√
h_3	√	×	√
Ensemble	√	√	√

(a) Ensemble: Positive Effect.

	Test 1	Test 2	Test 3
h_1	√	√	×
h_2	√	√	×
h_3	√	√	×
Ensemble	√	√	×

(b) Ensemble: No Effect.

	Test 1	Test 2	Test 3
h_1	√	×	×
h_2	×	√	×
h_3	×	×	√
Ensemble	×	×	×

(c) Ensemble: Negative Effect.

Diversity for Ensemble Input

- **Successful ensembles require diversity**
  - Predictors may make different mistakes
  - Encourage to
    - Involve different types of predictors
    - Vary the training sets
    - Vary the feature sets

Cause of the Mistake	Diversification Strategy
Pattern was difficult	Try different models
Overfitting	Vary the training sets
Some features are noisy	Vary the set of input features

Ensemble Methods

- Bagging
- Random Forest
- Boosting (Adaboost)
- Gradient Boosting Decision Trees

Ensemble Methods

- Bagging
- Random Forest
- Boosting (Adaboost)
- Gradient Boosting Decision Trees

Bagging: Bootstrap Aggregating

- A technique to reduce generalization error by combining several models.
- **Idea**: train several different models separately, then have all of the models vote on the output for test examples.

Bagging: Bootstrap Aggregating

- A technique to reduce generalization error by combining several models.
- **Idea**: train several different models separately, then have all of the models vote on the output for test examples.

- **Underlying philosophy:** different models will usually not make all the same errors on the test set.
- Allows the **same** kind of model/training algorithm/objective function to be reused several times.

Bagging

- Leo Breiman (1994)

Leo Breiman (1928–2005)

- Born in New York City, 1928
- Graduated from *Roosevelt High School*, 1945.
- Degree in physics from the *California Institute of Technology (Caltech)*, 1949.

Leo Breiman (1928–2005)

- Born in New York City, 1928
- Graduated from *Roosevelt High School*, 1945.
- Degree in physics from the *California Institute of Technology (Caltech)*, 1949.

Leo Breiman (1928–2005)

- Born in New York City, 1928
- Graduated from *Roosevelt High School*, 1945.
- Degree in physics from the *California Institute of Technology (Caltech)*, 1949.
- Master degree in mathematics from *Columbia University*, 1950.
  - In fact, he wants to learn philosophy..
- Ph.D. from *UC Berkeley*, 1954.
- Then hired to teach probability theory at *UCLA* (earned tenure 1960-1967)
- Writing books (Probability Theory)
- Spend the next 13 years as a consultant (1969-1982).

Leo Breiman (1928–2005)

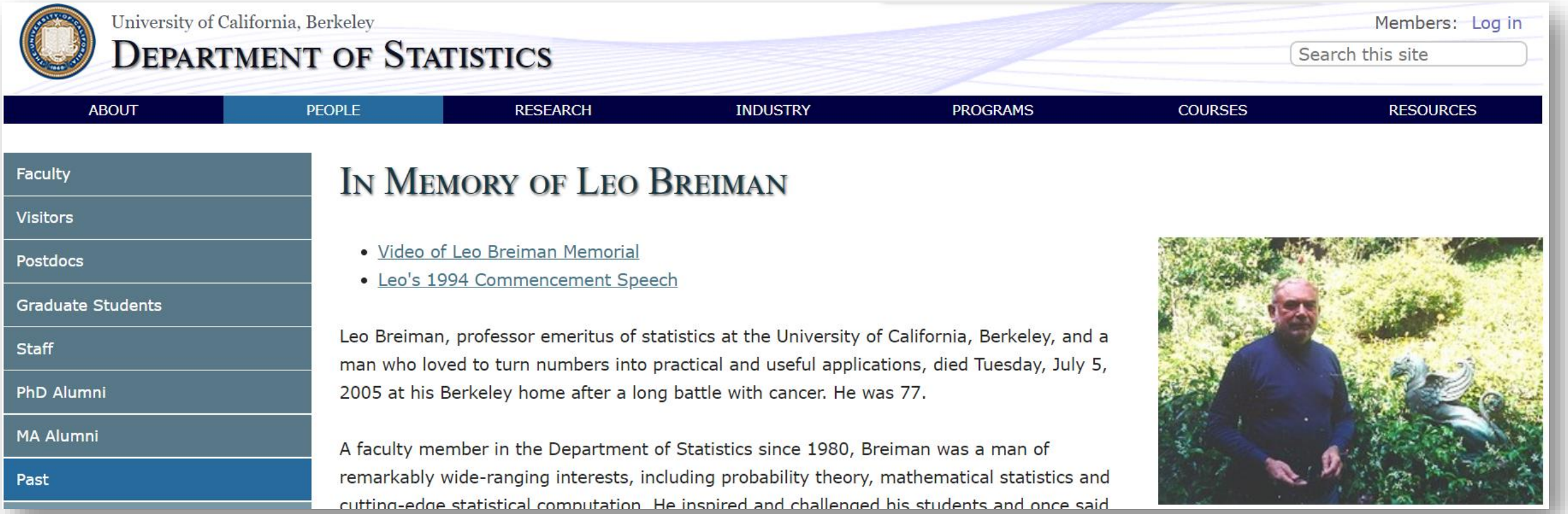
- Statistician at the University of California, Berkeley (1980-1993)
- A member of the *United States National Academy of Science*.
- Important Contributions
  - CART [1984]
  - Bagging [1994]
  - Random Forest [2001]

Leo Breiman (1928–2005)

- Statistician at the University of California, Berkeley (1980-1993)
- A member of the *United States National Academy of Science*.
- Important Contributions
  - CART [1984]
  - Bagging [1994]
  - Random Forest [2001]

Some of Breiman's best work was done after retirement...

Leo Breiman (1928–2005)



http://statistics.berkeley.edu/memory/leo-breiman

The Annals of Applied Statistics
2010, Vol. 4, No. 4, 1621–1633
DOI: 10.1214/10-AOAS427

REMEMBERING LEO BREIMAN[1]

BY ADELE CUTLER

Utah State University

Leo Breiman was a highly creative, influential researcher with a down-to-earth personal style and an insistence on working on important real world problems and producing useful solutions. This paper is a short review of Breiman's extensive contributions to the field of applied statistics.

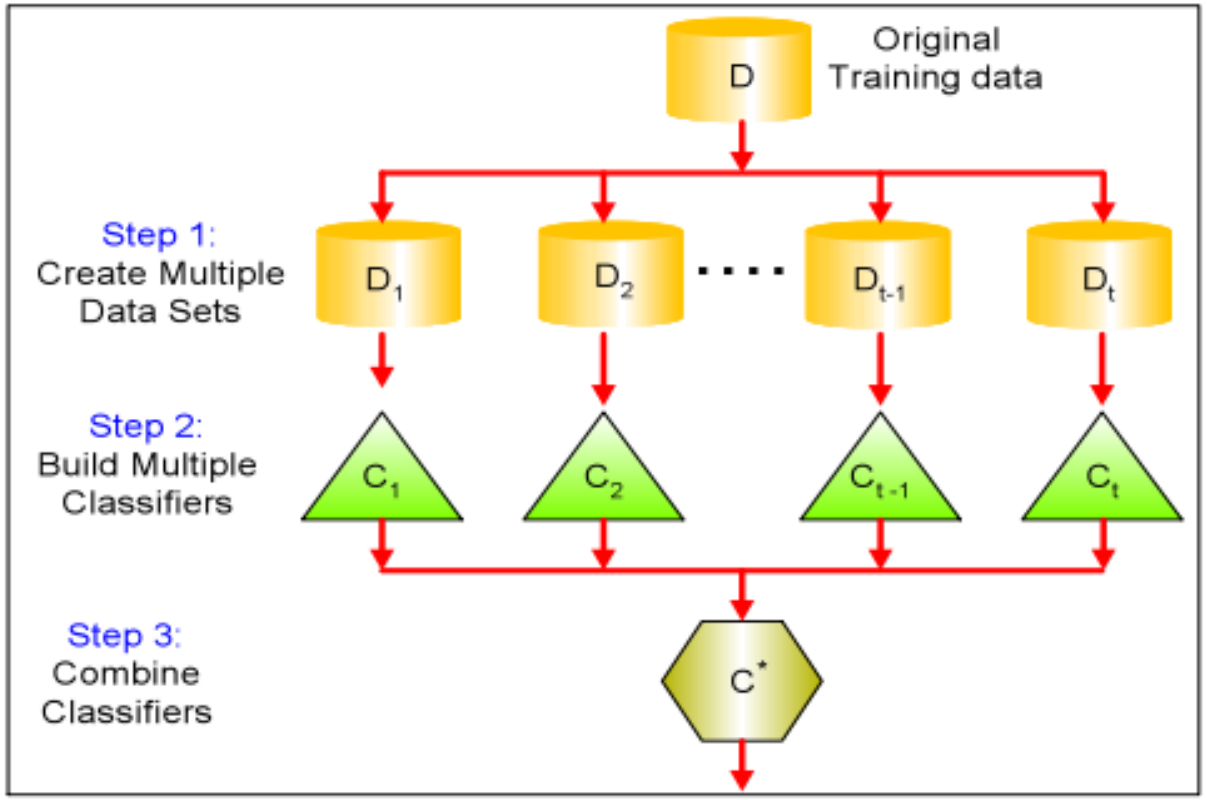
1. Introduction. How many theoretical probabilists walk away from a tenured faculty position at a top university and set out to make their living as consultants? How many applied consultants get hired into senior faculty positions in first-rate research universities? How many professors with a fine reputation in their field, establish an equally fine reputation in a *different* field, *after* retirement? Leo Breiman did all of these things and more. He

https://www.researchgate.net/publication/48178196_Remembering_Leo_Breiman#pfb

Bagging

- Leo Breiman (1994)
- Take repeated bootstrap samples from training set D.
- ***Bootstrap sampling***: Given set D containing N training examples, create D' by drawing N examples at random with replacement from D.

- Bagging:
  - Create k bootstrap samples $D_1,\ldots,D_k$.
  - Train distinct classifier on each D_i.
  - Classify new instance by majority vote.



Bagging

- Sampling with replacement

Training Data

Data ID

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- A particular training data has a probability of $1 - 1/N$ not being picked
- Thus its probability of ending up in the test data (not selected) is:

$$(1 - 1/N)^N \approx e^{-1} = 0.368$$

- This means the training data will contain approximately 63.2% of the instances

Bagging

- Bagging involves constructing k different datasets.
- Each dataset has the same number of examples as the original dataset
- Each dataset is constructed by sampling with replacement from the original dataset.

Bagging

- Bagging involves constructing k different datasets.
- Each dataset has the same number of examples as the original dataset.
- Each dataset is constructed by sampling with replacement from the original dataset.
  - With high probability, each dataset is missing some of the examples from the original dataset.
  - Each dataset also contains several duplicate examples.
  - On average around 2/3 of the examples from the original dataset can be found in the resulting training set.

Bagging

- Bagging involves constructing k different datasets.
- Each dataset has the same number of examples as the original dataset.
- Each dataset is constructed by sampling with replacement from the original dataset.
  - With high probability, each dataset is missing some of the examples from the original dataset.
  - Each dataset also contains several duplicate examples.
  - On average around 2/3 of the examples from the original dataset can be found in the resulting training set.

This works best when perturbing the training set can cause significant changes in the estimated model.

Bagging

Why Does Bagging Work?

Bagging

- Suppose we train an ‘8’ detector on the dataset, containing an ‘8’, a ‘6’ and a ‘9’.
- Suppose we make two different resampled datasets by sampling with replacement.
  - The first dataset omits the ‘9’ and repeats the ‘8’.
    -
  - The second dataset omits the ‘6’ and repeats the ‘9’.
    -

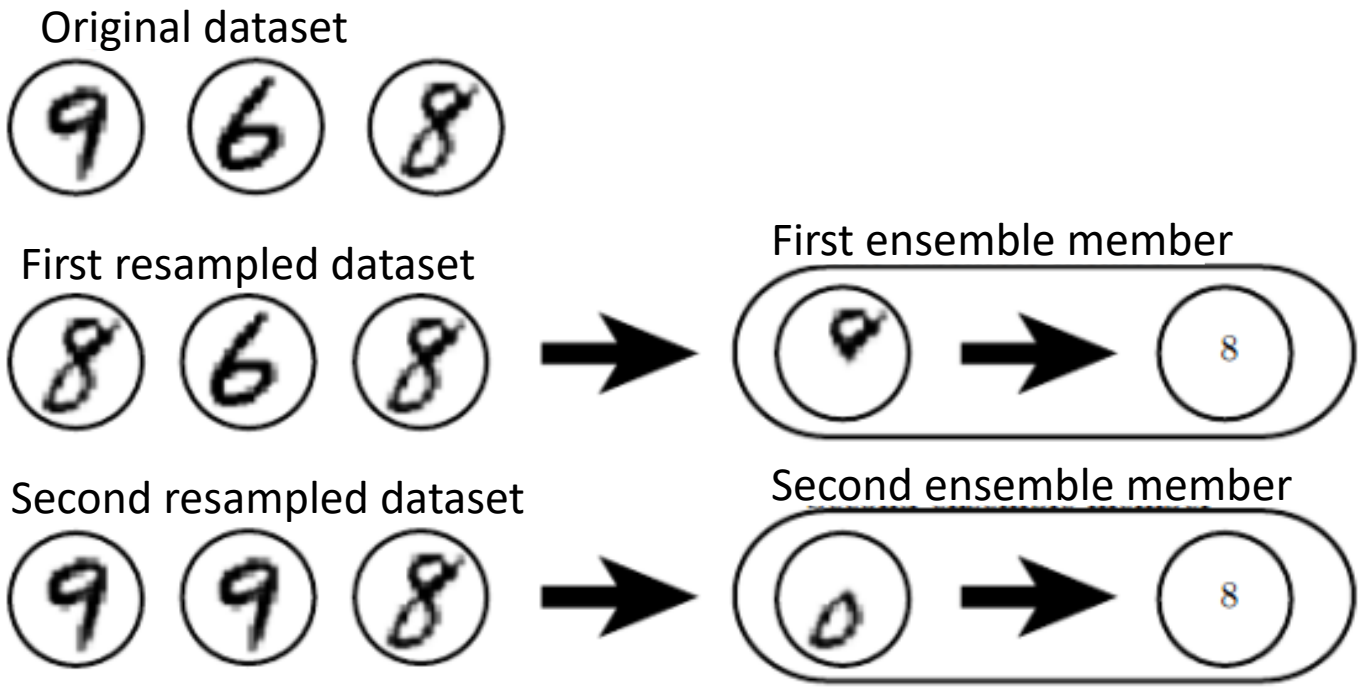
Original dataset

First resampled dataset

Second resampled dataset

Bagging

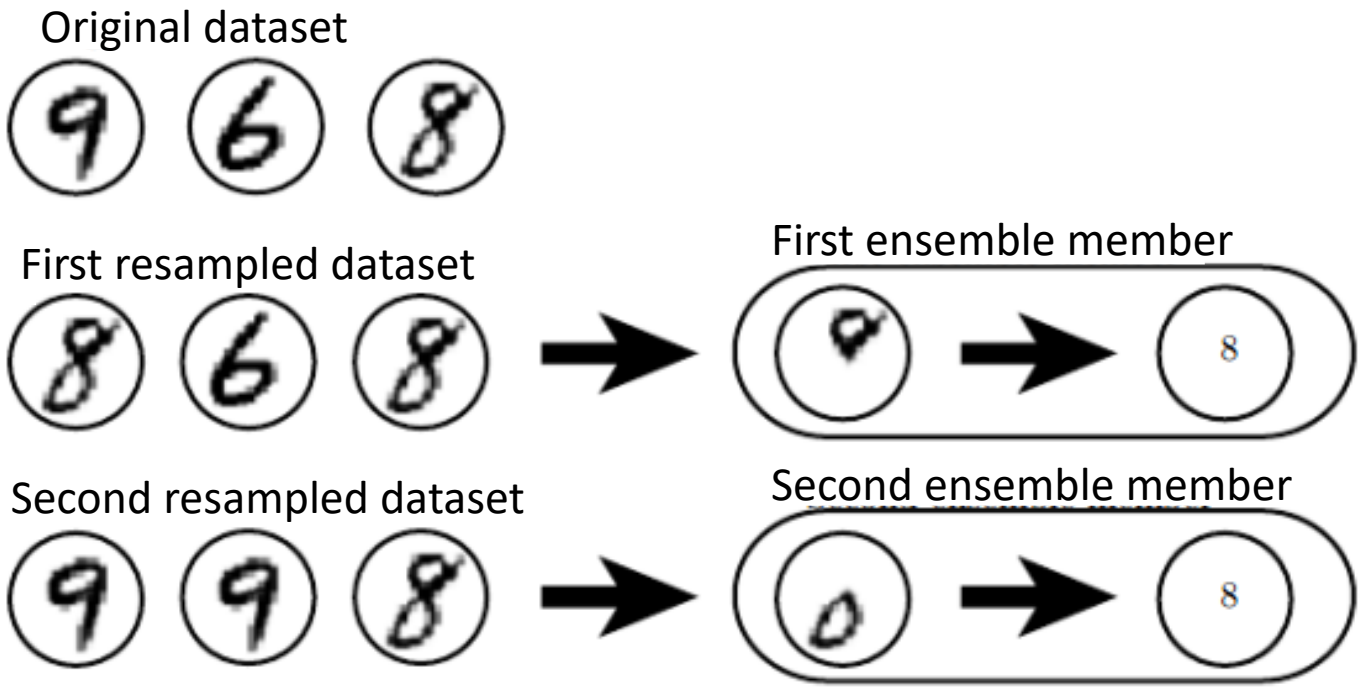
- Suppose we train an ‘8’ detector on the dataset, containing an ‘8’, a ‘6’ and a ‘9’.
- Suppose we make two different resampled datasets by sampling with replacement.
  - The first dataset omits the ‘9’ and repeats the ‘8’.
    - The detector learns that **a loop on top** of the digit corresponds to an ‘8’.
  - The second dataset omits the ‘6’ and repeats the ‘9’.
    - The detector learns that **a loop on the bottom** of the digit corresponds to an ‘8’.



Bagging

- Suppose we train an '8' detector on the dataset, containing an '8', a '6' and a '9'.
- Suppose we make two different resampled datasets by sampling with replacement.
  - The first dataset omits the '9' and repeats the '8'.
    - The detector learns that **a loop on top** of the digit corresponds to an '8'.
  - The second dataset omits the '6' and repeats the '9'.
    - The detector learns that **a loop on the bottom** of the digit corresponds to an '8'.

- Each of these individual classification rules is brittle.
- But if we average their outputs then the detector is robust, achieving maximal confidence only when both loops of the '8' are present.



Bagging

Consider a regression problem in which we are trying to predict the value of a single continuous variable, and suppose we generate M bootstrap data sets and then use each to train a separate copy $f_m(x)$ of a predictive model where $m = 1, \ldots, M$. The committee prediction is given by

$$f_{COM}(x) = \frac{1}{M}\sum_{m=1}^{M} f_m(x)$$

Suppose the true regression function we are trying to predict is $h(\boldsymbol{x})$, so that the output of each of the models can be written as the true value plus an error in the form

$$f_m(x) = h(x) + \epsilon_m(x)$$

Bagging

The average sum-of-squares error then takes the form

$$\mathbb{E}_x[\{f_m(x) - h(x)\}^2] = \mathbb{E}_x[\epsilon_m(x)^2]$$

Bagging

The average sum-of-squares error then takes the form

$$\mathbb{E}_x[\{f_m(x) - h(x)\}^2] = \mathbb{E}_x[\epsilon_m(x)^2]$$

The average error made by the models acting individually

$$\mathbb{E}_{IN} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_x[\epsilon_m(x)^2]$$

Bagging

The average sum-of-squares error then takes the form

$$\mathbb{E}_x[\{f_m(x) - h(x)\}^2] = \mathbb{E}_x[\epsilon_m(x)^2]$$

The average error made by the models acting individually

$$\mathbb{E}_{IN} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_x[\epsilon_m(x)^2]$$

The expected error made from the committee is given by,

$$\mathbb{E}_{COM} = \mathbb{E}_x\left[\left\{\frac{1}{M}\sum_{m=1}^{M} f_m(x) - h(x)\right\}^2\right] = \mathbb{E}_x\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(x)\right\}^2\right]$$

Bagging

Assume that the errors have zero mean and uncorrelated, we have,

$$\mathbb{E}_x[\epsilon_m(x)] = 0$$

$$\mathbb{E}_x[\epsilon_m(x)\epsilon_l(x)] = 0 \quad m \neq l$$

Then we obtain,

$$\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{IN}$$

Bagging

Assume that the errors have zero mean and uncorrelated, we have,

$$\mathbb{E}_x[\epsilon_m(x)] = 0$$

$$\mathbb{E}_x[\epsilon_m(x)\epsilon_l(x)] = 0 \quad m \neq l$$

Then we obtain,

$$\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{IN}$$

- The average error of a model can be reduced by a factor of M simply by averaging.
- The key assumption that the errors due to the individual models are uncorrelated. (In practice, **this is not true!**)
- The reduction in overall error is generally small. However, we still have,

$$\mathbb{E}_{COM} \leq \mathbb{E}_{IN}$$

Ensemble Methods

- Bagging
- Random Forest
- Boosting (Adaboost)
- Gradient Boosting Decision Trees

Ensemble Methods

- Bagging
- Random Forest
- Boosting (Adaboost)
- Gradient Boosting Decision Trees

CART (1984)

Breiman	Friedman	Olshen	Stone
UC Berkeley	Stanford University	Stanford University	UC Berkeley

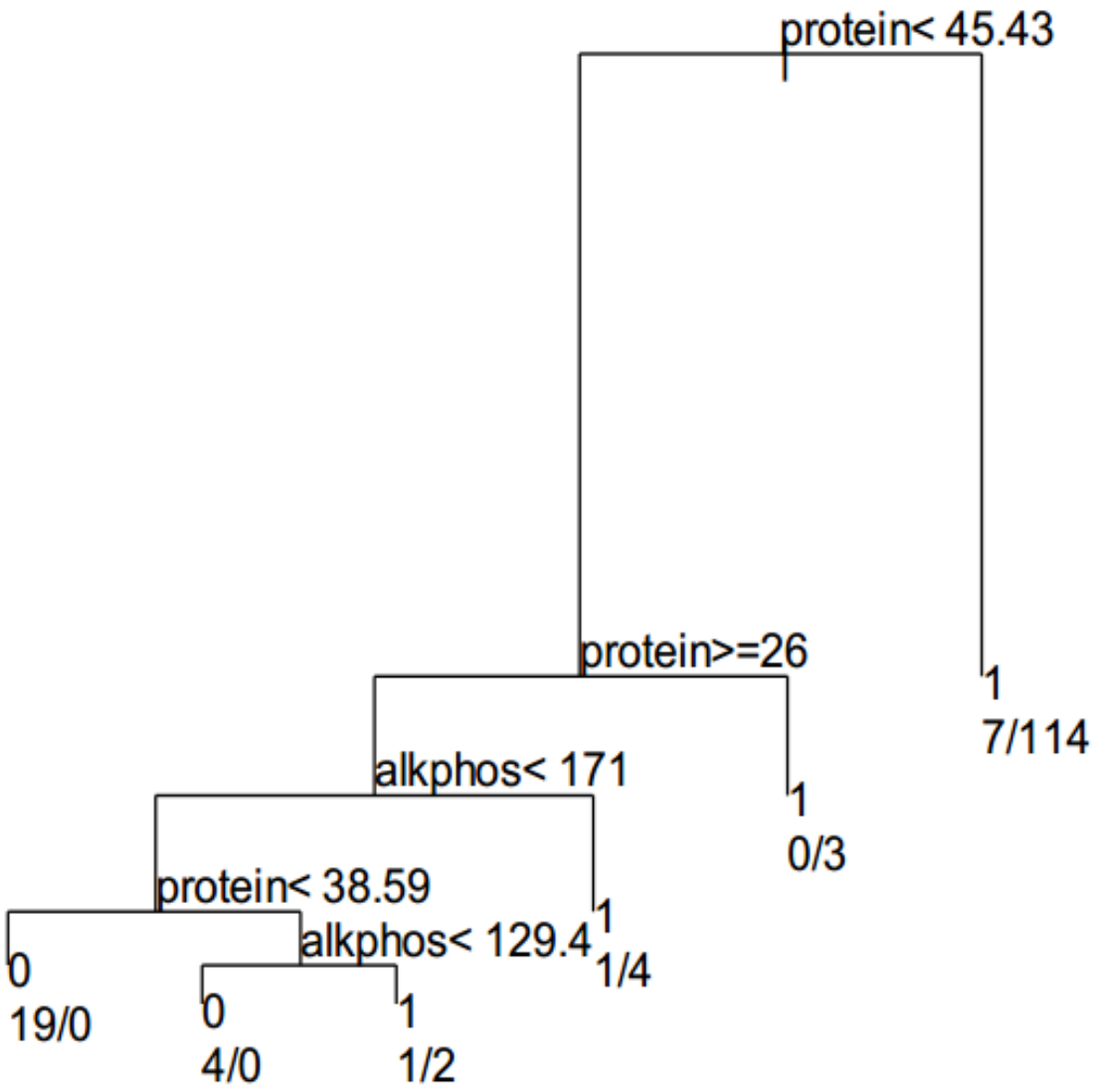
CART

- Grow a binary tree.
  - At each node, split the data into two child nodes.
- Splits are chosen using a splitting criterion.
- Bottom nodes are “terminal” nodes.
- For regression the predicted value at a node is the *average* response variable for all observations in the node.
- For classification the predicted class is the *most common class* in the node (majority voting).
- For classification, can also get estimated probability of membership in each of the classes.

A Classification Tree

- Predict hepatitis (0=absent, 1=present)
- Using *protein* and *alkaline phosphate.*

- "Yes" goes left.



Advantages of CART

- Applicable to both regression and classification problems
- Handle categorical predictors naturally
- Computationally simple and quick to fit, even for large problems.
- No formal distribution assumption (non-parametric).

Advantages of CART

- Applicable to both regression and classification problems
- Handle categorical predictors naturally
- Computationally simple and quick to fit, even for large problems.
- No formal distribution assumption (non-parametric).

The term “non-parametric” might sound a bit confusing at first: non-parametric does not mean that they have NO parameters!
On the contrary, non-parametric models (can) become more complex with an increasing amount of data.

Parametric vs Nonparametric Models

- Parametric models assume some finite set of parameters θ. Given the parameters, future predictions are independent of the observed data $f(x|\theta, \mathcal{D}) = f(x|\theta)$

  A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called a parametric model. No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs.
  — Artificial Intelligence: A Modern Approach

- Non-parametric models assume that the data distribution cannot be defined in terms of such a finite set of parameters (θ can be infinite dimensional).
- Does not assume anything about the form of the mapping function.
- The number of parameters grows with the size of the training set.

  A precise and universally acceptable definition of the term 'non-parametric' is not presently available.
  —The Handbook of Non parametric Statistics 1 (1962) on p. 2

Parametric Models

- Parametric models involve two steps:
  - Select a form for the function.
  - Learn the parameters for the function from the training data.

Pros

- **Simpler**: Easier to understand and interpret results.
- **Speed**: Fast to learn. Once fitted, only parameters need to be stored.
- **Less Data**: Do not require as much training data and can work well.

Cons

- **Constrained**: Highly constrained to the specified function form.
- **Limited Complexity**: More suited to simpler problems.
- **Underfitting**: Are unlikely to fit the underlying mapping function.

Nonparametric Models

- An easy to understand nonparametric model
  - The k-nearest neighbors algorithm
  - Makes predictions based on the k most similar training samples for a new data.
  - Does not assume anything about the form of the mapping function.

Pros

- **Flexibility**: Capable of fitting a large number of functional forms.
- **Power**: No assumptions about the underlying function.
- **Performance**: Can result in higher performance models for prediction.

Cons

- **More data**: Require a lot more training data to estimate the mapping function.
- **Slower**: Have more parameters to train. The entire dataset needs to be stored and the computation involves all data.
- **Overfitting**: More of a risk to overfit and it is harder to interpret the results.

Parametric vs Nonparametric Models

Parametric: *a learning model that summarizes data with a finite set of parameters of fixed size (independent of the number of training examples).*

Non-parametric: does not assume anything about the form of the mapping function.

Parametric vs Nonparametric Models

Parametric: *a learning model that summarizes data with a finite set of parameters of fixed size (independent of the number of training examples).*
Non-parametric: does not assume anything about the form of the mapping function.

- **Parametric models:**
  - Logistic Regression
  - Linear Discriminant Analysis
  - Perceptron $f(\boldsymbol{x}) = sign(\boldsymbol{w}^T \boldsymbol{x} + b)$
  - Naive Bayes
  - Linear SVM

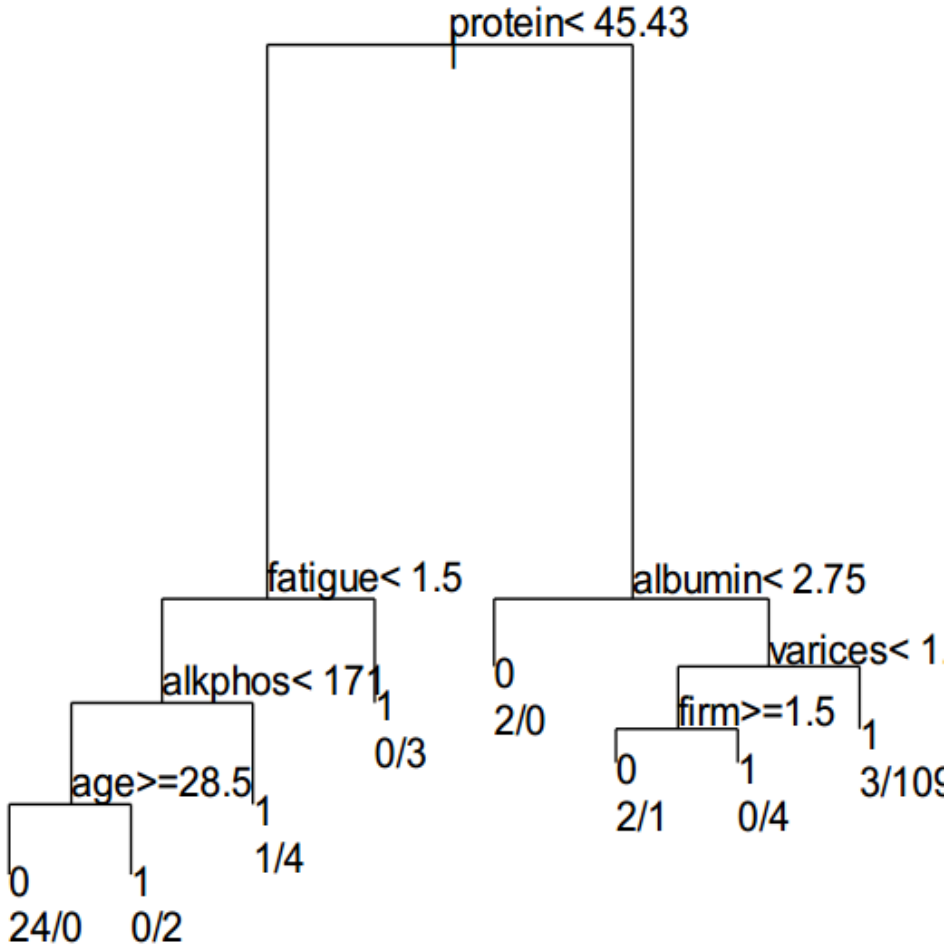
- **Nonparametric models:**
  - k-Nearest Neighbors
  - Decision Trees, like CART and C4.5
  - Advanced SVM (RBF)

Advantages of CART

- Applicable to both regression and classification problems
- Handle categorical predictors naturally
- Computationally simple and quick to fit, even for large problems.
- No formal distribution assumption (non-parametric).
- Can handle highly non-linear interactions and classification boundaries.
- Automatic variable selection.
- Handle missing values. Using proximities.
- Very easy to interpret if the tree is small.

Advantages of CART (Cont.)

- The picture of the tree can give valuable insights into which variables are important and where.
- The terminal nodes suggest a natural clustering of data.



Disadvantages of CART

- *Accuracy:* current methods, such as SVM and ensemble classifiers often have 30% lower error rates than CART.
- *Instability:* if we change the data a little, the tree picture can change a lot. So the interpretation is not as straightforward as it appears.

Disadvantages of CART

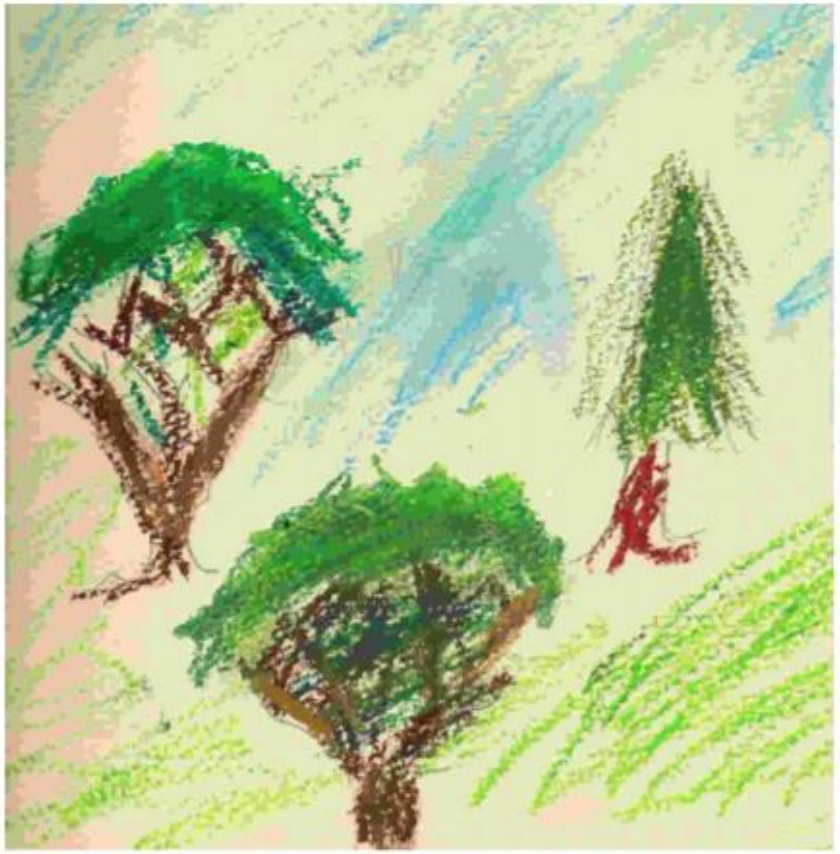
- *Accuracy:* current methods, such as SVM and ensemble classifiers often have 30% lower error rates than CART.
- *Instability:* if we change the data a little, the tree picture can change a lot. So the interpretation is not as straightforward as it appears.

Today, we can do better!

Random Forest

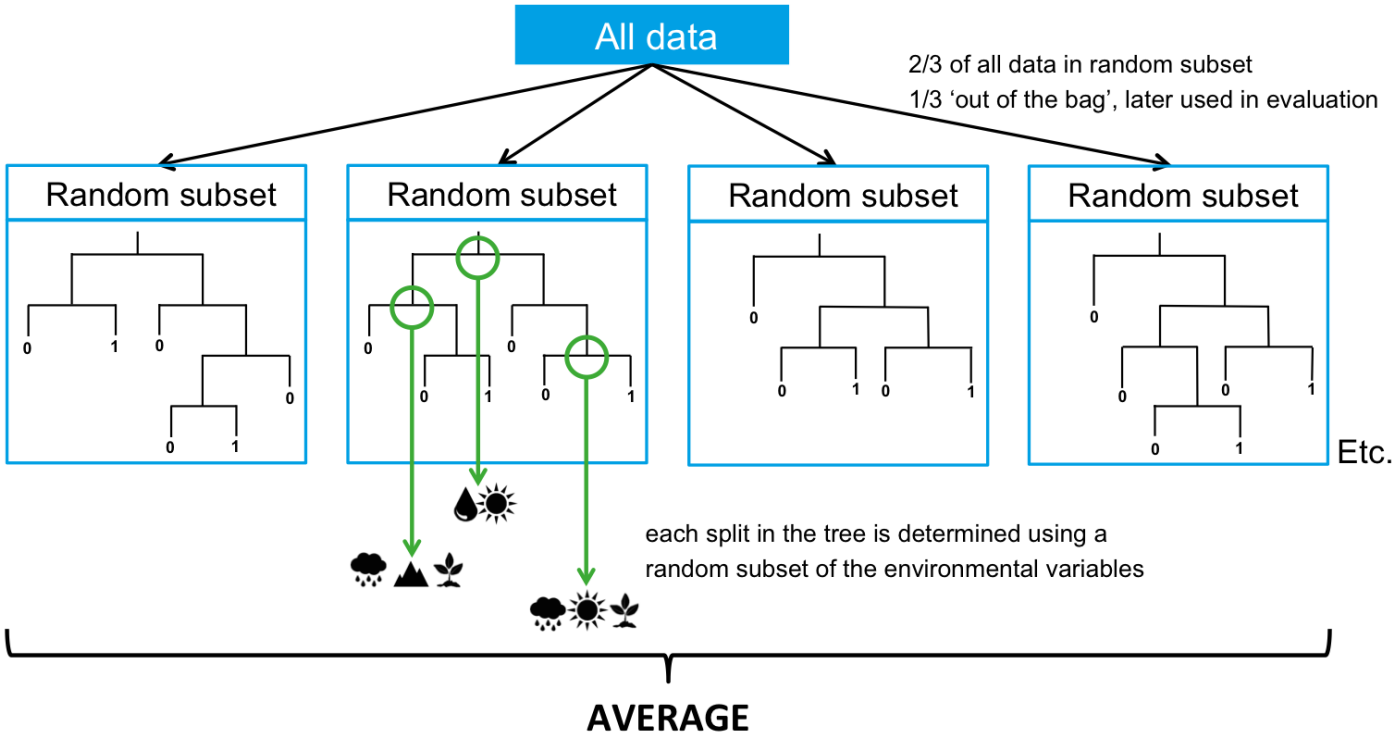
Random Forest

- Ensemble method specifically designed for decision tree classifiers.
- Random forest is a substantial modification of bagging that builds a large collection of de-correlated trees, and then average them.



Random Forest

- Two sources of randomness:
  - Bagging method: each tree is grown using a bootstrap sample of training data
  - Random vector method: at each node, best split is chosen from a random sample of m attributes instead of all attributes.



> find the set of predictor variables that produce the strongest classification model

Random Forest Algorithm

1. For b=1 to B:
   - Draw a bootstrap sample $\boldsymbol{D}^*$ of size N from the training data
   - Grow a tree T_b from $\boldsymbol{D}^*$, by recursively repeating the following steps

     For each terminal node of the tree, until the minimum node size n_{min} is reached.
     - Select m variables at random from the d variables.
       - Typically $m = log_2 d$
     - Pick the best variable/split-point among the m.
     - Split the node into two daughter nodes
2. Output the ensemble of trees $\{T_b\}_1^B$.

Random Forest Algorithm

1. For b=1 to B:
   - Draw a bootstrap sample $\boldsymbol{D}^*$ of size N from the training data
   - Grow a tree T_b from $\boldsymbol{D}^*$, by recursively repeating the following steps

     For each terminal node of the tree, until the minimum node size n_{min} is reached.
     - Select m variables at random from the d variables.
       - Typically $m = log_2 d$
     - Pick the best variable/split-point among the m.
     - Split the node into two daughter nodes
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new points,

- *Regression*: prediction average $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$
- *Classification*: majority voting $\hat{C}_{rf}^B(x) = majority\ vote\ \{\hat{C}_b(x)\}_1^B$.

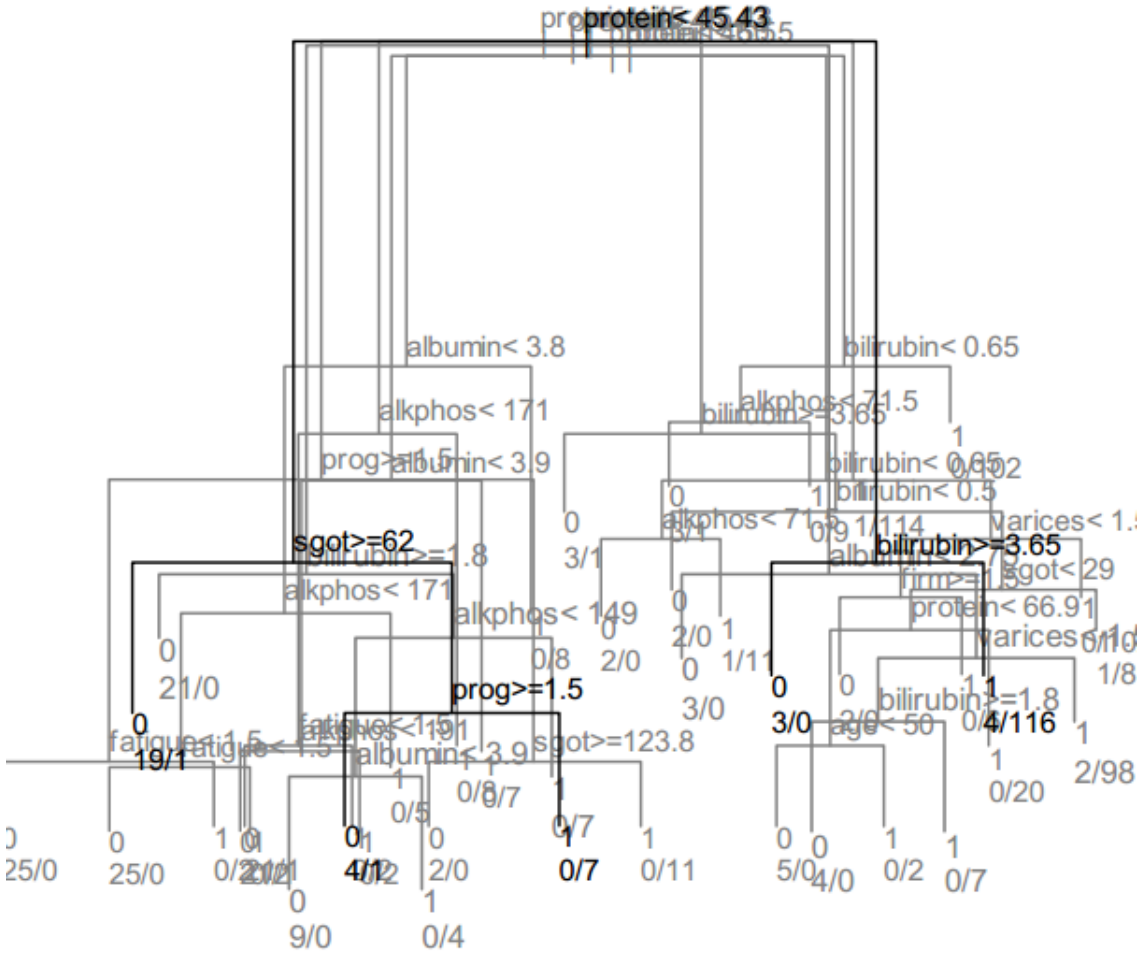
Inherit Many Advantages of CART

- Applicable to both regression and classification problems
- Handle categorical predictors naturally
- Computationally simple and quick to fit, even for large problems.
- No formal distribution assumption (non-parametric).
- Can handle highly non-linear interactions and classification boundaries.
- Automatic variable selection.
- Handle missing values. Using proximities.
- ~~Very easy to interpret if the tree is small.~~ NO!

But Do Not Inherit…

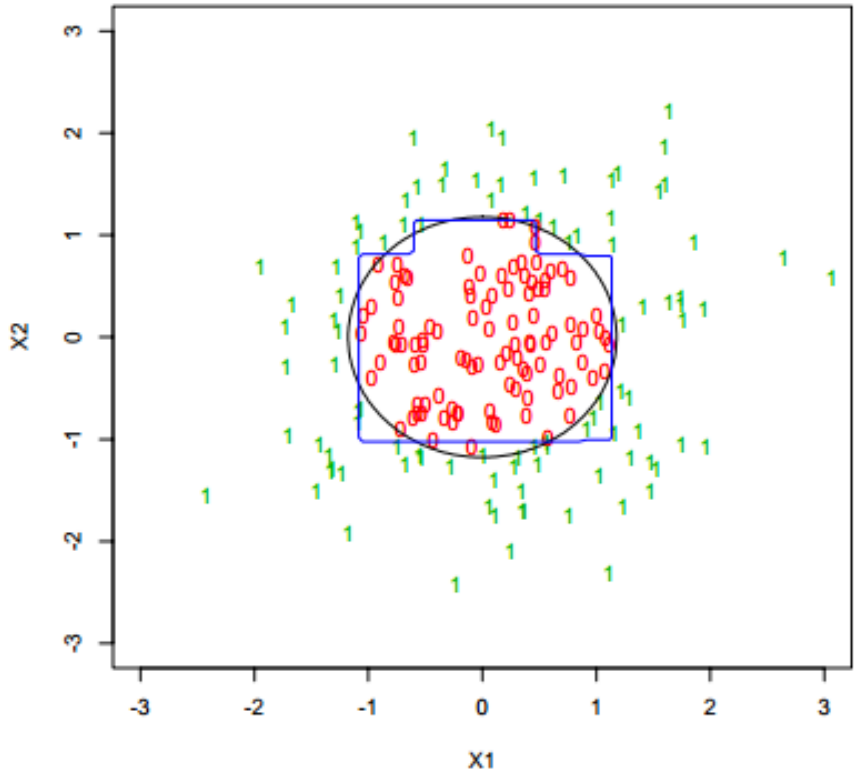
- The picture of the tree can give valuable insights into which variables are important and where.

- The terminal nodes suggest a natural clustering of data.

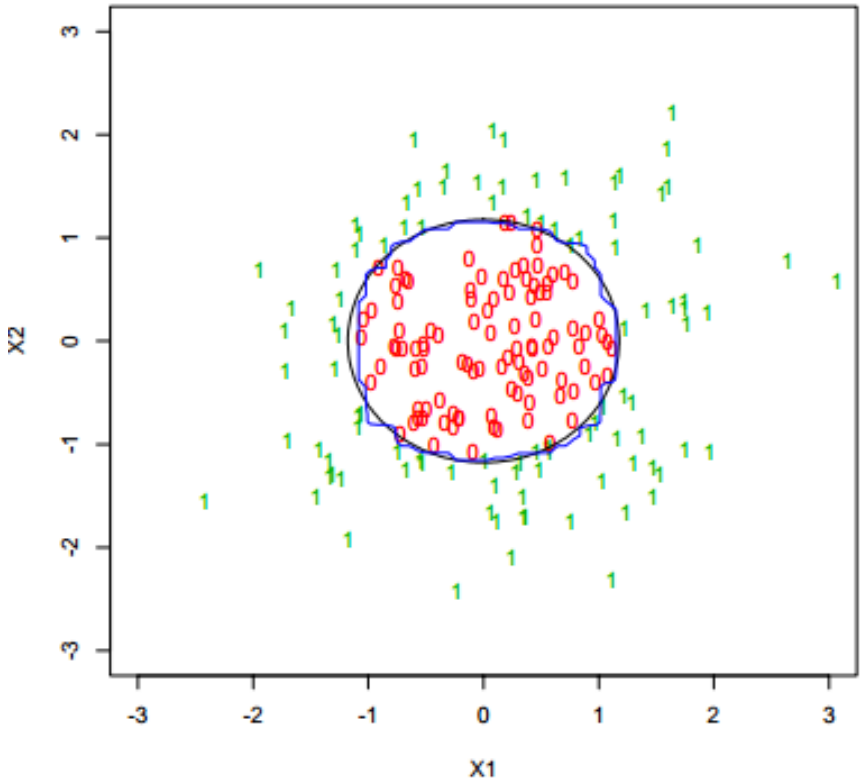


Decision Boundary: Bagging

- Bagging averages many trees, and produces smoother decision boundaries.



Single Tree



Bagging

Ensemble Methods

- Bagging
- Random Forest
- Boosting (Adaboost)
- Gradient Boosting Decision Trees

Ensemble Methods

- Bagging
- Random Forest
- Boosting (Adaboost)
- Gradient Boosting Decision Trees

Boosting

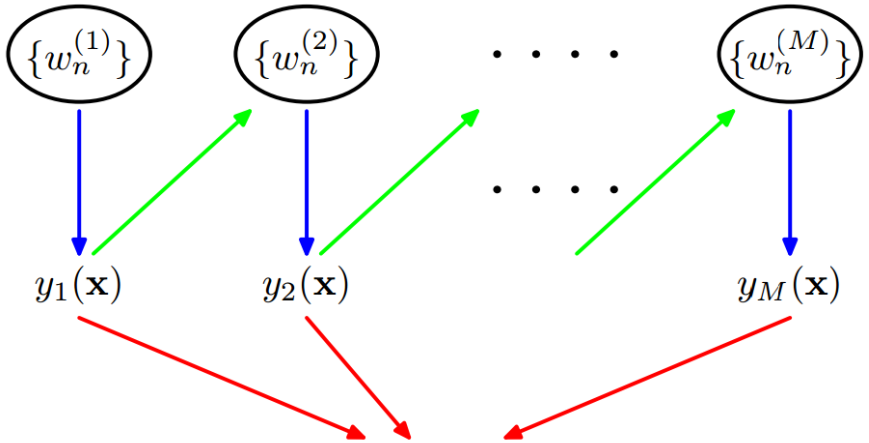
- Boosting is a powerful technique for combining multiple 'base' classifiers to produce a form of committee whose performance can be significantly better than that of any of the base classifiers.

- Boosting can give good results even if the base classifiers have a performance that is only slightly better than random, and hence sometimes the base classifiers are known as *weak learners*.

- *AdaBoost*, short for 'adaptive boosting', developed by Freund and Schapire 1996).

Boosting

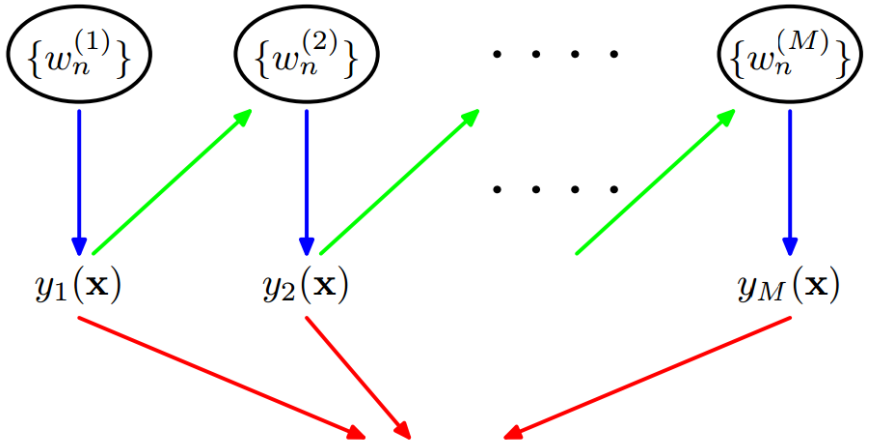
- The base classifiers are trained in sequence,
- Each base classifier is trained using a weighted form of the data set,
- The weighting coefficient associated with each data point depends on the performance of the previous classifiers.



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

Boosting

- The base classifiers are trained in sequence,
- Each base classifier is trained using a weighted form of the data set,
- The weighting coefficient associated with each data point depends on the performance of the previous classifiers.
  - Points that are misclassified by one of the base classifiers are given greater weight when used to train the next classifier in the sequence.

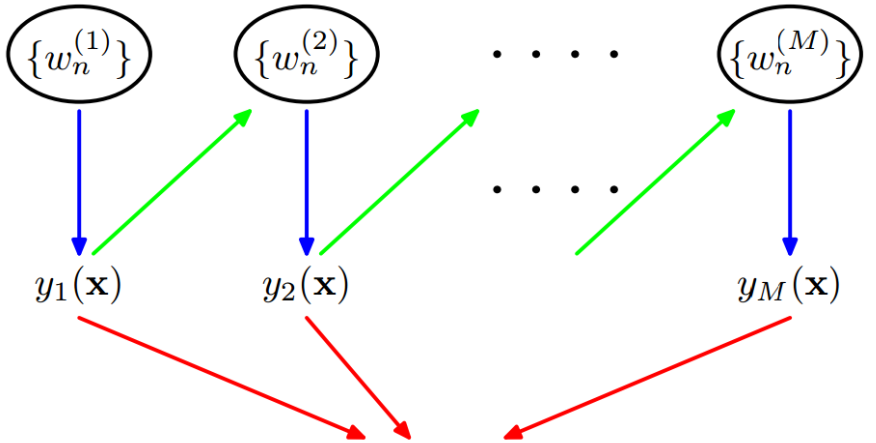


$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

Boosting

- The base classifiers are trained in sequence,
- Each base classifier is trained using a weighted form of the data set,
- The weighting coefficient associated with each data point depends on the performance of the previous classifiers.
  - Points that are misclassified by one of the base classifiers are given greater weight when used to train the next classifier in the sequence.

- Once all the classifiers have been trained, their predictions are then combined through a weighted majority voting scheme.



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

The Born of AdaBoost

An open problem [Kearns & Valiant, STOC'89]:

"weakly learnable" ?= "strongly learnable"

a problem is ***learnable*** or ***strongly learnable*** if there exists an algorithm that outputs a learner h in polynomial time such that for all $0 < \delta, \epsilon \leq 0.5$, $P(\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[\mathbb{I}[h(\boldsymbol{x}) \neq f(\boldsymbol{x})]] < \epsilon) \geq 1 - \delta$

a problem is ***weakly learnable*** if there exists an algorithm that outputs a learner with error $0.5–1/p$ where p is a polynomial in problem size and other parameters

In other words, whether a "weak" learning algorithm that works just slightly better than random guess can be "boosted" into an arbitrarily accurate "strong" learning algorithm

The Born of AdaBoost

- Amazingly, in 1990 Schapire proves that the answer is "yes". More importantly, the proof is a construction!

  This is the first Boosting algorithm

- In 1993, Freund presents a scheme of combining weak learners by majority voting in Phd thesis at UC Santa Cruz

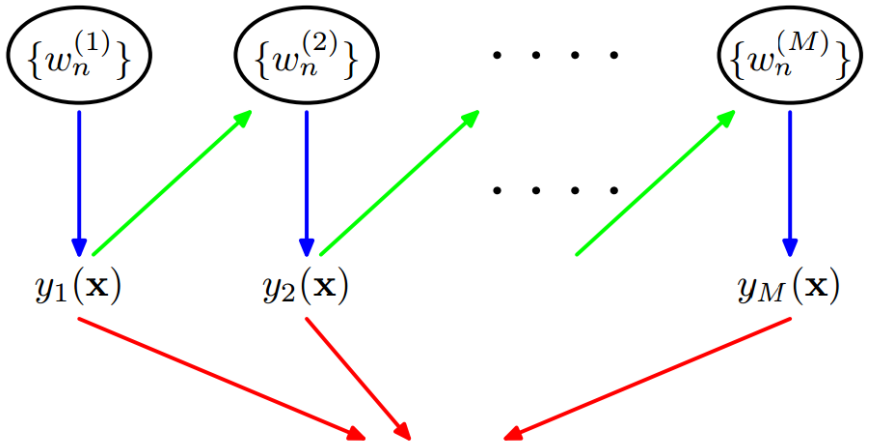
  However, these algorithms are not practical

- Later, at AT&T Bell Labs, Freund & Schapire published the 1997 journal paper (the work was reported in EuroCOLT'95), which proposed the AdaBoost algorithm, a practical algorithm

AdaBoost

- Consider a two-class classification problem.
- The training data:$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ along with corresponding binary target variables $t_1, \ldots, t_N$ where $t_n \in \{-1, 1\}$.
- Each data point is given an associated weighting parameter w_n, which is initially set $1/N$ for all data points.

- We shall suppose that we have a procedure available for training a base classifier using weighted data to give a function $y(\boldsymbol{x}) \in \{-1, 1\}$.

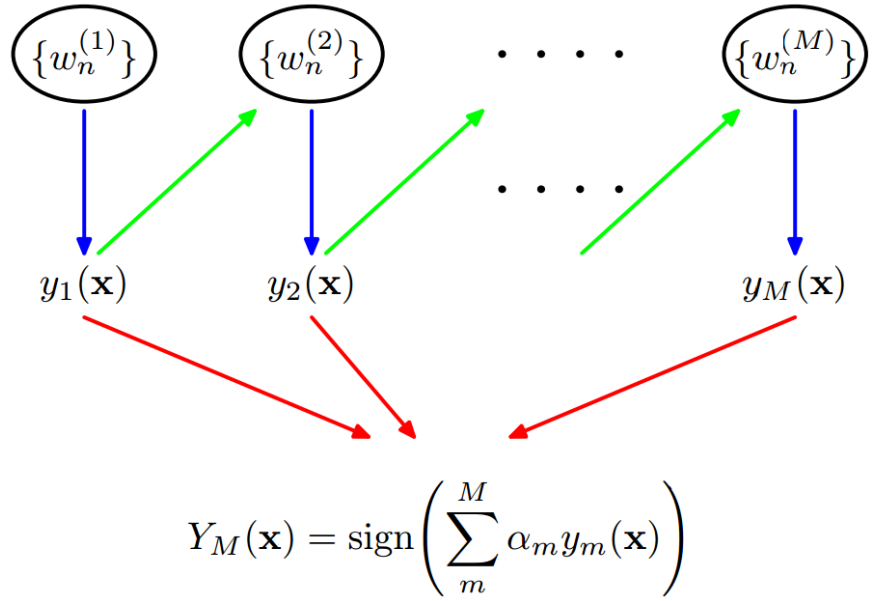


$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

AdaBoost

- At each stage, AdaBoost trains a new classifier using a data set in which the weighting coefficients are adjusted according to the performance of the previously trained classifier.
  - Give greater weight to the misclassified data points.
- When the desired number of base classifiers have been trained, they are combined to form a committee using coefficients that give different weight to different base classifiers.

$$f_M(x) = \frac{1}{2}\sum_{l=1}^{M} \alpha_l y_l(x)$$

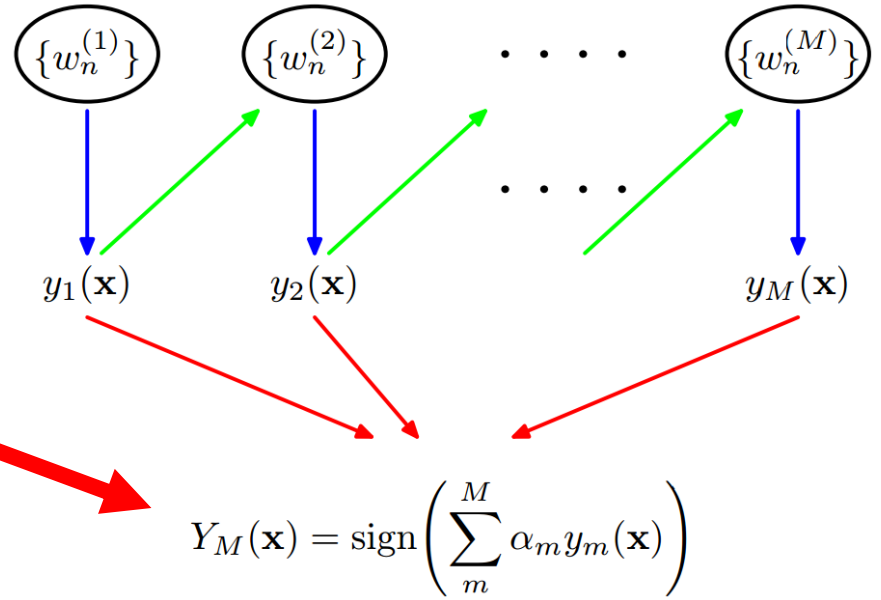


AdaBoost

- At each stage, AdaBoost trains a new classifier using a data set in which the weighting coefficients are adjusted according to the performance of the previously trained classifier.
  - Give greater weight to the misclassified data points.
- When the desired number of base classifiers have been trained, they are combined to form a committee using coefficients that give different weight to different base classifiers.

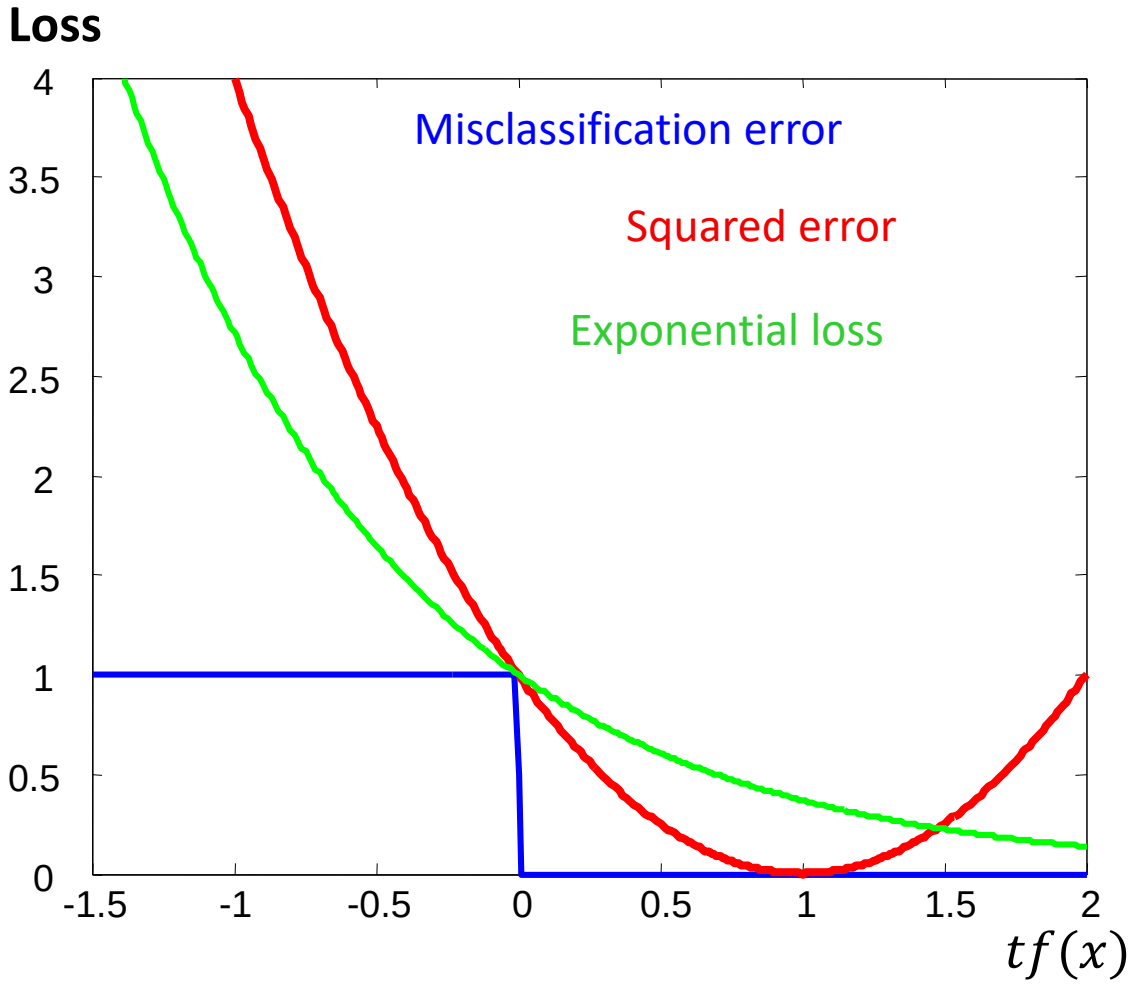
$$f_M(x) = \frac{1}{2}\sum_{l=1}^{M} \alpha_l y_l(x)$$

Equivalent



Exponential Error Function

[proposed by Schapire and Singer 1998 as an upper bound on misclassification error]



Squared error

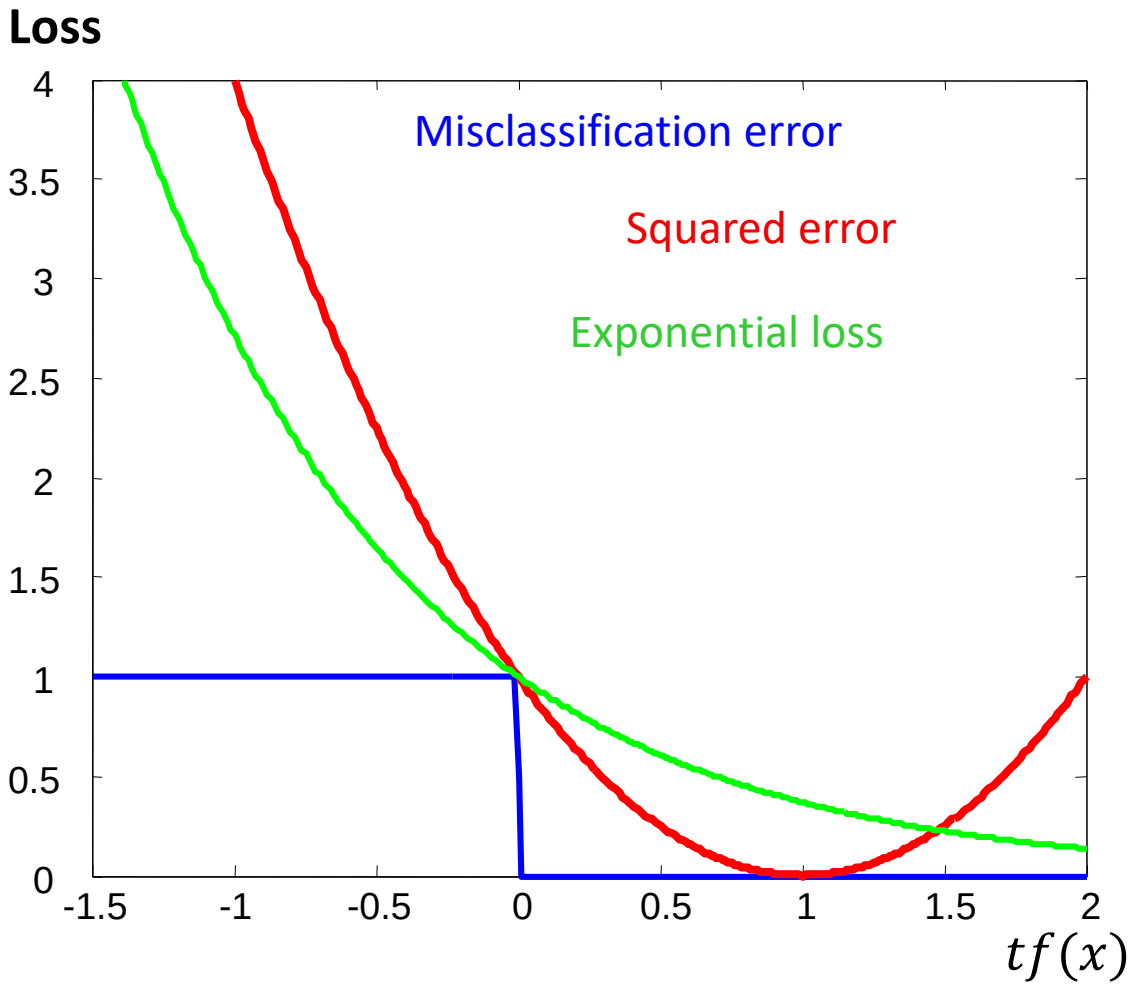
$$J = \sum_{n=1}^{N} (t_n - f(x_n))^2$$

Exponential loss

$$J = \sum_{n=1}^{N} \exp\{-t_n f(x_n)\}$$

Exponential Error Function

[proposed by Schapire and Singer 1998 as an upper bound on misclassification error]



- Exponential loss is a monotone, smooth upper bound on misclassification loss at x.
- Leads to simple reweighting scheme.

Additive Classification Model

For binary classification, $t \in \{-1, 1\}$

$$f(x) = \sum_{l=1}^{M} \alpha_l y_l(x)$$

$$p(t = 1|x) = \frac{\exp\big(f(x)\big)}{1 + \exp\big(f(x)\big)}$$

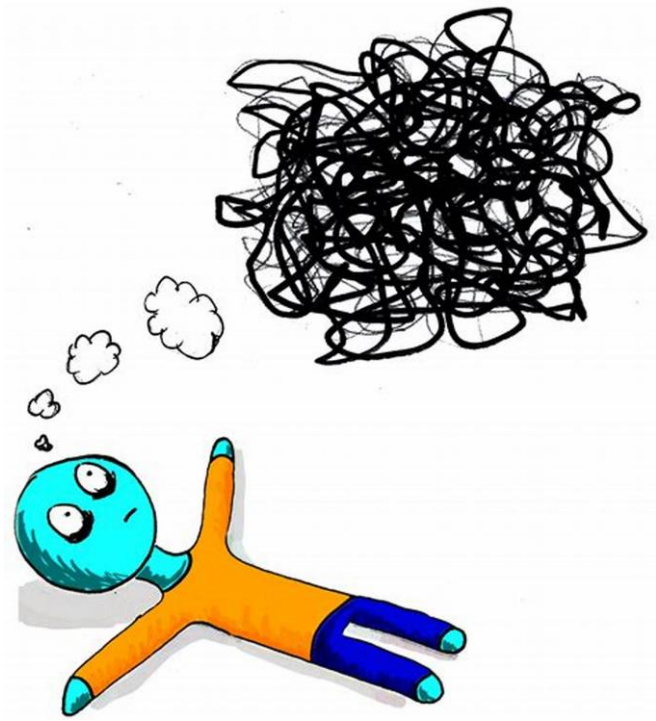
$$p(t = -1|x) = \frac{1}{1 + \exp\big(f(x)\big)}$$

Exponential Error Function

For binary classification, consider the exponential error function

$$J(f) = \mathbb{E}[\exp\{-tf(x)\}] \qquad t \in \{-1, 1\}$$

It is almost equivalent to the logistic cross entropy loss (for t=+1, t=-1 labels).



Exponential Error Function

For binary classification, consider the exponential error function

$$J(f) = \mathbb{E}[\exp\{-tf(x)\}] \qquad t \in \{-1, 1\}$$

It is almost equivalent to the logistic cross entropy loss (for t=+1, t=-1 labels).

Let us revisit the logistic cross entropy loss!

Logistic Regression--$t \in \{0,1\}$

Logistic regression use $h(\boldsymbol{x}) = \theta(\boldsymbol{w}^T\boldsymbol{x})$ to approximate the target $f(\boldsymbol{x}) = p(+1|\boldsymbol{x})$

Consider $\mathcal{D} = \{(\boldsymbol{x_1}, +), (\boldsymbol{x_2}, -), \dots, (\boldsymbol{x_m}, -)\}$

$$\theta(s) = \frac{e^{\boldsymbol{s}}}{1+e^{\boldsymbol{s}}} = \frac{1}{1+e^{-\boldsymbol{s}}}$$

$$p(t_i|\boldsymbol{x_i}) = \begin{cases} h(\boldsymbol{x_i}) & \text{for } t_i = 1 \\ 1 - h(\boldsymbol{x_i}) & \text{for } t_i = 0 \end{cases} \Leftrightarrow p(t_i|\boldsymbol{x_i}) = h(\boldsymbol{x_i})^{t_i}(1 - h(\boldsymbol{x_i}))^{(1-t_i)}$$

Bernoulli distribution

$$likelihood(h) = \prod_{i=1}^{m} p(\boldsymbol{x_i})p(t_i|\boldsymbol{x_i}) \propto \prod_{i=1}^{m} p(t_i|\boldsymbol{x_i}) = \prod_{i=1}^{m} h(\boldsymbol{x_i})^{t_i}(1 - h(\boldsymbol{x_i}))^{(1-t_i)}$$

Logistic Regression--$t \in \{0,1\}$

Negative Log-likelihood

$$\min_{h} E(h) = \sum_{i=1}^{m} -(t_i ln\, h(\boldsymbol{x_i}) + (1 - t_i)\, ln(1 - h(\boldsymbol{x_i}))\,)$$

Cross-entropy loss

Cross-entropy

$$H(p, q) = -\sum_{x} p(x) \log(q(x))$$

$p \in \{t, 1 - t\}$
$q \in \{h(\boldsymbol{x}), 1 - h(\boldsymbol{x})\}$

Negative Log-likelihood

$$\min_{\boldsymbol{w}} \sum_{i=1}^{m} \left[-t_i ln \left(\frac{1}{1 + e^{-\boldsymbol{w^T x_i}}} \right) - (1 - t_i) ln \left(\frac{1}{1 + e^{\boldsymbol{w^T x_i}}} \right) \right]$$

$$\min_{\boldsymbol{w}} \sum_{i=1}^{m} \left[-t_i \boldsymbol{w^T x_i} + ln(1 + e^{\boldsymbol{w^T x_i}}) \right]$$

Logistic Regression--$t \in \{-1,1\}$

Consider $\mathcal{D} = \{(\boldsymbol{x_1}, +), (\boldsymbol{x_2}, -), \dots, (\boldsymbol{x_m}, -)\}$

$$h(\boldsymbol{x_i}) = P(+1|\boldsymbol{x_i}) \quad \Leftrightarrow \quad p(t|\boldsymbol{x_i}) = \begin{cases} h(\boldsymbol{x_i}) & \text{for } t = +1 \\ 1 - h(\boldsymbol{x_i}) & \text{for } t = -1 \end{cases}$$

$$\Leftrightarrow \quad p(t|\boldsymbol{x_i}) = \begin{cases} h(\boldsymbol{x_i}) & \text{for } t = +1 \\ h(-\boldsymbol{x_i}) & \text{for } t = -1 \end{cases} \quad \Leftrightarrow \quad p(t|\boldsymbol{x_i}) = h(t\boldsymbol{x_i})$$

$$1 - \theta(z) = \theta(-z)$$

$$\theta(s) = \frac{e^{\boldsymbol{s}}}{1 + e^{\boldsymbol{s}}} = \frac{1}{1 + e^{-\boldsymbol{s}}}$$

Logistic Regression--$t \in \{-1,1\}$

Consider $\mathcal{D} = \{(\boldsymbol{x_1}, +), (\boldsymbol{x_2}, -), \dots, (\boldsymbol{x_m}, -)\}$

$$likelihood(h) = \prod_{i=1}^{m} p(\boldsymbol{x_i}) p(t_i|\boldsymbol{x_i}) = p(\boldsymbol{x_1}) h(\boldsymbol{x_1}) p(\boldsymbol{x_2}) h(-\boldsymbol{x_2}) \dots p(\boldsymbol{x_m}) h(-\boldsymbol{x_m})$$

$$\max_h likelihood(h) \propto \prod_{i=1}^{m} p(t_i|\boldsymbol{x_i}) = \prod_{i=1}^{m} h(t_i \boldsymbol{x_i}) = \prod_{i=1}^{m} \theta(t_i \boldsymbol{w^T x_i})$$

$$\min_{\boldsymbol{w}} -\sum_{i=1}^{m} \ln \theta(t_i \boldsymbol{w^T x_i}) \quad \Leftrightarrow \quad \min_{\boldsymbol{w}} -\sum_{i=1}^{m} \ln 1/(1 + e^{-t_i \boldsymbol{w^T x_i}})$$

$$\min_{\boldsymbol{w}} -\frac{1+t_i}{2} \sum_{i=1}^{m} \ln \frac{1}{1+e^{-\boldsymbol{w^T x_i}}} - \frac{1-t_i}{2} \sum_{i=1}^{m} \ln \frac{1}{1+e^{\boldsymbol{w^T x_i}}}$$

Exponential Error Function

For binary classification, consider the exponential error function

$$J(f) = \mathbb{E}[\exp\{-tf(\boldsymbol{x})\}] \qquad t \in \{-1, 1\}$$

It is almost equivalent to the logistic cross entropy loss (for t=+1, t=-1 labels).

Solution 1:

$$f(\boldsymbol{x}) = \sum_{l=1}^{M} \alpha_l y_l(\boldsymbol{x})$$

$$\min_{\boldsymbol{w}} -\frac{1+t_i}{2} \sum_{\mathrm{i}=1}^{m} \ln \frac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}_i}} - \frac{1-t_i}{2} \sum_{\mathrm{i}=1}^{m} \ln \frac{1}{1+e^{\boldsymbol{w}^T \boldsymbol{x}_i}}$$

$$p(t=1|\boldsymbol{x}) = \frac{\exp\big(f(\boldsymbol{x})\big)}{1+\exp\big(f(\boldsymbol{x})\big)}$$

$$p(t=-1|\boldsymbol{x}) = \frac{1}{1+\exp\big(f(\boldsymbol{x})\big)}$$

Exponential Error Function

For binary classification, consider the exponential error function

$$J(f) = \mathbb{E}[\exp\{-tf(\boldsymbol{x})\}] \qquad t \in \{-1, 1\}$$

It is almost equivalent to the logistic cross entropy loss (for t=+1, t=-1 labels).

Solution 1:

$$f(\boldsymbol{x}) = \sum_{l=1}^{M} \alpha_l y_l(\boldsymbol{x})$$

$$p(t = 1|\boldsymbol{x}) = \frac{\exp\big(f(\boldsymbol{x})\big)}{1 + \exp\big(f(\boldsymbol{x})\big)}$$

$$p(t = -1|\boldsymbol{x}) = \frac{1}{1 + \exp\big(f(\boldsymbol{x})\big)}$$

$$\min_{\boldsymbol{w}} -\frac{1 + t_i}{2} \sum_{\mathrm{i}=1}^{m} \ln \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}_i}} - \frac{1 - t_i}{2} \sum_{\mathrm{i}=1}^{m} \ln \frac{1}{1 + e^{\boldsymbol{w}^T \boldsymbol{x}_i}}$$

$$\min_{\boldsymbol{w}} -\frac{1 + t_i}{2} \sum_{\mathrm{i}=1}^{m} \ln \frac{1}{1 + e^{-f(\boldsymbol{x}_i)}} - \frac{1 - t_i}{2} \sum_{\mathrm{i}=1}^{m} \ln \frac{1}{1 + e^{f(\boldsymbol{x}_i)}}$$

Exponential Error Function

For binary classification, consider the exponential error function

$$J(f) = \mathbb{E}[\exp\{-tf(\boldsymbol{x})\}] \qquad t \in \{-1, 1\}$$

It is almost equivalent to the logistic cross entropy loss (for t=+1, t=-1 labels).

Solution 1:

$$\mathcal{L}(t, \boldsymbol{x}) = -\frac{1+t}{2}\log\frac{\exp\{f(\boldsymbol{x})\}}{1+\exp\{f(\boldsymbol{x})\}} - \frac{1-t}{2}\log\frac{1}{1+\exp\{f(\boldsymbol{x})\}}$$

$$\mathcal{L}(t, \boldsymbol{x}) = -\frac{1+t}{2}(f(\boldsymbol{x}) - \log(1+\exp\{f(\boldsymbol{x})\})) + \frac{1-t}{2}\log(1+\exp\{f(\boldsymbol{x})\})$$

$$\mathcal{L}(t, \boldsymbol{x}) = -\frac{1+t}{2}f(\boldsymbol{x}) + \log(1+\exp\{f(\boldsymbol{x})\})$$

$$\mathcal{L}(t, \boldsymbol{x}) = \log\frac{1+\exp\{f(\boldsymbol{x})\}}{\exp\left\{\frac{1+t}{2}f(\boldsymbol{x})\right\}} = \begin{cases}\log(1+\exp\{f(\boldsymbol{x})\}) & if\ t=-1\\ \log(1+\exp\{-f(\boldsymbol{x})\}) & if\ t=+1\end{cases} = \log(1+\exp\{-tf(\boldsymbol{x})\})$$

Exponential Error Function

For binary classification, consider the exponential error function

$$J(f) = \mathbb{E}[\exp\{-tf(\boldsymbol{x})\}] \qquad t \in \{-1, 1\}$$

It is almost equivalent to the logistic cross entropy loss (for t=+1, t=-1 labels).

Solution 2:

$$J(f) = \int \left[p(t=1|\boldsymbol{x})e^{-f(x)} + p(t=-1|\boldsymbol{x})e^{f(x)}\right]p(\boldsymbol{x})d\boldsymbol{x}$$

$$\frac{\partial J(f)}{\partial f} = -p(t=1|\boldsymbol{x})\exp\{-f(\boldsymbol{x})\} + p(t=-1|\boldsymbol{x})\exp\{f(\boldsymbol{x})\} = 0$$

$$f(\boldsymbol{x}) = \frac{1}{2}\ln\frac{p(t=1|\boldsymbol{x})}{p(t=-1|\boldsymbol{x})} \qquad p(t=1|x) = \frac{\exp\{2f(\boldsymbol{x})\}}{1+\exp\{2f(\boldsymbol{x})\}}$$

Hence, AdaBoost and LR are equivalent up to a factor 2.

Exponential Error Function

For binary classification, consider the exponential error function

$$J(f) = \mathbb{E}[\exp\{-tf(\boldsymbol{x})\}] \qquad t \in \{-1, 1\}$$

It is almost equivalent to the logistic cross entropy loss (for t=+1, t=-1 labels).

Solution 2:

$$J(f) = \int \left[p(t=1|\boldsymbol{x})e^{-f(x)} + p(t=-1|\boldsymbol{x})e^{f(x)}\right]p(\boldsymbol{x})d\boldsymbol{x}$$

$$\frac{\partial J(f)}{\partial f} = -p(t=1|\boldsymbol{x})\exp\{-f(\boldsymbol{x})\} + p(t=-1|\boldsymbol{x})\exp\{f(\boldsymbol{x})\} = 0$$

$$f(\boldsymbol{x}) = \frac{1}{2}\ln\frac{p(t=1|\boldsymbol{x})}{p(t=-1|\boldsymbol{x})} \qquad sign\, f(\boldsymbol{x}) = \begin{cases} 1, & p(t=1|\boldsymbol{x}) > p(t=-1|\boldsymbol{x}) \\ -1, & p(t=1|\boldsymbol{x}) < p(t=-1|\boldsymbol{x}) \end{cases}$$

Exponential Error Function

For binary classification, consider the exponential error function

$$J(f) = \mathbb{E}[\exp\{-tf(\boldsymbol{x})\}] \qquad t \in \{-1, 1\}$$

It is almost equivalent to the logistic cross entropy loss (for t=+1, t=-1 labels).

Solution 2:

$$J(f) = \int \left[p(t=1|\boldsymbol{x})e^{-f(x)} + p(t=-1|\boldsymbol{x})e^{f(x)}\right]p(\boldsymbol{x})d\boldsymbol{x}$$

$$\frac{\partial J(f)}{\partial f} = -p(t=1|\boldsymbol{x})\exp\{-f(\boldsymbol{x})\} + p(t=-1|\boldsymbol{x})\exp\{f(\boldsymbol{x})\} = 0$$

$$f(\boldsymbol{x}) = \frac{1}{2}\ln\frac{p(t=1|\boldsymbol{x})}{p(t=-1|\boldsymbol{x})} \qquad sign\, f(\boldsymbol{x}) = \begin{cases} 1, & p(t=1|\boldsymbol{x}) > p(t=-1|\boldsymbol{x}) \\ -1, & p(t=1|\boldsymbol{x}) < p(t=-1|\boldsymbol{x}) \end{cases}$$

Bayes error rate $= \arg\max_{t\in\{-1,1\}} p(f(\boldsymbol{x}) = t|\boldsymbol{x})$

Discrete AdaBoost

Consider the exponential error function

$$E = \sum_{n=1}^{N} \exp\{-t_n f_m(\boldsymbol{x}_n)\}$$

$t_n \in \{-1, 1\}$: target values.

$$f_m(x_n) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l y_l(\boldsymbol{x})$$

- Suppose that $y_1(\boldsymbol{x}), \ldots, y_{m-1}(\boldsymbol{x})$ and $\alpha_1, \ldots, \alpha_{m-1}$ are fixed.
- Minimize E with respect to only both α_m and parameters of the model $y_m(\boldsymbol{x})$.

$$E = \sum_{n=1}^{N} \exp\{-t_n f_{m-1}(\boldsymbol{x}_n) - \frac{1}{2} t_n \alpha_m y_m(\boldsymbol{x}_n)\} = \sum_{n=1}^{N} w_n^m \exp\{-\frac{1}{2} t_n \alpha_m y_m(\boldsymbol{x}_n)\}$$

where $w_n^m = \exp\{-t_n f_{m-1}(\boldsymbol{x}_n)\}$ can be treated as constants.

AdaBoost

Let $\mathcal{T}_m$ and $\mathcal{M}_m$ denote the set of data points that are correctly classified and misclassified by $y_m(\boldsymbol{x})$.

$$E = \sum_{n=1}^{N} \exp\{-t_n f_{m-1}(\boldsymbol{x}_n) - \frac{1}{2} t_n \alpha_m y_m(\boldsymbol{x}_n)\} = \sum_{n=1}^{N} w_n^m \exp\{-\frac{1}{2} t_n \alpha_m y_m(\boldsymbol{x}_n)\}$$

Can you rewrite E?

AdaBoost

Let $\mathcal{T}_m$ and $\mathcal{M}_m$ denote the set of data points that are correctly classified and misclassified by $y_m(\boldsymbol{x})$.

$$E = \sum_{n=1}^{N} \exp\{-t_n f_{m-1}(\boldsymbol{x}_n) - \frac{1}{2} t_n \alpha_m y_m(\boldsymbol{x}_n)\} = \sum_{n=1}^{N} w_n^m \exp\{-\frac{1}{2} t_n \alpha_m y_m(\boldsymbol{x}_n)\}$$

Can you rewrite E?

$$E = e^{-\frac{\alpha_m}{2}} \sum_{n \in \mathcal{T}_m} w_n^m + e^{\frac{\alpha_m}{2}} \sum_{n \in \mathcal{M}_m} w_n^m = (e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}) \sum_{n=1}^{N} w_n^m I(y_m(x_n) \neq t_n) + e^{-\frac{\alpha_m}{2}} \sum_{n=1}^{N} w_n^m$$

AdaBoost

Let $\mathcal{T}_m$ and $\mathcal{M}_m$ denote the set of data points that are correctly classified and misclassified by $y_m(\boldsymbol{x})$.

$$E = e^{-\frac{\alpha_m}{2}} \sum_{n \in \mathcal{T}_m} w_n^m + e^{\frac{\alpha_m}{2}} \sum_{n \in \mathcal{M}_m} w_n^m = (e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}) \sum_{n=1}^{N} w_n^m I(y_m(\boldsymbol{x}_n) \neq t_n) + e^{-\frac{\alpha_m}{2}} \sum_{n=1}^{N} w_n^m$$

AdaBoost

Let $\mathcal{T}_m$ and $\mathcal{M}_m$ denote the set of data points that are correctly classified and misclassified by $y_m(\boldsymbol{x})$.

$$E = e^{-\frac{\alpha_m}{2}} \sum_{n \in \mathcal{T}_m} w_n^m + e^{\frac{\alpha_m}{2}} \sum_{n \in \mathcal{M}_m} w_n^m = (e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}) \sum_{n=1}^{N} w_n^m I(y_m(x_n) \neq t_n) + e^{-\frac{\alpha_m}{2}} \sum_{n=1}^{N} w_n^m$$

Minimize E with respect to $y_m(\boldsymbol{x})$, α_m as constants. It is equivalent to minimize ?

AdaBoost

Let $\mathcal{T}_m$ and $\mathcal{M}_m$ denote the set of data points that are correctly classified and misclassified by $y_m(\boldsymbol{x})$.

$$E = e^{-\frac{\alpha_m}{2}} \sum_{n \in \mathcal{T}_m} w_n^m + e^{\frac{\alpha_m}{2}} \sum_{n \in \mathcal{M}_m} w_n^m = (e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}) \sum_{n=1}^{N} w_n^m I(y_m(x_n) \neq t_n) + e^{-\frac{\alpha_m}{2}} \sum_{n=1}^{N} w_n^m$$

Minimize E with respect to $y_m(\boldsymbol{x})$, α_m as constants. It is equivalent to minimize,

$$J_m = \sum_{n=1}^{N} w_n^m I(y_m(\boldsymbol{x}_n) \neq t_n)$$

Weighted error function

AdaBoost

Let $\mathcal{T}_m$ and $\mathcal{M}_m$ denote the set of data points that are correctly classified and misclassified by $y_m(\boldsymbol{x})$.

$$E = e^{-\frac{\alpha_m}{2}} \sum_{n \in \mathcal{T}_m} w_n^m + e^{\frac{\alpha_m}{2}} \sum_{n \in \mathcal{M}_m} w_n^m = (e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}) \sum_{n=1}^{N} w_n^m I(y_m(x_n) \neq t_n) + e^{-\frac{\alpha_m}{2}} \sum_{n=1}^{N} w_n^m$$

Minimize E with respect to $y_m(\boldsymbol{x})$, α_m as constants. It is equivalent to minimize,

$$J_m = \sum_{n=1}^{N} w_n^m I(y_m(\boldsymbol{x}_n) \neq t_n)$$ Weighted error function

Similarly, minimizing E with respect to α_m, we can obtain,

AdaBoost

Let $\mathcal{T}_m$ and $\mathcal{M}_m$ denote the set of data points that are correctly classified and misclassified by $y_m(\boldsymbol{x})$.

$$E = e^{-\frac{\alpha_m}{2}} \sum_{n \in \mathcal{T}_m} w_n^m + e^{\frac{\alpha_m}{2}} \sum_{n \in \mathcal{M}_m} w_n^m = (e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}) \sum_{n=1}^{N} w_n^m I(y_m(x_n) \neq t_n) + e^{-\frac{\alpha_m}{2}} \sum_{n=1}^{N} w_n^m$$

Minimize E with respect to $y_m(\boldsymbol{x})$, α_m as constants. It is equivalent to minimize,

$$J_m = \sum_{n=1}^{N} w_n^m I(y_m(\boldsymbol{x}_n) \neq t_n)$$ Weighted error function

Similarly, minimizing E with respect to α_m, we can obtain, Weighted error rate

$$\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\} \quad \text{where} \quad \epsilon_m = \frac{\sum_{n=1}^{N} w_n^m I(y_m(\boldsymbol{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^m}$$

AdaBoost

Having found $y_m(\boldsymbol{x})$ and α_m, the weights are updated as

$$w_n^m = \exp\{-t_n f_{m-1}(\boldsymbol{x}_n)\} \quad \Rightarrow \quad w_n^{(m+1)} = w_n^m \exp\{-\frac{1}{2} t_n \alpha_m y_m(\boldsymbol{x}_n)\}$$

AdaBoost

Having found $y_m(\boldsymbol{x})$ and α_m, the weights are updated as

$$w_n^m = \exp\{-t_n f_{m-1}(\boldsymbol{x}_n)\} \quad \Rightarrow \quad w_n^{(m+1)} = w_n^m \exp\{-\frac{1}{2} t_n \alpha_m y_m(\boldsymbol{x}_n)\}$$

Making use of the fact that $t_n y_m(\boldsymbol{x}_n) = 1 - 2I(y_m(\boldsymbol{x}_n) \neq t_n)$

The weights can be updated as

$$w_n^{(m+1)} = w_n^m \exp\left\{-\frac{\alpha_m}{2}[1 - 2I(y_m(\boldsymbol{x}_n) \neq t_n)]\right\}$$

$$= w_n^m \exp(-\frac{\alpha_m}{2}) \exp\{\alpha_m I(y_m(\boldsymbol{x}_n) \neq t_n)\} = C w_n^m \exp\{\alpha_m I(y_m(\boldsymbol{x}_n) \neq t_n)\}$$

The constant C can be omitted.

AdaBoost

Once all the base classifiers are trained, new data points are classified by evaluating

$$f_m(\boldsymbol{x}_n) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l y_l(\boldsymbol{x})$$

The factor of 1/2 does not affect the sign and can be omitted, giving

$$Y_M(\boldsymbol{x}) = sign\left(\sum_{m=1}^{M} \alpha_m y_m(\boldsymbol{x})\right)$$

AdaBoost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \tag{14.15}$$

   where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

   (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} \tag{14.16}$$

   and then use these to evaluate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}. \tag{14.17}$$

   (c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp \{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\} \tag{14.18}$$

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x}) \right).$$

AdaBoost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \tag{14.15}$$

   where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

   (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} \tag{14.16}$$

   and then use these to evaluate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}. \tag{14.17}$$

   (c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp \{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\} \tag{14.18}$$

- $w_n^{(m)}$ increases for points that are misclassified.
- $w_n^{(m)}$ decreases for that are correctly classified.
- The successive classifier is forced to put greater emphasis on points that were misclassified by the previous classifier.
- Points that continue to be misclassified by the successive classifier receive greater weight.

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x}) \right).$$

AdaBoost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \tag{14.15}$$

   where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

   (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} \tag{14.16}$$

   and then use these to evaluate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}. \tag{14.17}$$

   (c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp \{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\} \tag{14.18}$$

- $w_n^{(m)}$ increases for points that are misclassified.
- $w_n^{(m)}$ decreases for that are correctly classified.
- The successive classifier is forced to put greater emphasis on points that were misclassified by the previous classifier.
- Points that continue to be misclassified by the successive classifier receive greater weight.

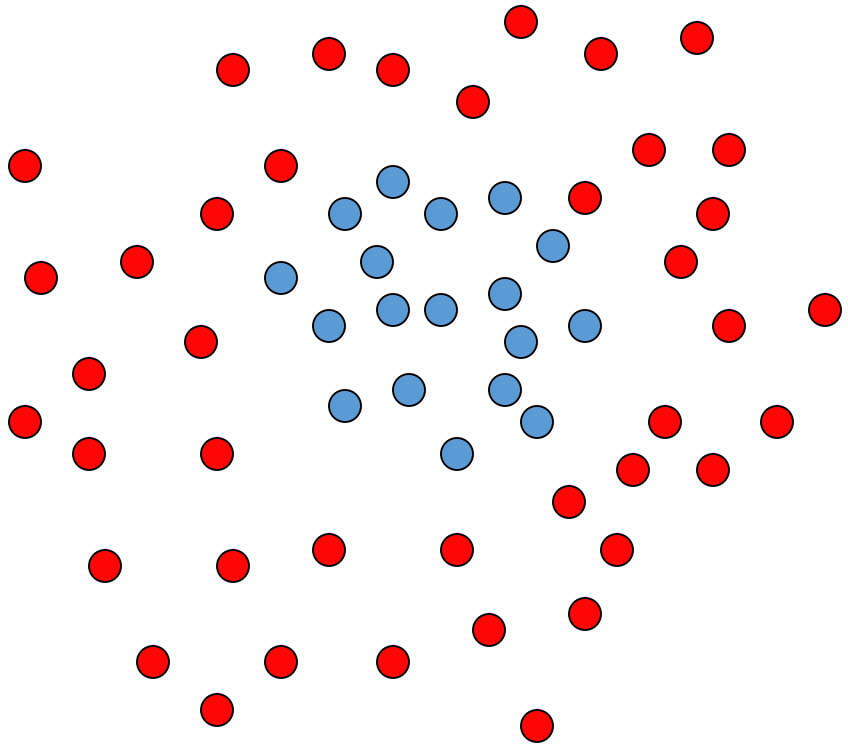
- ϵ_m: weighted error rate of each base classifier.
- α_m: gives larger weight to more accurate classifier.

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x}) \right).$$

Toy Example

- It is a sequential procedure:



Each data point has a class label:

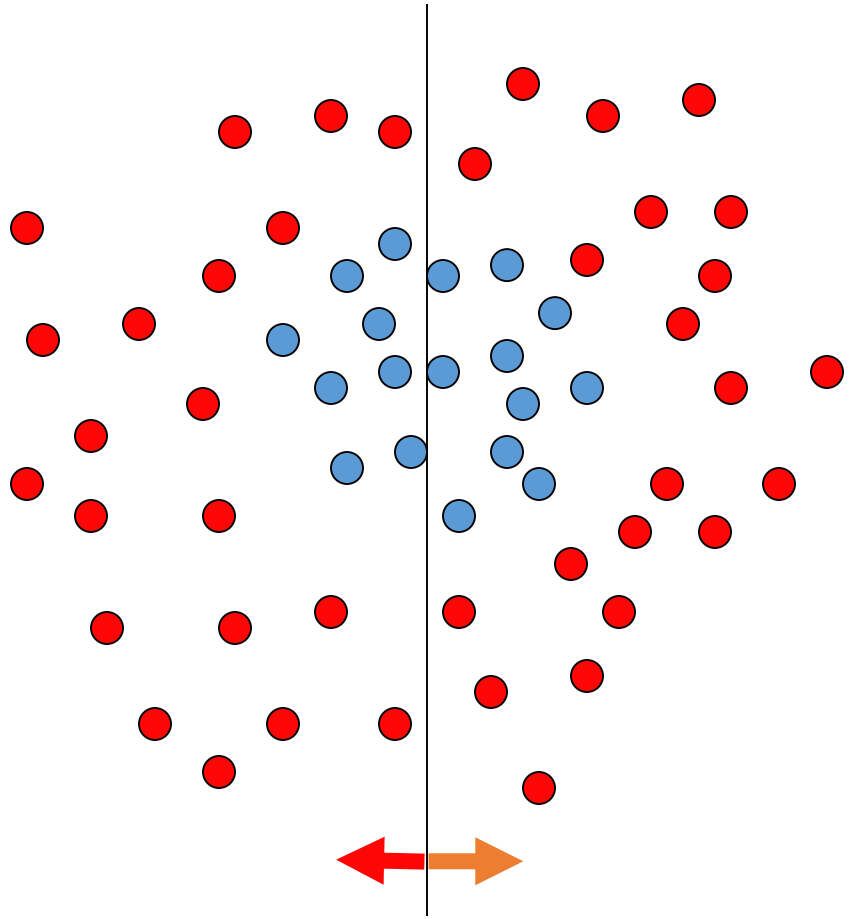
$$t_n = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$$

and a weight:

$$w_n^1 = \frac{1}{N}$$

Toy Example

- Weak learners from the family of lines



Each data point has

a class label:

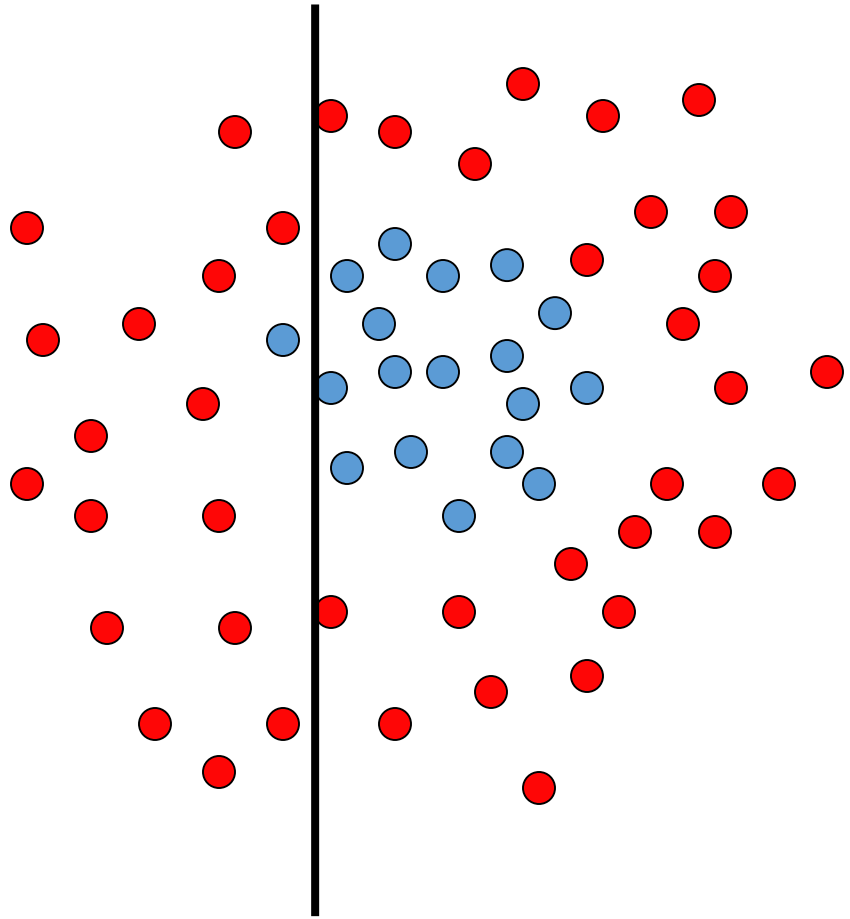
$$t_n = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$$

and a weight:

$$w_n^1 = \frac{1}{N}$$

Toy Example

- This is a ‘weak classifier’: It performs slightly better than chance.



This one seems to be the best

Each data point has a class label:

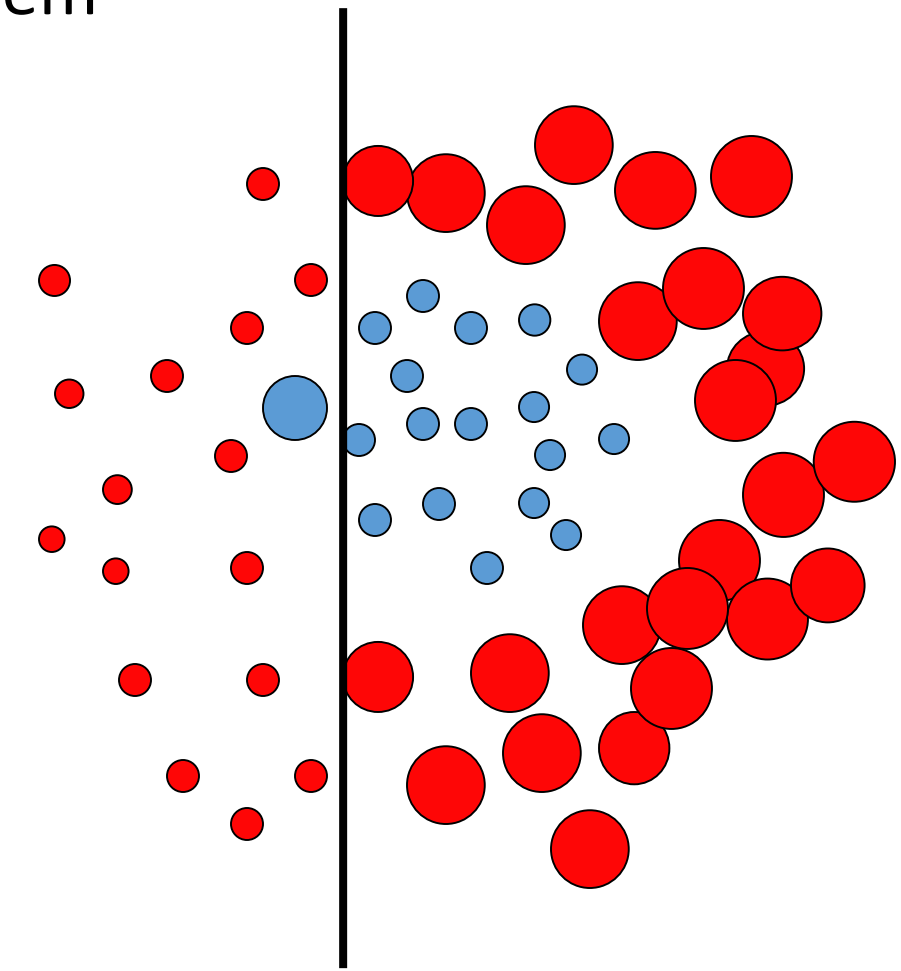
$$t_n = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$$

and a weight:

$$w_n^1 = \frac{1}{N}$$

Toy Example

- We set a new problem



Each data point has

a class label:

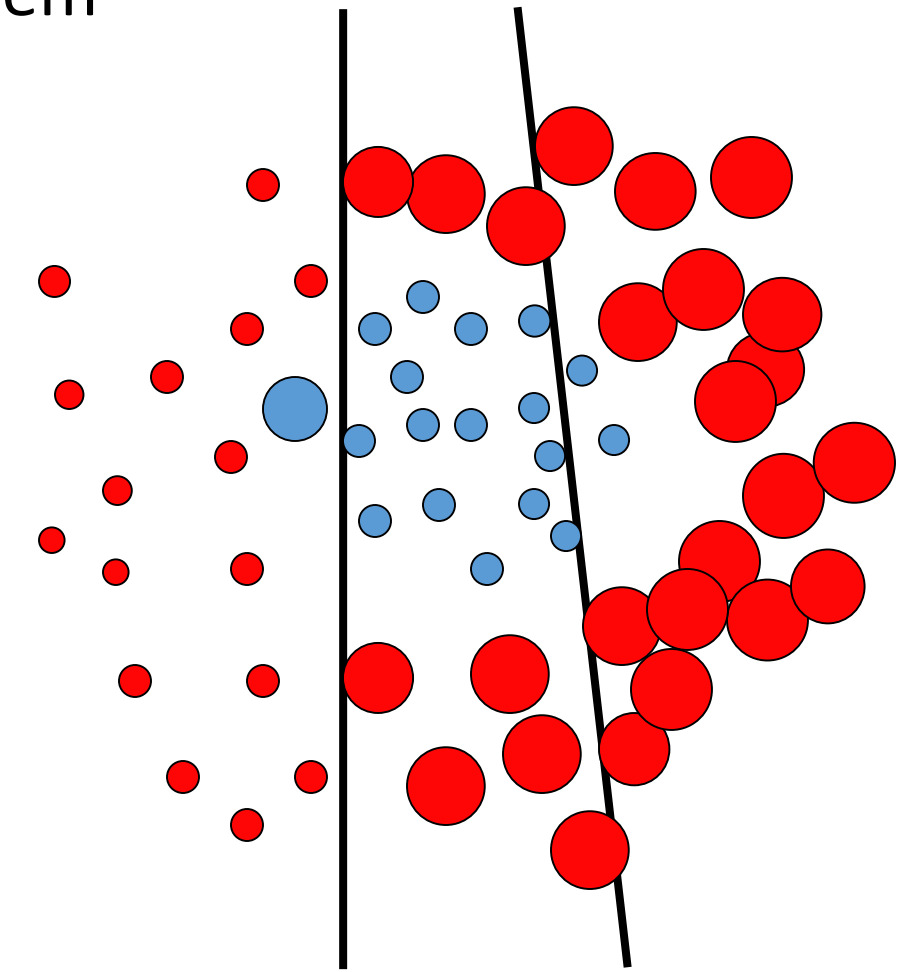
$$t_n = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$$

We update the weights:

$$w_n^2 = w_n^1 \exp\{\alpha_1 I(y_1(x_n) \neq t_n)\}$$

Toy Example

- We set a new problem



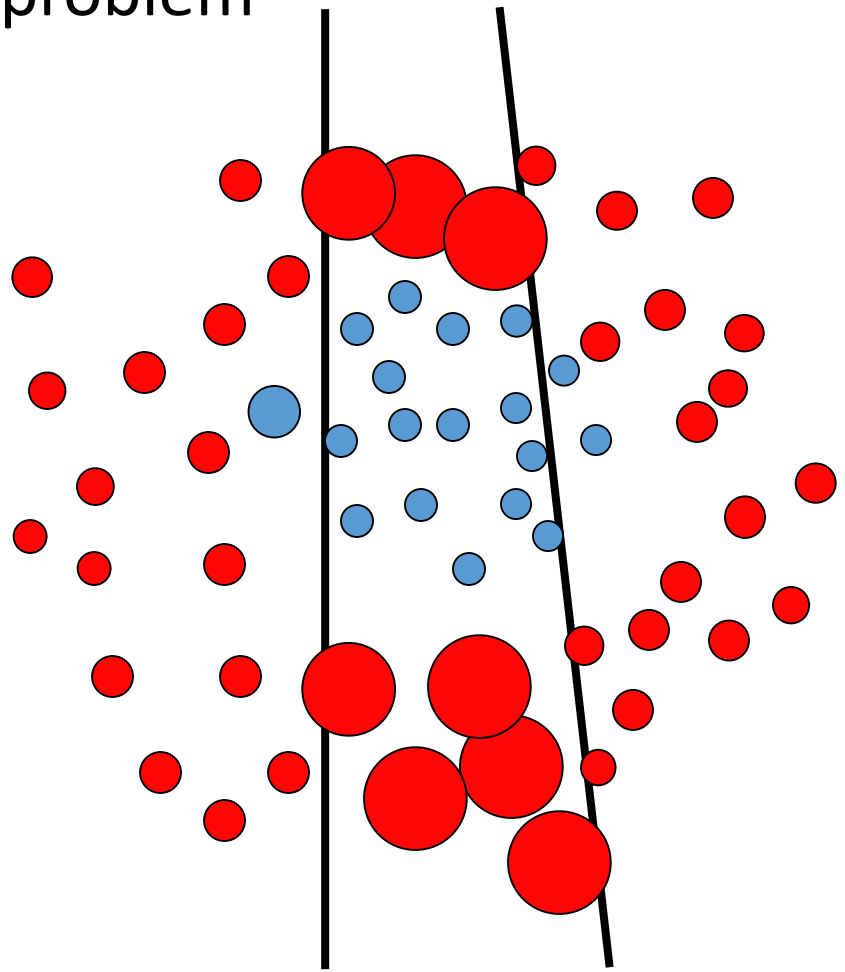
Each data point has a class label:

$$t_n = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$$

- Similarly, we learn another weak classifier

Toy Example

- Continue set a new problem



Reweighting

Each data point has

a class label:

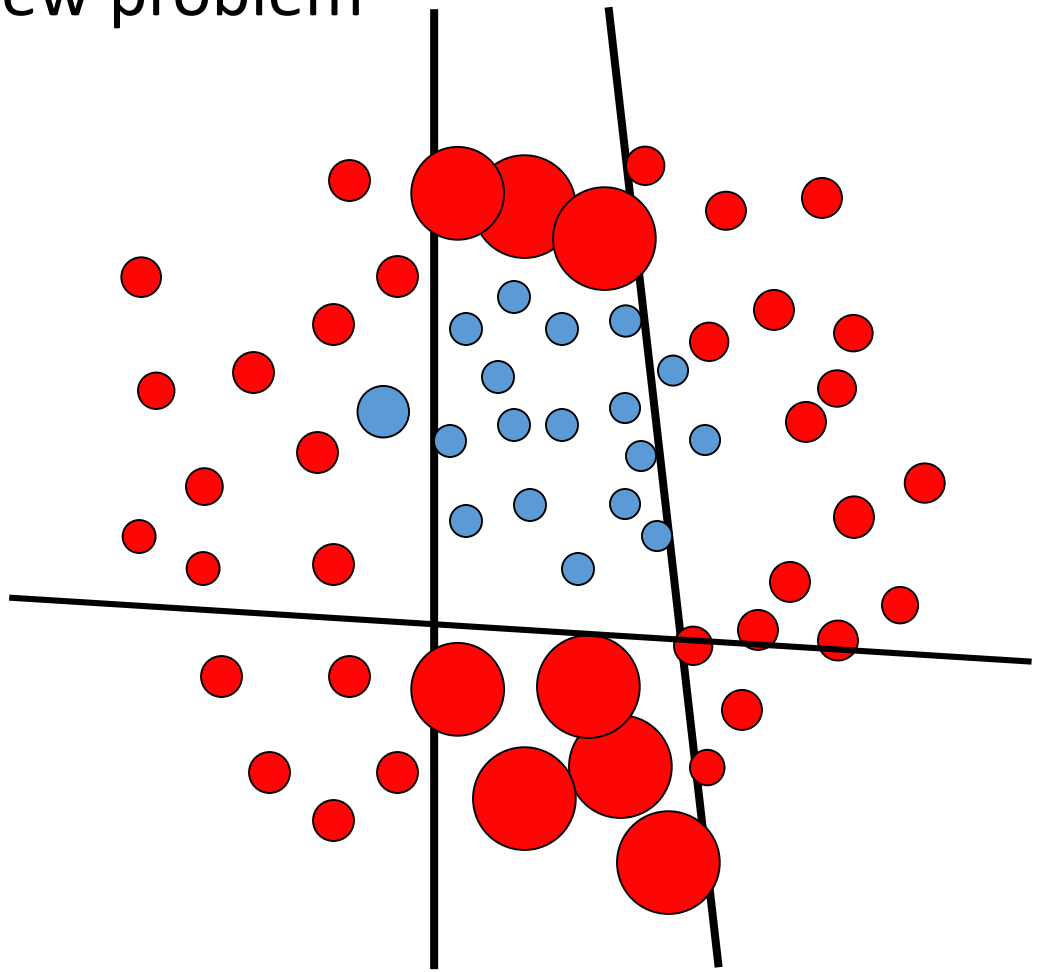
$$t_n = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$$

We update the weights:

$$w_n^3 = w_n^2 \exp\{\alpha_2 I(y_2(x_n) \neq t_n)\}$$

Toy Example

- Continue set a new problem



Each data point has

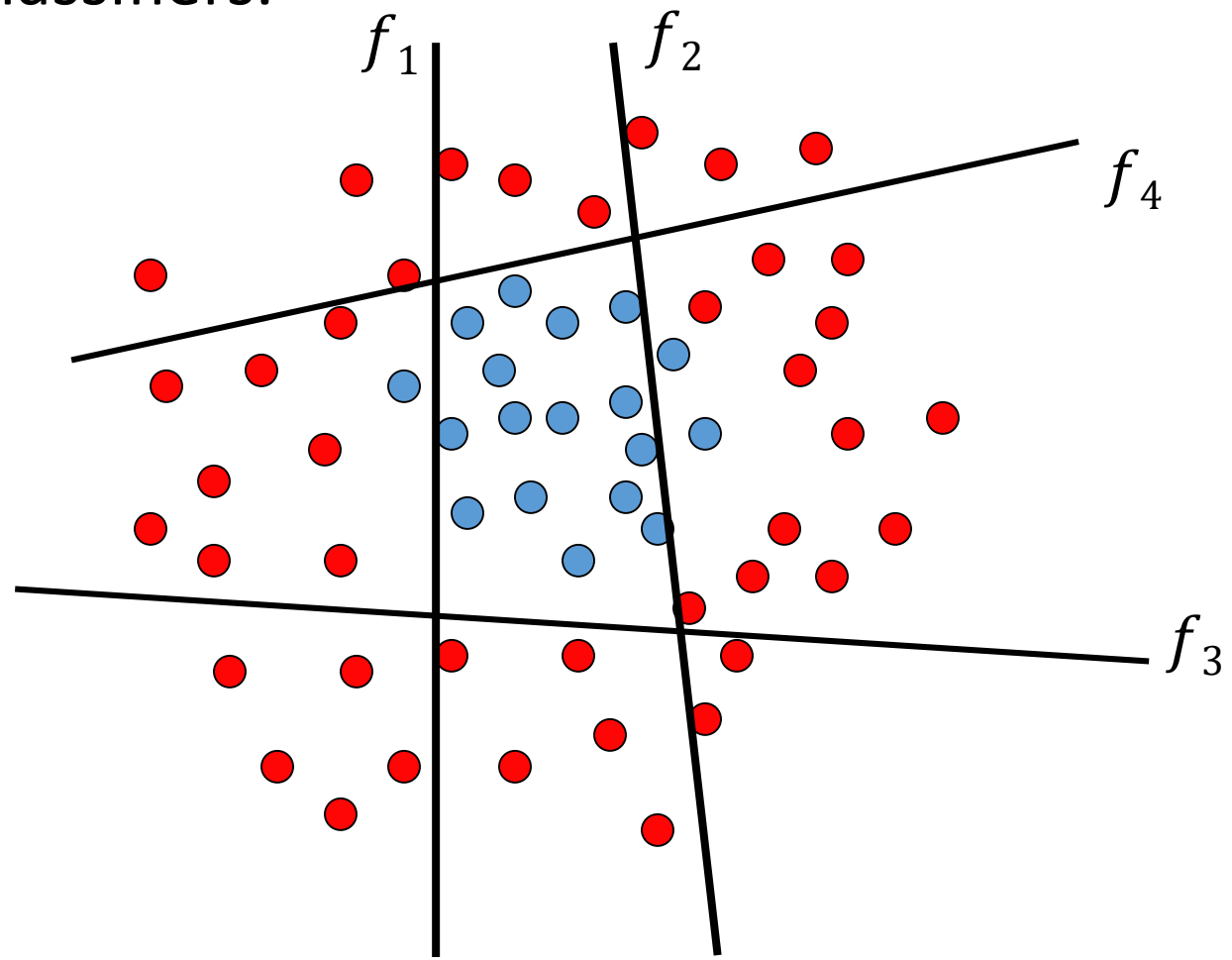
a class label:

$$t_n = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$$

- Similarly, we learn another weak classifier

Toy Example

- The strong (non-linear) classifier is built as the combination of all the weak (linear) classifiers.



AdaBoost

- Significant Advantages
  - Very accurate prediction
  - Very simple (just 10 lines of code as Schapire said)
  - Wide and successful applications
  - Sound theoretical foundation
  - …

Gödel Prize (2003)

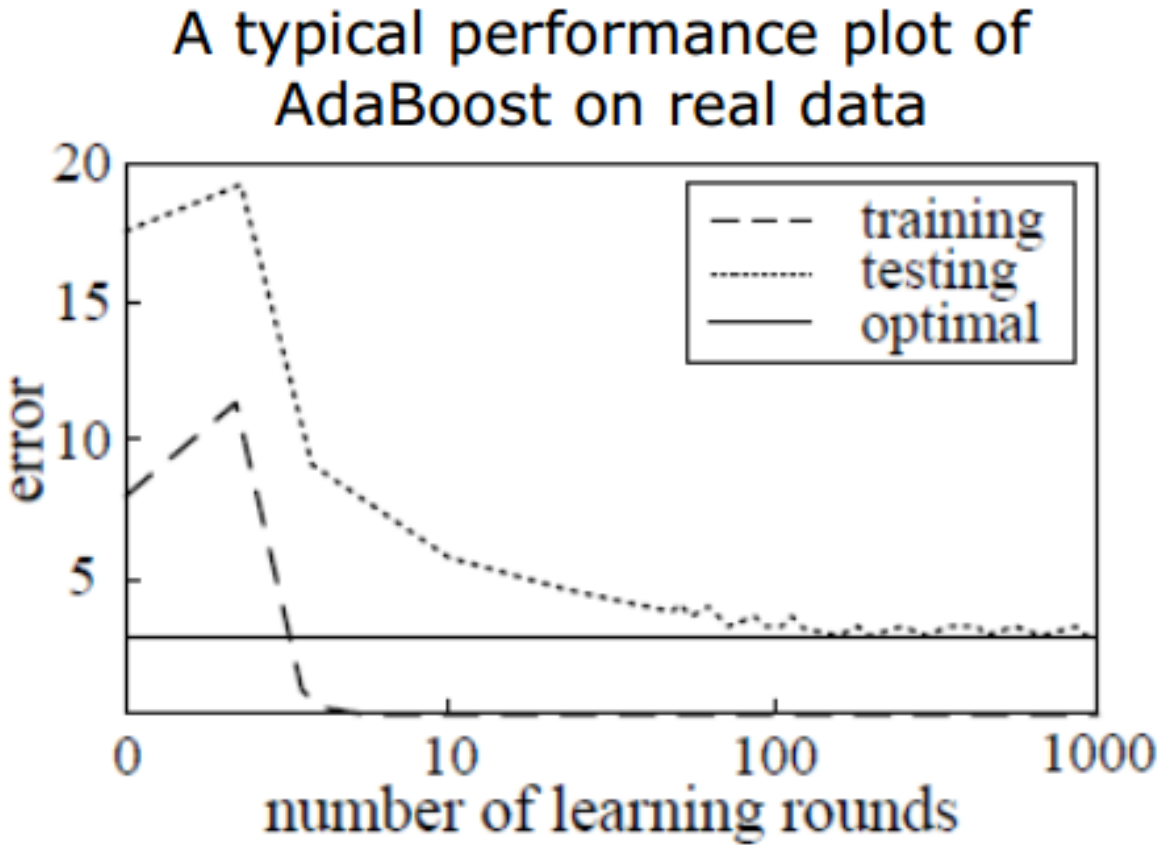
Freund & Schapire, A decision theoretic generalization of on-line learning and an application to Boosting. Journal of Computer and System Sciences, 1997, 55: 119-139.

Why AdaBoost High Impact?

- First, it is simple yet effective.
  - Can be applied to almost all tasks where one wants to apply machine learning techniques.
  - For example, in computer vision, the Viola-Jones detector
    - AdaBoost using harr-like features in a cascade structure
- Second, it generates the Boosting Family of algorithms.
  - A lot of Boosting algorithms: AdaBoost.M1, AdaBoost.MR, FilterBoost, GentleBoost, GradientBoost, MadaBoost, LogitBoost, LPBoost, MultiBoost, RealBoost, RobustBoost,…
- Third, there are sound theoretical results.

The Mystery

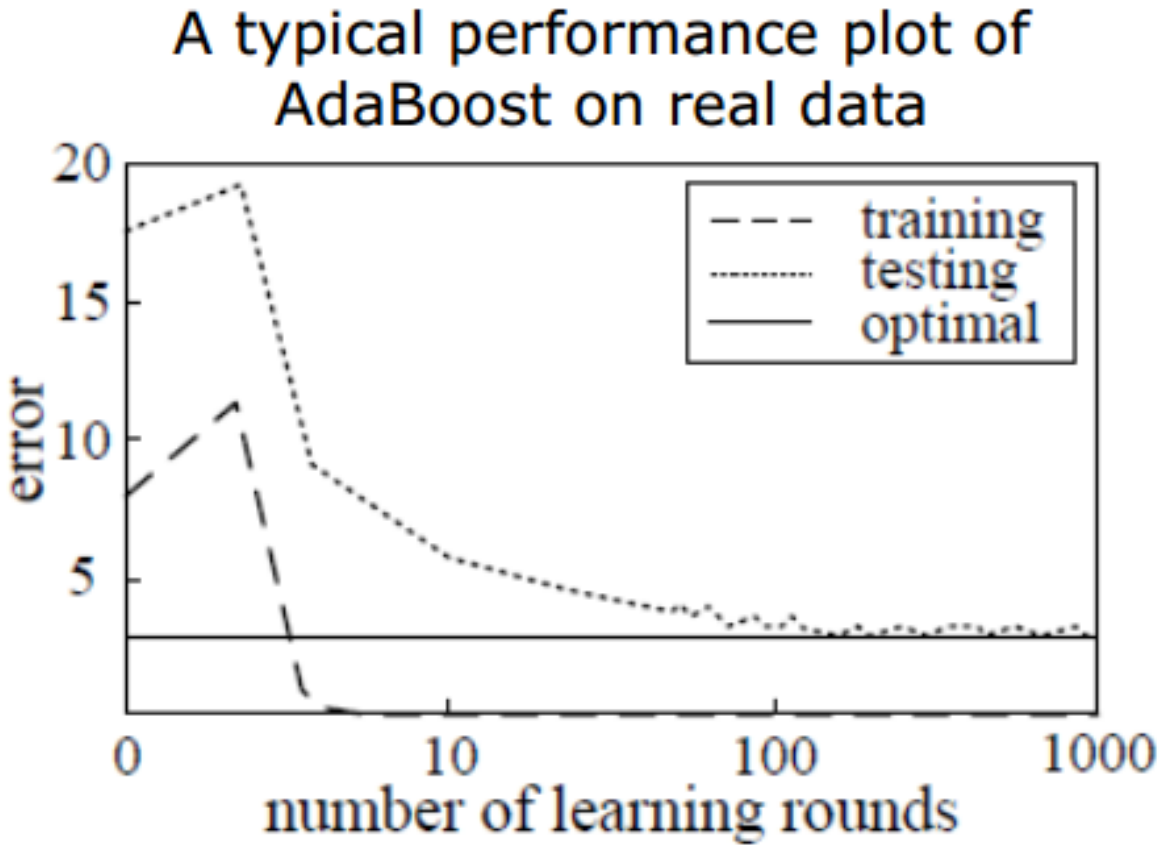
- AdaBoost often does not overfit in real practice!



- AdaBoost drives the training error to zero.
- Further iterations continue to improve test error in many examples.

The Mystery

- AdaBoost often does not overfit in real practice!



- AdaBoost drives the training error to zero.
- Further iterations continue to improve test error in many examples.

Understanding why AdaBoost seems resistant to overfitting is the most fascinating fundamental theoretical issue.

Major Theoretical Efforts

- Margin Theory
  - Started from [Schapire, Freund, Bartlett&Lee, Boosting the margin: A new explanation for the effectiveness of voting methods. Annals of Statistics, 26(5):1651–1686, 1998]
- Statistical View
  - Started from [Friedman, Hastie&Tibshirani. Additive logistic regression: A statistical view of boosting (with discussions). Annals of Statistics, 28(2):337–407, 2000]

Major Theoretical Efforts

- Margin Theory
  - Started from [Schapire, Freund, Bartlett&Lee, Boosting the margin: A new explanation for the effectiveness of voting methods. Annals of Statistics, 26(5):1651–1686, 1998]

- Statistical View
  - Started from [Friedman, Hastie&Tibshirani. Additive logistic regression: A statistical view of boosting (with discussions). Annals of Statistics, 28(2):337–407, 2000]

The biggest issue: The statistical view did not explain why AdaBoost is resistant to overfitting.

Major Theoretical Efforts

- Margin Theory
  - Started from [Schapire, Freund, Bartlett&Lee, Boosting the margin: A new explanation for the effectiveness of voting methods. Annals of Statistics, 26(5):1651–1686, 1998]

Because it is able to increase the ensemble margin even after the training error reaches zero. This explanation is quite intuitive.

- Statistical View
  - Started from [Friedman, Hastie&Tibshirani. Additive logistic regression: A statistical view of boosting (with discussions). Annals of Statistics, 28(2):337–407, 2000]

The biggest issue: The statistical view did not explain why AdaBoost is resistant to overfitting.

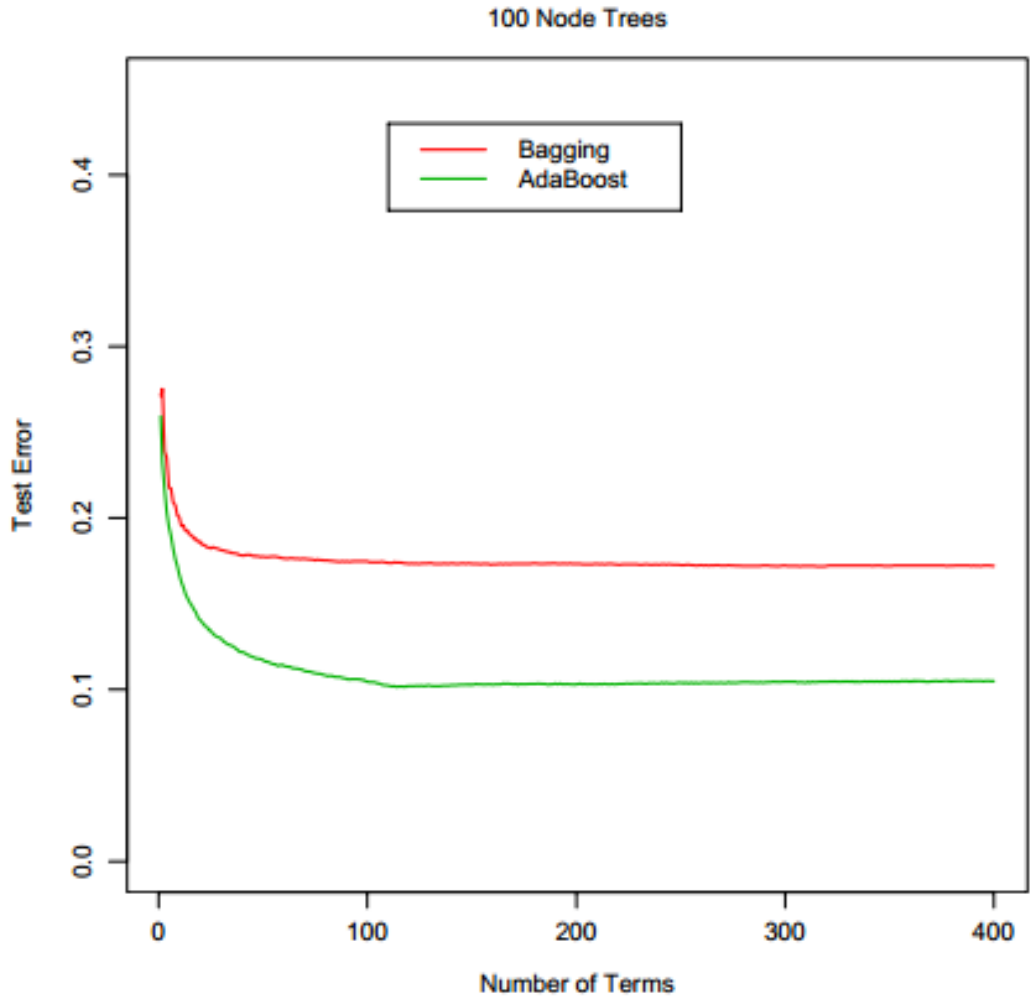
Long March of Margin Theory for AdaBoost

- 1989, [Kearns & Valiant], open problem "weakly learnable"?="strongly learnable"
- 1990, [Schapire], proof by construction, the first Boosting algorithm
- 1993, [Freund], another impractical boosting algorithm by voting
- 1995/97, [Freund & Schapire], AdaBoost

- 1998, [Schapire, Freund, Bartlett & Lee], Margin theory
- 1999, [Breiman], serious doubt by minimum margin bound
- 2006, [Reyzin & Schapire], finding the model complexity issue in exps, emphasizing the importance of margin distribution [ICML'06 best paper]
- 2008, [Wang, Sugiyama, Yang, Zhou & Feng], Emargin bound, believed to be a margin distribution bound
- 2013, [Gao & Zhou], a real margin distribution bound, sheding new insight ; margin theory defensed

Boosting vs Bagging

- 2000 points from Nested Spheres in $\mathbb{R}^{10}$. Leftmost term is a single tree.



Ensemble Methods

- Bagging
- Random Forest
- Boosting (Adaboost)
- Gradient Boosting Decision Trees

Ensemble Methods

- Bagging
- Random Forest
- Boosting (Adaboost)
- Gradient Boosting Decision Trees

What is Gradient Boosting

Gradient Boosting = Gradient Descent + Boosting

Adaboost

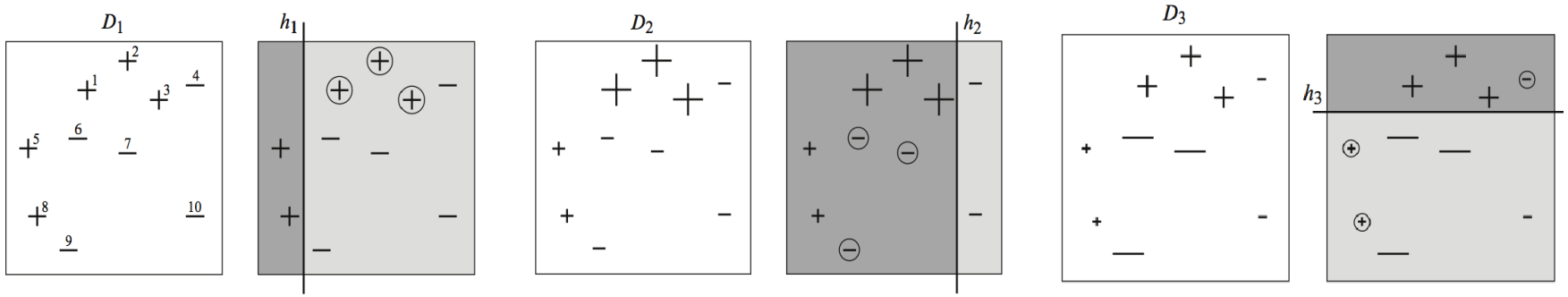


Figure: AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]

- Fit an additive model (ensemble) $f_m(x_n) = \sum_{l=1}^{m} \alpha_l h_l(x)$ in a forward stage-wise manner (sequential).
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- In Adaboost,"shortcomings" are identified by high-weight data points.

What is Gradient Boosting

Gradient Boosting = Gradient Descent + Boosting

Gradient Boosting

- Fit an additive model (ensemble) $f_m(\boldsymbol{x}_n) = \sum_{l=1}^{m} \alpha_l h_l(\boldsymbol{x})$ in a forward stage-wise manner (sequential).
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- In Gradient Boosting, "shortcomings" are identified by gradients.
  - In Adaboost, "shortcomings" are identified by high-weight data points.
- Both high-weight data points and gradients tell us how to improve our model.

A Brief History of Gradient Boosting

- Invent Adaboost, the first successful boosting algorithm
  [Freund et al., 1996, Freund and Schapire, 1997]

- Formulate Adaboost as gradient descent with a special loss function
  [Breiman et al., 1998, Breiman, 1999]

- Generalize Adaboost to Gradient Boosting in order to handle a variety of loss functions
  [Friedman et al., 2000, Friedman, 2001]

Gradient Boosting for Regression

- **Let's play a game…**
- Given $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, and the task is to fit a model $F(\boldsymbol{x})$ to minimize the square loss.

Gradient Boosting for Regression

- **Let's play a game…**
- Given $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, and the task is to fit a model $F(\boldsymbol{x})$ to minimize the square loss.
- Suppose your friend wants to help you and gives you a model F.

Gradient Boosting for Regression

- **Let's play a game…**
- Given $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, and the task is to fit a model $F(\boldsymbol{x})$ to minimize the square loss.
- Suppose your friend wants to help you and gives you a model F.
- You check his model and find the model is good but not perfect. There are some mistakes: $F(\boldsymbol{x}_1) = 0.8$, while $y_1 = 0.9$, and $F(\boldsymbol{x}_2) = 1.4$ while $y_2 = 1.3$ …

Gradient Boosting for Regression

- **Let's play a game…**
- Given $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, and the task is to fit a model $F(\boldsymbol{x})$ to minimize the square loss.
- Suppose your friend wants to help you and gives you a model F.
- You check his model and find the model is good but not perfect. There are some mistakes: $F(\boldsymbol{x}_1) = 0.8$, while $y_1 = 0.9$, and $F(\boldsymbol{x}_2) = 1.4$ while $y_2 = 1.3$ …
- How can you improve this model?

Gradient Boosting for Regression

- **Let's play a game…**
- Given $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, and the task is to fit a model $F(\boldsymbol{x})$ to minimize the square loss.
- Suppose your friend wants to help you and gives you a model F.
- You check his model and find the model is good but not perfect. There are some mistakes: $F(\boldsymbol{x}_1) = 0.8$, while $y_1 = 0.9$, and $F(\boldsymbol{x}_2) = 1.4$ while $y_2 = 1.3$ …
- How can you improve this model?

Rule of the game:

- You are not allowed to remove anything from F or change any parameter in F.
- You can add an additional model (regression tree) h to F, so the new prediction will be $F(\boldsymbol{x}) + h(\boldsymbol{x})$.

Gradient Boosting for Regression

Simple Solution:

You wish to improve the model such that

$$F(\boldsymbol{x}_1) + h(\boldsymbol{x}_1) = y_1$$
$$F(\boldsymbol{x}_2) + h(\boldsymbol{x}_2) = y_2$$
$$\dots$$
$$F(\boldsymbol{x}_n) + h(\boldsymbol{x}_n) = y_n$$

Gradient Boosting for Regression

Simple Solution:

Or, equivalently, you wish

$$h(\boldsymbol{x}_1) = y_1 - F(\boldsymbol{x}_1)$$
$$h(\boldsymbol{x}_2) = y_2 - F(\boldsymbol{x}_2)$$
$$\dots$$
$$h(\boldsymbol{x}_n) = y_n - F(\boldsymbol{x}_n)$$

Gradient Boosting for Regression

Simple Solution:

Or, equivalently, you wish

$$h(\boldsymbol{x}_1) = y_1 - F(\boldsymbol{x}_1)$$
$$h(\boldsymbol{x}_2) = y_2 - F(\boldsymbol{x}_2)$$
$$\dots$$
$$h(\boldsymbol{x}_n) = y_n - F(\boldsymbol{x}_n)$$

- Can any regression tree h achieve this goal perfectly?

Gradient Boosting for Regression

Simple Solution:

Or, equivalently, you wish

$$h(\boldsymbol{x}_1) = y_1 - F(\boldsymbol{x}_1)$$
$$h(\boldsymbol{x}_2) = y_2 - F(\boldsymbol{x}_2)$$
$$\dots$$
$$h(\boldsymbol{x}_n) = y_n - F(\boldsymbol{x}_n)$$

- Can any regression tree h achieve this goal perfectly? Maybe not....

Gradient Boosting for Regression

Simple Solution:

Or, equivalently, you wish

$$h(\boldsymbol{x}_1) = y_1 - F(\boldsymbol{x}_1)$$
$$h(\boldsymbol{x}_2) = y_2 - F(\boldsymbol{x}_2)$$
$$\dots$$
$$h(\boldsymbol{x}_n) = y_n - F(\boldsymbol{x}_n)$$

- Can any regression tree h achieve this goal perfectly? Maybe not....
- But some regression tree might be able to do this approximately. How?

Gradient Boosting for Regression

Simple Solution:

Or, equivalently, you wish

$$h(\boldsymbol{x}_1) = y_1 - F(\boldsymbol{x}_1)$$
$$h(\boldsymbol{x}_2) = y_2 - F(\boldsymbol{x}_2)$$
$$\dots$$
$$h(\boldsymbol{x}_n) = y_n - F(\boldsymbol{x}_n)$$

- Can any regression tree h achieve this goal perfectly? Maybe not....
- But some regression tree might be able to do this approximately. How?
- Just fit a regression tree h to data $(\boldsymbol{x}_1, y_1 - F(\boldsymbol{x}_1)), (\boldsymbol{x}_2, y_2 - F(\boldsymbol{x}_2)), \dots, (\boldsymbol{x}_n, y_n - F(\boldsymbol{x}_n))$
- Congratulations, you get a better model!

Gradient Boosting for Regression

Simple Solution:

- $y_i - F(\boldsymbol{x}_i)$ are called **residuals**. These are the parts that existing model F cannot do well.
- The role of h is to compensate the shortcoming of existing model F.

Gradient Boosting for Regression

Simple Solution:

- $y_i - F(\boldsymbol{x}_i)$ are called **residuals**. These are the parts that existing model F cannot do well.
- The role of h is to compensate the shortcoming of existing model F.
- If the new model $F + h$ is still not satisfactory, THEN???

Gradient Boosting for Regression

Simple Solution:

- $y_i - F(\boldsymbol{x}_i)$ are called **residuals**. These are the parts that existing model F cannot do well.
- The role of h is to compensate the shortcoming of existing model F.
- If the new model $F + h$ is still not satisfactory, we can add another regression tree...

Gradient Boosting for Regression

Simple Solution:

- $y_i - F(\boldsymbol{x}_i)$ are called **residuals**. These are the parts that existing model F cannot do well.
- The role of h is to compensate the shortcoming of existing model F.
- If the new model $F + h$ is still not satisfactory, we can add another regression tree...

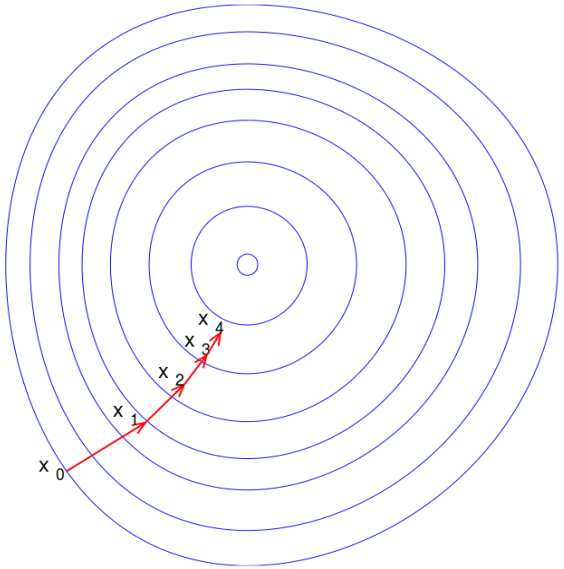
How is this related to gradient descent?

Gradient Boosting for Regression

Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$



Gradient Boosting for Regression

How is this related to gradient descent?

Gradient Boosting for Regression

How is this related to gradient descent?

Loss function $L(y, F(\boldsymbol{x})) = (y - F(\boldsymbol{x}))^2/2$

We want to minimize $J = \sum_i L(y_i, F(\boldsymbol{x}_i))$ by adjusting $F(\boldsymbol{x}_1), F(\boldsymbol{x}_2), \ldots, F(\boldsymbol{x}_n)$.

Gradient Boosting for Regression

How is this related to gradient descent?

Loss function $L(y, F(\boldsymbol{x})) = (y - F(\boldsymbol{x}))^2/2$

We want to minimize $J = \sum_i L(y_i, F(\boldsymbol{x}_i))$ by adjusting $F(\boldsymbol{x}_1), F(\boldsymbol{x}_2), \ldots, F(\boldsymbol{x}_n)$.

Notice that $F(\boldsymbol{x}_1), F(\boldsymbol{x}_2), \ldots, F(\boldsymbol{x}_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives,

$$\frac{\partial J}{\partial F(\boldsymbol{x}_i)} = \frac{\partial \sum_i L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)} = \frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)} = F(\boldsymbol{x}_i) - y_i$$

Gradient Boosting for Regression

How is this related to gradient descent?

Loss function $L(y, F(\boldsymbol{x})) = (y - F(\boldsymbol{x}))^2/2$

We want to minimize $J = \sum_i L(y_i, F(\boldsymbol{x}_i))$ by adjusting $F(\boldsymbol{x}_1), F(\boldsymbol{x}_2), \ldots, F(\boldsymbol{x}_n)$.

Notice that $F(\boldsymbol{x}_1), F(\boldsymbol{x}_2), \ldots, F(\boldsymbol{x}_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives,

$$\frac{\partial J}{\partial F(\boldsymbol{x}_i)} = \frac{\partial \sum_i L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)} = \frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)} = F(\boldsymbol{x}_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(\boldsymbol{x}_i) = -\frac{\partial J}{\partial F(\boldsymbol{x}_i)}$$

Gradient Boosting for Regression

How is this related to gradient descent?

$$F(\boldsymbol{x}_i) \coloneqq F(\boldsymbol{x}_i) + h(\boldsymbol{x}_i)$$

$$F(\boldsymbol{x}_i) \coloneqq F(\boldsymbol{x}_i) + y_i - F(\boldsymbol{x}_i)$$

$$F(\boldsymbol{x}_i) \coloneqq F(\boldsymbol{x}_i) - 1 \cdot \frac{\partial J}{\partial F(\boldsymbol{x}_i)}$$

$$\boldsymbol{\theta}_i \coloneqq \boldsymbol{\theta}_i - \rho \frac{\partial J}{\partial \boldsymbol{\theta}_i}$$

Gradient Boosting for Regression

How is this related to gradient descent?

For regression with **square loss**,

$$\textit{residual} \Leftrightarrow \textit{negative gradient}$$

$$\textit{fit } h \textit{ to residual} \Leftrightarrow \textit{fit } h \textit{ to negative gradient}$$

$$\textit{update } F \textit{ based on residual} \Leftrightarrow \textit{update } F \textit{ based on negative gradient}$$

So we are actually updating our model using **gradient descent**!

Gradient Boosting for Regression

How is this related to gradient descent?

For regression with **square loss**,

$$\textit{residual} \Leftrightarrow \textit{negative gradient}$$

$$\textit{fit } h \textit{ to residual} \Leftrightarrow \textit{fit } h \textit{ to negative gradient}$$

$$\textit{update } F \textit{ based on residual} \Leftrightarrow \textit{update } F \textit{ based on negative gradient}$$

So we are actually updating our model using **gradient descent**!

It turns out that the concept of **gradients** is more general and useful than the concept of **residuals**.

Gradient Boosting for Regression

Regression with Square Loss

Let us summarize the algorithm we just derived using the concept of gradients.

Negative gradient:

$$-g(\boldsymbol{x}_i) = -\frac{\partial L\big(y_i, F(\boldsymbol{x}_i)\big)}{\partial F(\boldsymbol{x}_i)} = y_i - F(\boldsymbol{x}_i)$$

Gradient Boosting for Regression

Regression with Square Loss

Let us summarize the algorithm we just derived using the concept of gradients.

Negative gradient:

$$-g(\boldsymbol{x}_i) = -\frac{\partial L\big(y_i, F(\boldsymbol{x}_i)\big)}{\partial F(\boldsymbol{x}_i)} = y_i - F(\boldsymbol{x}_i)$$

start with an initial model, say $F(\boldsymbol{x}) = \frac{\sum_i y_i}{n}$,

iterate until converge:

- calculate negative gradients $-g(\boldsymbol{x}_i)$
- fit a regression tree h to negative gradients $-g(\boldsymbol{x}_i)$
- $F := F + \rho h$, where $\rho = 1$

Gradient Boosting for Regression

Regression with Square Loss

Let us summarize the algorithm we just derived using the concept of gradients.

Negative gradient:

$$-g(\boldsymbol{x}_i) = -\frac{\partial L\big(y_i, F(\boldsymbol{x}_i)\big)}{\partial F(\boldsymbol{x}_i)} = y_i - F(\boldsymbol{x}_i)$$

start with an initial model, say $F(\boldsymbol{x}) = \frac{\sum_i y_i}{n}$,

iterate until converge:

- calculate negative gradients $-g(\boldsymbol{x}_i)$
- fit a regression tree h to negative gradients $-g(\boldsymbol{x}_i)$
- $F := F + \rho h$, where $\rho = 1$

The benefit of formulating this algorithm using gradients is that it allows us to consider other loss functions and derive the corresponding algorithms in the same way.

Gradient Boosting for Regression

Why do we need to consider other loss functions?
Is not square loss good enough?

Gradient Boosting for Regression

Loss Functions for Regression Problem

Square Loss:

- Easy to deal with mathematically
- Not robust to outliers

  Outliers are heavily punished because the error is squared. Example:

y_i	0.5	1.2	2	5*
$F(\boldsymbol{x}_i)$	0.6	1.4	1.5	1.7
$L = (y - F)^2/2$	0.005	0.02	0.125	5.445

Gradient Boosting for Regression

Loss Functions for Regression Problem

Square Loss:

- Easy to deal with mathematically
- Not robust to outliers

  Outliers are heavily punished because the error is squared. Example:

y_i	0.5	1.2	2	5*
$F(\boldsymbol{x}_i)$	0.6	1.4	1.5	1.7
$L = (y - F)^2/2$	0.005	0.02	0.125	5.445

Consequence?

Gradient Boosting for Regression

Loss Functions for Regression Problem

Square Loss:

- Easy to deal with mathematically
- Not robust to outliers

  Outliers are heavily punished because the error is squared. Example:

y_i	0.5	1.2	2	5*
$F(\boldsymbol{x}_i)$	0.6	1.4	1.5	1.7
$L = (y - F)^2/2$	0.005	0.02	0.125	5.445

Consequence?

Pay too much attention to outliers. Try hard to incorporate outliers into the model.

Degrade the overall performance.

Gradient Boosting for Regression

Loss Functions for Regression Problem

- Absolute Loss (more robust to outliers)

$$L(y, F) = |y - F|$$

- Huber Loss (more robust to outliers)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \le \delta \\ \delta\,(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

y_i	0.5	1.2	2	5*
$F(\boldsymbol{x}_i)$	0.6	1.4	1.5	1.7
Square loss	0.005	0.02	0.125	5.445
Absolute loss	0.1	0.2	0.5	3.3
Huber loss ($\delta = 0.5$)	0.005	0.02	0.125	1.525

Gradient Boosting for Regression

Regression with Absolute Loss

Negative Gradient:

$$-g(\boldsymbol{x}_i) = -\frac{\partial L\big(y_i, F(\boldsymbol{x}_i)\big)}{\partial F(\boldsymbol{x}_i)} = sign(y_i - F(\boldsymbol{x}_i))$$

start with an initial model, say $F(\boldsymbol{x}) = \frac{\sum_i y_i}{n}$,

iterate until converge:

- calculate negative gradients $-g(\boldsymbol{x}_i)$
- fit a regression tree h to negative gradients $-g(\boldsymbol{x}_i)$
- $F := F + \rho h$, where $\rho = 1$

Gradient Boosting for Regression

Regression with Huber Loss

Negative Gradient:

$$-g(\boldsymbol{x}_i) = -\frac{\partial L\big(y_i, F(\boldsymbol{x}_i)\big)}{\partial F(\boldsymbol{x}_i)} = \begin{cases} y_i - F(\boldsymbol{x}_i) & |y_i - F(\boldsymbol{x}_i)| \le \delta \\ \delta\, sign(y_i - F(\boldsymbol{x}_i)) & |y_i - F(\boldsymbol{x}_i)| > \delta \end{cases}$$

start with an initial model, say $F(\boldsymbol{x}) = \frac{\sum_i y_i}{n}$,

iterate until converge:

- calculate negative gradients $-g(\boldsymbol{x}_i)$
- fit a regression tree h to negative gradients $-g(\boldsymbol{x}_i)$
- $F := F + \rho h$, where $\rho = 1$

Gradient Boosting for Regression

Regression with Any Differentiable Loss Function L

General Procedure

start with an initial model, say $F(\boldsymbol{x}) = \frac{\sum_i y_i}{n}$,

iterate until converge:

- calculate negative gradients $-g(\boldsymbol{x}_i)$
- fit a regression tree h to negative gradients $-g(\boldsymbol{x}_i)$
- $F := F + \rho h$, where $\rho = 1$

Gradient Boosting for Regression

Regression with Any Differentiable Loss Function L

General Procedure

start with an initial model, say $F(\boldsymbol{x}) = \frac{\sum_i y_i}{n}$,

iterate until converge:

- calculate negative gradients $-g(\boldsymbol{x}_i)$
- fit a regression tree h to negative gradients $-g(\boldsymbol{x}_i)$
- $F := F + \rho h$, where $\rho = 1$

In general,

negative gradients $\nLeftrightarrow$ *residuals*

Gradient Boosting for Regression

Regression with Any Differentiable Loss Function L

General Procedure

start with an initial model, say $F(\boldsymbol{x}) = \frac{\sum_i y_i}{n}$,

iterate until converge:

- calculate negative gradients $-g(\boldsymbol{x}_i)$
- fit a regression tree h to negative gradients $-g(\boldsymbol{x}_i)$
- $F := F + \rho h$, where $\rho = 1$

In general,

negative gradients $\Leftrightarrow$ *residuals*

We should follow negative gradients rather than residuals. Why?

Gradient Boosting for Regression

Negative Gradient vs Residual: An Example Huber loss

$$L(y,F) = \begin{cases} \frac{1}{2}(y-F)^2 & |y-F| \le \delta \\ \delta\,(|y-F| - \delta/2) & |y-F| > \delta \end{cases}$$

- Update by Negative Gradient:

$$h(\boldsymbol{x}_i) = -g(\boldsymbol{x}_i) = \begin{cases} y_i - F(\boldsymbol{x}_i) & |y_i - F(\boldsymbol{x}_i)| \le \delta \\ \delta\, sign(y_i - F(\boldsymbol{x}_i)) & |y_i - F(\boldsymbol{x}_i)| > \delta \end{cases}$$

- Update by Residual: $h(\boldsymbol{x}_i) = y_i - F(\boldsymbol{x}_i)$

Difference: negative gradient pays less attention to outliers.

Gradient Boosting for Regression

Mini Summary

- Fit an additive model (ensemble) $f_m(\boldsymbol{x}_n) = \sum_{l=1}^{m} \alpha_l h_l(\boldsymbol{x})$ in a forward stage-wise manner (sequential).
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
- In Gradient Boosting, "shortcomings" are identified by negative gradients.
- For any loss function, we can derive a gradient boosting algorithm.
- Absolute loss and Huber loss are more robust to outliers than square loss.

Gradient Boosting Decision Tree

- Gradient Boosting Decision Tree (Many aliases GBRT, boosted trees, GBM): Boosting with decision trees.
- $f_m(x)$ is a decision tree model.

1,536 test data instances

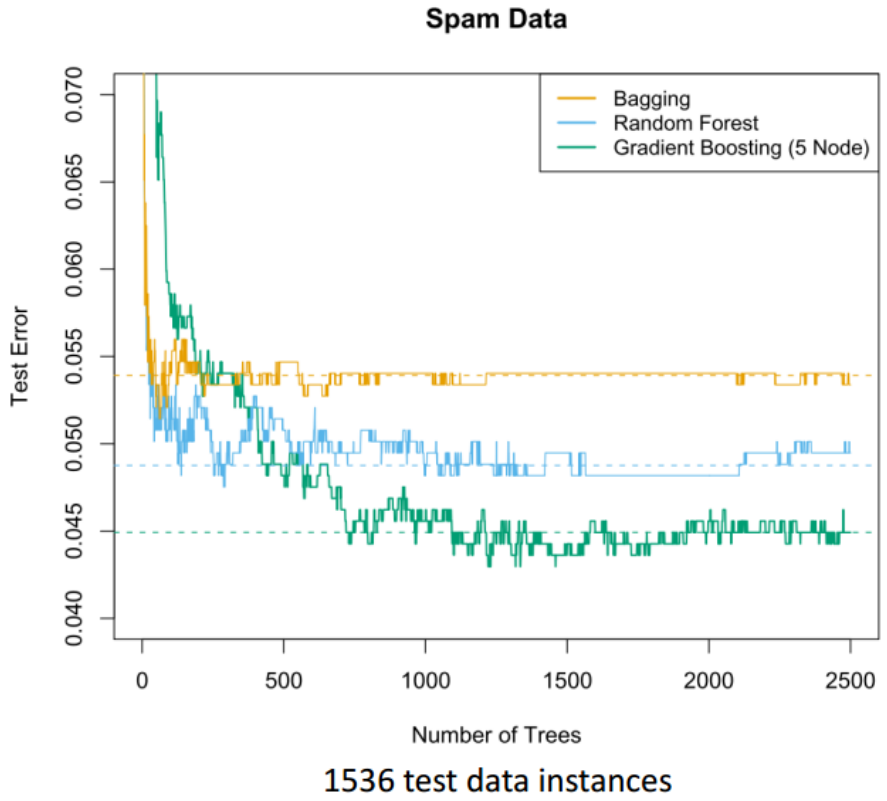


Fig. 15.1 of Hastie et al. The elements of statistical learning.

Trees Ensemble

- Assume we have K trees, $\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$

- Objective function

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss **Regularization**

Complexity of the trees

E.g., square loss $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$

L2 norm $\Omega(f_t) = \lambda \|f_t\|^2$

L1 norm $\Omega(f_t) = \lambda \|f_t\|_1$

Trees Ensemble

- Assume we have K trees, $\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$

- Objective function

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss **Regularization**

Complexity of the trees

E.g., square loss $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$

L2 norm $\Omega(f_t) = \lambda \|f_t\|^2$

L1 norm $\Omega(f_t) = \lambda \|f_t\|_1$

- We cannot use methods such as SGD, to find f_t

WHY?

Trees Ensemble

- Assume we have K trees, $\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$

- Objective function

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss **Regularization**
Complexity of the trees

E.g., square loss $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
L2 norm $\Omega(f_t) = \lambda \|f_t\|^2$
L1 norm $\Omega(f_t) = \lambda \|f_t\|_1$

- We cannot use methods such as SGD, to find f_t (since they are trees, instead of just numerical vectors)

Trees Ensemble

Solution: Additive Training (boosting)

Additive Training

- Start from constant prediction, add a new function each time

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = \sum_{m=1}^{1} f_m(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\vdots$$

$$\hat{y}_i^{(t)} = \sum_{m=1}^{t} f_m(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Model at training round t

New function (tree)

Keep functions added in previous round

- How do we decide which f_t to add?

Additive Training

- Start from constant prediction, add a new function each time

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = \sum_{m=1}^{1} f_m(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\vdots$$

$$\hat{y}_i^{(t)} = \sum_{m=1}^{t} f_m(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Model at training round t

New function (tree)

Keep functions added in previous round

- How do we decide which f_t to add?
  - Optimize the objective!!

Additive Training

- The prediction at round t is $\hat{y}_i^{(t)} = \sum_{m=1}^{t} f_m(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$
- Objective w.r.t f_t:

What is this?

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + const$$

Additive Training

- The prediction at round t is $\hat{y}_i^{(t)} = \sum_{m=1}^{t} f_m(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$
- Objective w.r.t f_t:

What is this?

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + const$$

- Goal: find f_t to minimize this
- Grow the next tree f_t to minimize the loss function $Obj^{(t)}$, including the tree penalty $\Omega(f_t)$

$$\min_{f_t} Obj^{(t)}$$

Taylor Series Approximation

- Objective w.r.t f_t, $Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$
- Let's define the gradients,

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \qquad h_i = \partial^2{}_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

Taylor Series Approximation

- Objective w.r.t f_t, $Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$
- Let's define the gradients,

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \qquad h_i = \partial^2{}_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

- If you are not comfortable with this, think of the square loss,

$$g_i = \partial_{\hat{y}^{(t-1)}} \left(y_i - \hat{y}_i^{(t-1)}\right)^2 = 2(\hat{y}_i^{(t-1)} - y_i) \qquad h_i = \partial^2_{\hat{y}^{(t-1)}} \left(y_i - \hat{y}_i^{(t-1)}\right)^2 = 2$$

Taylor Series Approximation

- Objective w.r.t f_t, $Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$
- Let's define the gradients,

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \qquad h_i = \partial^2{}_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

- If you are not comfortable with this, think of the square loss,

$$g_i = \partial_{\hat{y}^{(t-1)}} \left(y_i - \hat{y}_i^{(t-1)}\right)^2 = 2(\hat{y}_i^{(t-1)} - y_i) \qquad h_i = \partial^2_{\hat{y}^{(t-1)}} \left(y_i - \hat{y}_i^{(t-1)}\right)^2 = 2$$

- Taylor series, $q(x + \Delta x) = q(x) + \frac{q'(x)}{1!}\Delta x + \frac{q''(x)}{2!}(\Delta x)^2$

$$q(s) = l(y_i, s) \qquad s \coloneqq \hat{y}_i^{(t-1)} \qquad \Delta s \coloneqq f_t(x_i)$$

Taylor Series Approximation

- Objective w.r.t f_t, $Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$
- Let's define the gradients,

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \qquad h_i = \partial^2{}_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

- Taylor series, $q(x + \Delta x) = q(x) + \frac{q'(x)}{1!}\Delta x + \frac{q''(x)}{2!}(\Delta x)^2$

$$q(s) = l(y_i, s) \qquad s \coloneqq \hat{y}_i^{(t-1)} \qquad \Delta s \coloneqq f_t(x_i)$$

- Approximation

$$Obj^{(t)} \approx \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

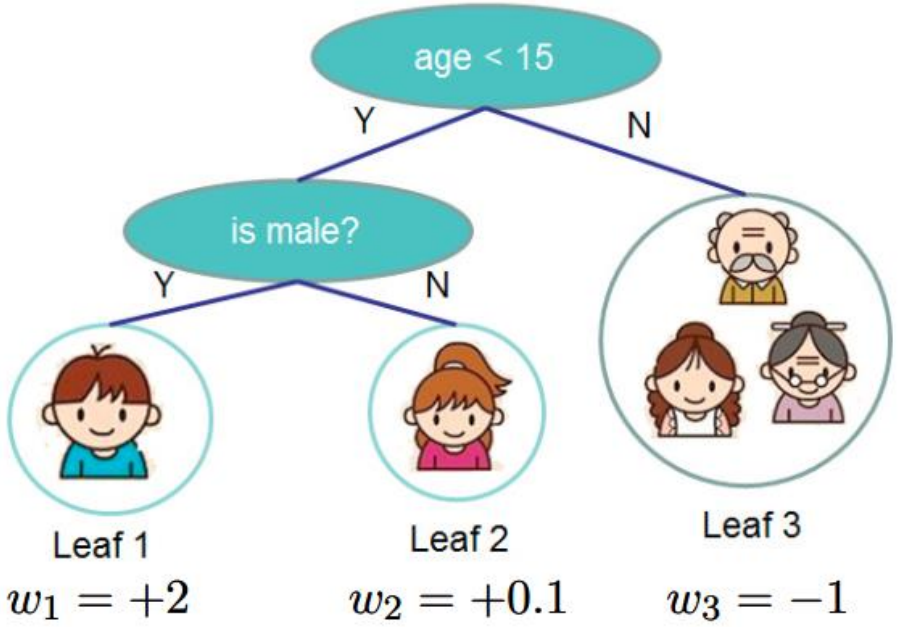
Refine the Definition of Tree

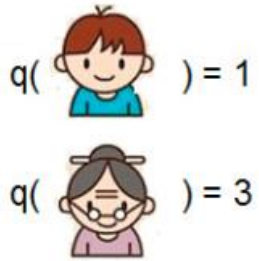
- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf:

The leaf weight of the tree The structure of the tree

$$f_t(x) = w_{q(x)}, w \in \mathbb{R}^T, q: \mathbb{R}^d \mapsto \{1,2,\dots,T\}$$

T:# leaves
w: the weight of the leave
$q(x)$: mapping the input to one of the leave.



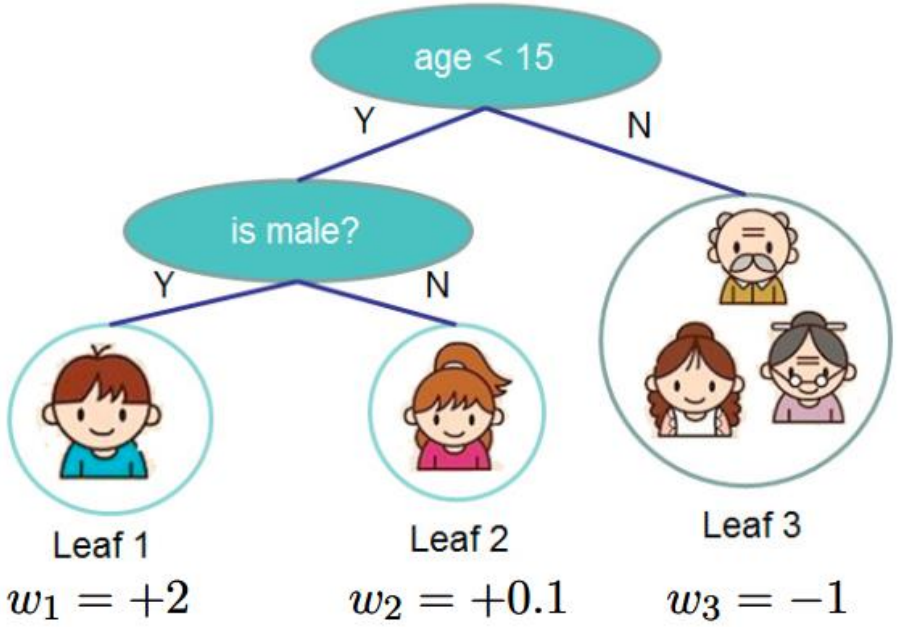


Penalty on Tree Complexity

- We could define the tree complexity as,

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves **L2 norm of leaf scores**

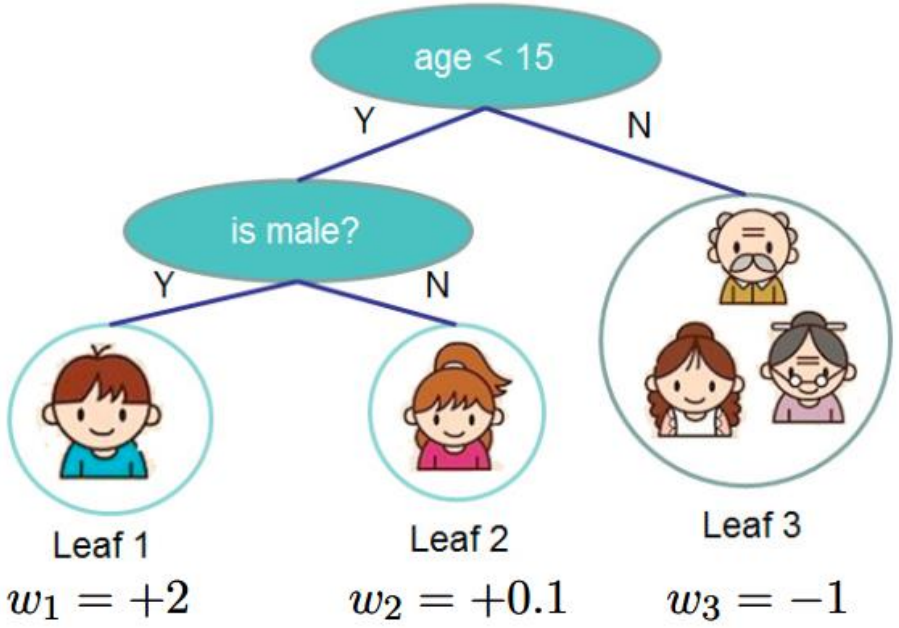


Penalty on Tree Complexity

- We could define the tree complexity as,

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Number of leaves **L2 norm of leaf scores**



$$\Omega(f_t) = 3\gamma + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

Rewritten Objective

- Rewrite objective function with penalty $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$

$$Obj^{(t)} \approx \sum_{i=1}^{n}\left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) = \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + C$$

Rewritten Objective

- Rewrite objective function with penalty $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$

What is C?

$$Obj^{(t)} \approx \sum_{i=1}^{n}\left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) = \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + C$$

Rewritten Objective

- Rewrite objective function with penalty $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$

What is C?

$$Obj^{(t)} \approx \sum_{i=1}^{n}\left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) = \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + C$$

- Regroup the objective by each leaf

$$Obj^{(t)} = \sum_{i=1}^{n}[\quad\quad] + (\quad\quad) = \sum_{j=1}^{T}[\quad\quad] + (\quad\quad)$$

Tree definition **Sum over leaves**

Rewritten Objective

- Rewrite objective function with penalty $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$

What is C?

$$Obj^{(t)} \approx \sum_{i=1}^{n}\left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) = \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + C$$

- Regroup the objective by each leaf

$$Obj^{(t)} = \sum_{i=1}^{n}\left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2\right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + C = \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right) w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right) w_j^2\right] + \gamma T + C$$

Tree definition **Sum over leaves**

- I_j is the instance set in leaf j: $\{i | q(x_i) = j\}$

Closed-form Solution

This is the sum of T independent quadratic functions.

- Objective function $$Obj^{(t)} = \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right)w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)w_j^2\right] + \gamma T$$

Closed-form Solution

This is the sum of T independent quadratic functions.

- Objective function $$Obj^{(t)} = \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right)w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)w_j^2\right] + \gamma T$$
- Define for simplicity $$G_j = \sum_{i\in I_j} g_i \qquad H_j = \sum_{i\in I_j} h_i$$

Closed-form Solution

This is the sum of T independent quadratic functions.

- Objective function $$Obj^{(t)} = \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right)w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)w_j^2\right] + \gamma T$$
- Define for simplicity $$G_j = \sum_{i\in I_j} g_i \qquad H_j = \sum_{i\in I_j} h_i$$
- Rewrite objective $$Obj^{(t)} = \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}\left(H_j + \lambda\right)w_j^2\right] + \gamma T$$

Closed-form Solution

This is the sum of T independent quadratic functions.

- Objective function $$Obj^{(t)} = \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right)w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)w_j^2\right] + \gamma T$$
- Define for simplicity $$G_j = \sum_{i\in I_j} g_i \qquad H_j = \sum_{i\in I_j} h_i$$
- Rewrite objective $$Obj^{(t)} = \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2\right] + \gamma T$$
- With the fixed tree structure $q \in \mathbb{R}^d \mapsto \{1,2,\dots,T\}$
- The closed-form solution
$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

Closed-form Solution

This is the sum of T independent quadratic functions.

- Objective function $$Obj^{(t)} = \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right)w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)w_j^2\right] + \gamma T$$
- Define for simplicity $$G_j = \sum_{i\in I_j} g_i \qquad H_j = \sum_{i\in I_j} h_i$$
- Rewrite objective $$Obj^{(t)} = \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2\right] + \gamma T$$
- With the fixed tree structure $q \in \mathbb{R}^d \mapsto \{1,2,\dots,T\}$
- The closed-form solution

$$w_j^* = -\frac{G_j}{H_j + \lambda} \qquad Obj^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

Closed-form Solution

This is the sum of T independent quadratic functions.

- Objective function $$Obj^{(t)} = \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right)w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)w_j^2\right] + \gamma T$$
- Define for simplicity $$G_j = \sum_{i\in I_j} g_i \qquad H_j = \sum_{i\in I_j} h_i$$
- Rewrite objective $$Obj^{(t)} = \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2\right] + \gamma T$$
- With the fixed tree structure $q \in \mathbb{R}^d \mapsto \{1,2,\dots,T\}$
- The closed-form solution

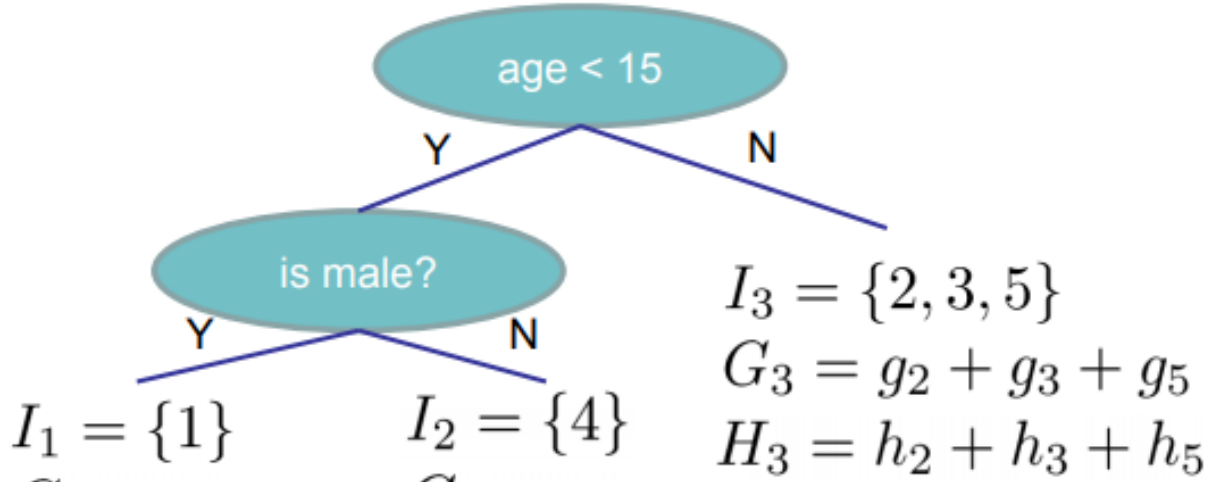
$$w_j^* = -\frac{G_j}{H_j + \lambda} \qquad Obj^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

This measures how good a tree structure is!

The Structure Score Calculation

- We define the $Obj^{(t)}$ as the structure score of the tree.

Instance index	gradient statistics
1	g1, h1
2	g2, h2
3	g3, h3
4	g4, h4
5	g5, h5



$$I_1 = \{1\}$$
$$G_1 = g_1$$
$$H_1 = h_1$$

$$I_2 = \{4\}$$
$$G_2 = g_4$$
$$H_4 = h_4$$

$$I_3 = \{2, 3, 5\}$$
$$G_3 = g_2 + g_3 + g_5$$
$$H_3 = h_2 + h_3 + h_5$$

$$Obj^{(t)} = -\frac{1}{2}\sum_{j=1}^{3}\frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

Searching Algorithm for Single Tree

- Enumerate all the possible tree structures q
- Calculate the structure score for the q using the equation

$$Obj^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Find the best tree structure with the smallest structure score.

- Use the optimal leaf weight

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

- But… there can be infinite possible tree structures.

Greedy Learning of the Tree

- In practice we grow the tree greedily
  - Start from tree with depth 0
  - For each leaf node of the tree, try to add a split.
  - The change of objective after adding the split is,

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

Left child score

Right child score

Score if we do not split

Penalty of the new leaf

Greedy Learning of the Tree

- In practice we grow the tree greedily
  - Start from tree with depth 0
  - For each leaf node of the tree, try to add a split.
  - The change of objective after adding the split is,

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

Left child score

Right child score

Score if we do not split

Penalty of the new leaf

- Introducing a split may not obtain positive gain, because of the last term.

Greedy Learning of the Tree

- In practice we grow the tree greedily
  - Start from tree with depth 0
  - For each leaf node of the tree, try to add a split.
  - The change of objective after adding the split is,

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

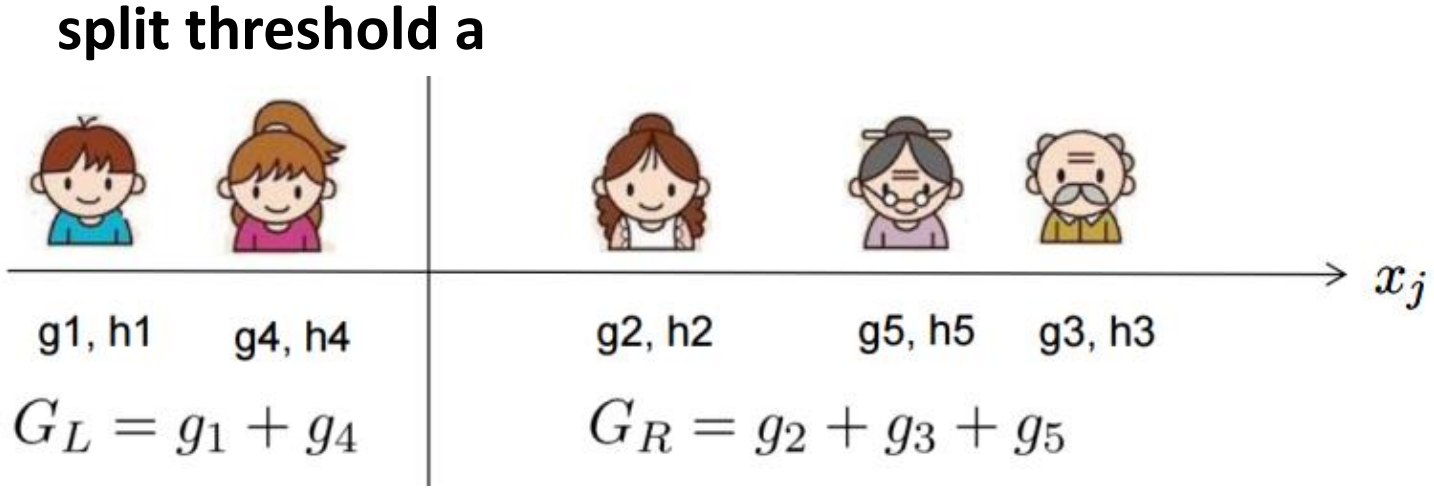
Left child score | Right child score | Score if we do not split | Penalty of the new leaf

- Introducing a split may not obtain positive gain, because of the last term.

We still need to enumerate all the possible splits.
How can we find the best split efficiently?

Efficiently Find the Optimal Split

- What is the gain of a split rule $x_j < a$? Say x_j is age.
- We sort the data ascendingly.



- All we need is the sum of g and h on each side, and calculate,

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

- Left to right linear scan over sorted instance is enough to decide the best split along the feature.

An Algorithm for Split Finding

- For each node, enumerate over all features.
  - For each feature, sorted the instances by feature value.
  - Use a linear scan to decide the best split along that feature.
  - Take the best split solution along all the features.

An Algorithm for Split Finding

- For each node, enumerate over all features.
  - For each feature, sorted the instances by feature value.
  - Use a linear scan to decide the best split along that feature.
  - Take the best split solution along all the features.

- Time complexity growing a tree of depth K.
  - It is $O(ndK \log n)$: or each level, need $O(n \log n)$ time to sort.
    - There are d features, and we need to do it for K level.
  - This can be further optimized (e.g., use approximation or caching the sorted features).
  - Can scale to very large dataset.

Pruning and Regularization

- Recall the gain of split, can it be negatiave?

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

Pruning and Regularization

- Recall the gain of split, it can be negative!
  - When the training loss reduction is smaller than regularization
  - Trade-off between simplicity and predictiveness

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

- Pre-stopping
  - Stop split if the best split have negative gain
  - But maybe a split can benefit future splits
- Post-Pruning
  - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain

Recap: Boosted Tree Algorithm

- Add a new tree in each iteration

Recap: Boosted Tree Algorithm

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \qquad h_i = \partial^2{}_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

Recap: Boosted Tree Algorithm

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \qquad h_i = \partial^2{}_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

- Use the statistics to greedily grow a tree $f_t(x)$

$$Obj^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

Recap: Boosted Tree Algorithm

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \qquad h_i = \partial^2{}_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

- Use the statistics to greedily grow a tree $f_t(x)$

$$Obj^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

Recap: Boosted Tree Algorithm

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \qquad h_i = \partial^2{}_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

- Use the statistics to greedily grow a tree $f_t(x)$

$$Obj^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

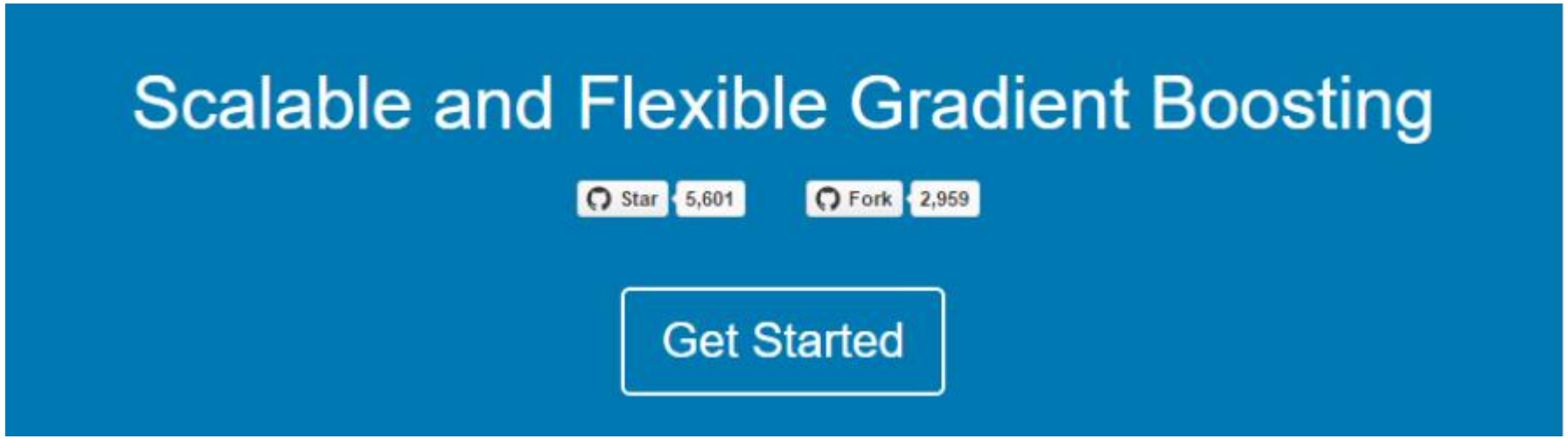
- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
  - ϵ is called step-size or shrinkage, usually set around 0.1
  - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting.

Reference

- Greedy function approximation a gradient boosting machine. J.H. Friedman
  - First paper about gradient boosting
- Stochastic Gradient Boosting. J.H. Friedman
  - Introducing bagging trick to gradient boosting
- Elements of Statistical Learning. T. Hastie, R. Tibshirani and J.H. Friedman
  - Contains a chapter about gradient boosted boosting
- Additive logistic regression a statistical view of boosting. J.H. Friedman T. Hastie R. Tibshirani
  - Uses second-order statistics for tree splitting, which is closer to the view presented in this slide
- Learning Nonlinear Functions Using Regularized Greedy Forest. R. Johnson and T. Zhang
  - Proposes to do fully corrective step, as well as regularizing the tree complexity. The regularizing trick is closed related to the view present in this slide

XGBoost

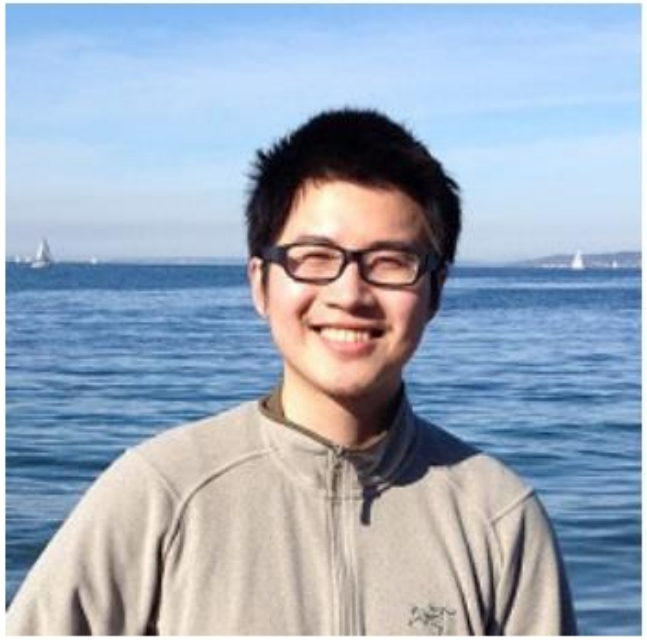
- The most effective and efficient toolkit for GBDT



https://xgboost.readthedocs.io/en/latest/

XGBoost

Tianqi Chen Publications Teaching Projects



Tianqi Chen

Ph.D. student 2013 - present

Paul G. Allen School of Computer Science & Engineering

University of Washington

Email: tqchen@cs.washington.edu

Google Scholar: Tianqi Chen

Github: @tqchen

Twitter: @tqchenml

https://homes.cs.washington.edu/~tqchen/

Summary

Summary

- Classification And Regression Tree.
- Binary tree.
- Gini index/residual sum of squares.
- Easy to interpret.

Classification Trees

Summary

- Boostrap Aggregating (Leo Breiman).
- **Idea**: train several different models separately, then majority voting.
- Usually treats each predictor trained on a bootstrap set with the same weight.

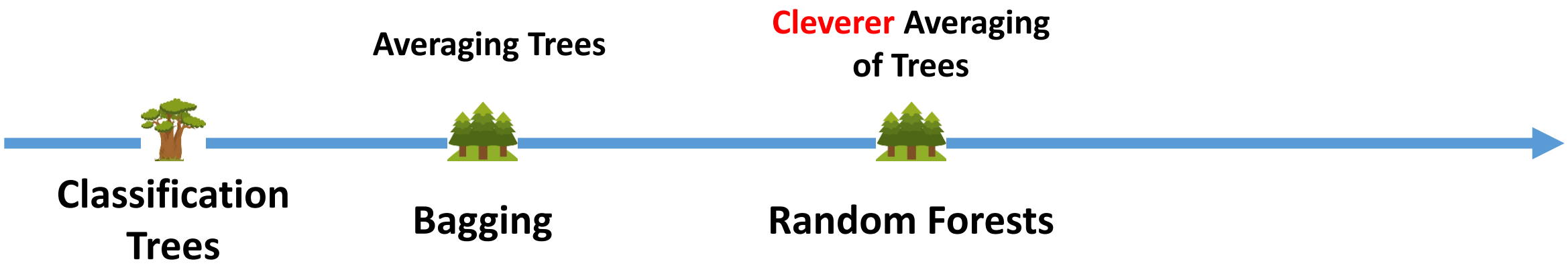
Averaging Trees

Classification Trees

Bagging

Summary

- Tries to de-correlate the bootstrap-trained predictors (decision trees) by sampling features.
- Two sources of randomness:
  - Random training samples;
  - Random attributes.



Summary

- Boosting strategically learns and combines the next predictor based on previous predictors.
- Learn in sequence.
- Using a weighted form of the data set.
- E.g., AdaBoost, GBDT, XgBoost.