Introduction to Machine Learning

Jiayu Zhou

Department of Computer Science and Engineering
Michigan State University
East Lansing, MI USA

Course Info

Course information

- Instructor: Dr. Jiayu Zhou `jiayuz@msu.edu`
  - Assistant Professor of Computer Science, Michigan State University
  - Research interests: large-scale machine learning, data mining, and biomedical informatics
  - Published more than 80 peer-reviewed journal and conference papers in data mining and machine learning. Best Student Paper Award in 2014 IEEE ICDM, the Best Student Paper Awards at the 2016 IEEE ISBI, 2016 IEEE BigData.
  - A recipient of the National Science Foundation CAREER Award (2018).

Course information (Cont'd)

- An introduction to the field of machine learning,
  - Introduction to machine learning (Perceptron and Deep Learning)
  - Tree methods and ensemble methods (Random Forest, Adaboost, Gradient Boosting)
- Prerequisites
  - Linear algebra, Basic probability and statistics, Basic numerical optimization, Python programming
- Learning outcomes include:
  1. Understanding the foundation, major techniques, applications, and challenges of machine learning;
  2. The ability to apply basic machine learning algorithms for solving real-world problems.

Intro to Machine Learning

Computation

$$213621.36521 \times 142513.54/1254.3215$$

Learning

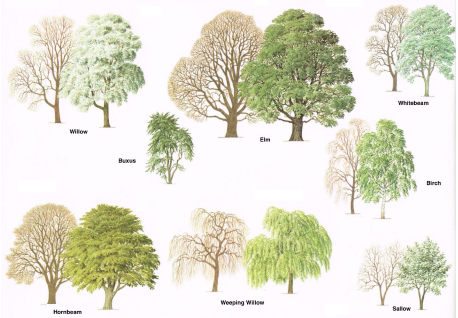
Learn the concept of a tree... How do you define a tree?

Learning

Learn the concept of a tree... How do you define a tree?

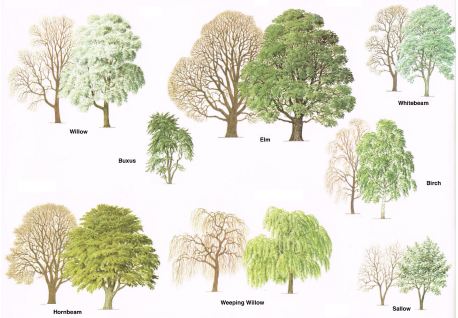
A brown *trunk* moving upward and *branching* with *leaves*

Defining is Hard; Recognizing is Easy



- Hard to give a complete mathematical definition of a tree
- Even a 3-year-old can tell a tree from a non-tree

Defining is Hard; Recognizing is Easy

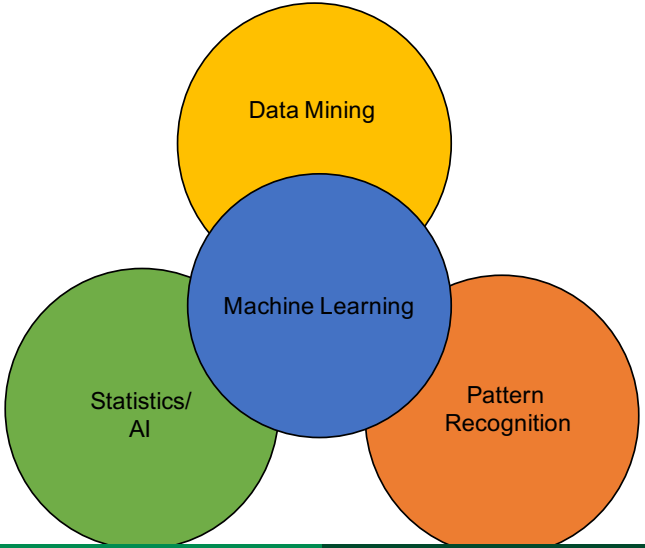


- Hard to give a complete mathematical definition of a tree
- Even a 3-year-old can tell a tree from a non-tree
- The 3-year-old has learned from data!

What is Machine Learning?

- The goal of the field of machine learning is to build computer systems that learn from experience (data) and that are capable to adapt to their environments.
- Learning techniques and methods developed in this field have been successfully applied to a variety of applications
  - text classification
  - collaborative filtering
  - digit/face recognition
  - software engineering
  - gene/protein function prediction
  - financial forecasting
  - credit card fraud detection
  - medical informatics

Related Fields



Machine Learning Tasks

- Supervised learning
  - Classification
  - Regression
- Unsupervised learning
  - Clustering
  - Visualization (dimensionality reduction)
- Semi-supervised learning
- Reinforcement learning

Case Study: Credit Approval

Let's use a conceptual example to crystallize the issues.

age	32 years
gender	male
salary	40,000
debt	26,000
years in job	1 year
years at home	3 years
...	...

Approve for credit?

Case Study: Credit Approval

Let's use a conceptual example to crystallize the issues.

age	32 years
gender	male
salary	40,000
debt	26,000
years in job	1 year
years at home	3 years
...	...

Approve for credit?

- Using salary, debt, years in residence, etc. approve for credit or not
- No magic credit approval formula
- Banks have lots of data
  - customer information: salary, debt, etc.
  - whether or not they defaulted on their credit.

Case Study: Credit Approval

Let's use a conceptual example to crystallize the issues.

age	32 years
gender	male
salary	40,000
debt	26,000
years in job	1 year
years at home	3 years
...	...

Approve for credit?

- Using salary, debt, years in residence, etc. approve for credit or not
- No magic credit approval formula
- Banks have lots of data
  - customer information: salary, debt, etc.
  - whether or not they defaulted on their credit.

A pattern exists. We don't know it. We have data to learn it.

The Key Players

- Salary, debt, years in residence
- Approve credit or not
- True relationship between $\mathbf{x}$ and y
- Data on customers

- input $\mathbf{x} \in \mathbb{R}^d \equiv \mathcal{X}$
- output $y \in \{-1, +1\} \equiv \mathcal{Y}$
- target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

$\mathcal{X}, \mathcal{Y}$ and $\mathcal{D}$ are *given* by the learning problem;
The target f is fixed but unknown.

The Key Players

- Salary, debt, years in residence
- Approve credit or not
- True relationship between $\mathbf{x}$ and y
- Data on customers

- input $\mathbf{x} \in \mathbb{R}^d \equiv \mathcal{X}$
- output $y \in \{-1, +1\} \equiv \mathcal{Y}$
- target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

$\mathcal{X}, \mathcal{Y}$ and $\mathcal{D}$ are *given* by the learning problem;
The target f is fixed but unknown.

Goal: learn the function f from the data $\mathcal{D}$.

Learning

- Start with a set of candidate hypothese $\mathcal{H}$ which you think are likely to represent f.

$$\mathcal{H} = \{h_1, h_2, \ldots, \}$$

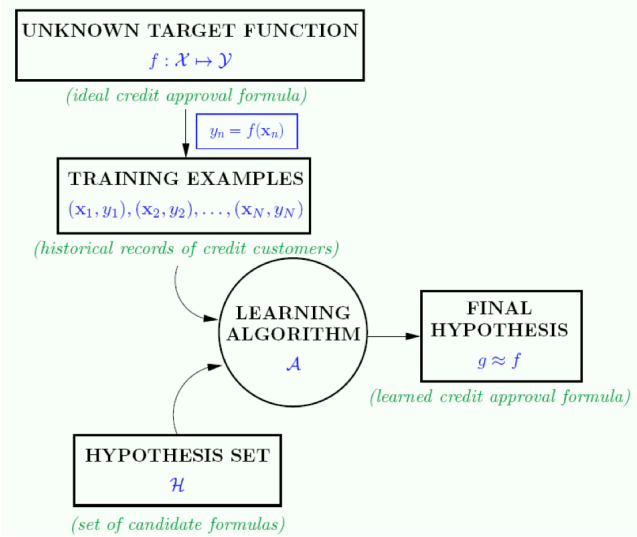
  is called the hypothesis set or *model*.
- Select a hypothesis g from $\mathcal{H}$. The way we do this is called a *learning algorithm*.
- Use g for new customers. We hope $g \approx f$.

$\mathcal{X}, \mathcal{Y}$ and $\mathcal{D}$ are *given* by the learning problem;
The target f is **fixed but unknown**

We choose $\mathcal{H}$ and the learning algorithm

This is a very general setup (e.g., choose $\mathcal{H}$ to be all possible hypotheses)

Summary of the Learning Setup



Evaluation

- Evaluate machine learning models on **unseen** data points that are *not* used in training.
- Metrics
  - Classification: Accuracy, Precision, Recall, Area under the ROC curve (AUC). (Issue: class imbalance?)
  - Regression: Mean squared error (MSE), Correlation R^2. (Issue: inconsistency between metrics)

Perceptron and Deep Learning

The Key Players

Credit Approval Problem

- **Input:** Salary, debt, years in residence; $\mathbf{x} \in \mathbb{R}^d \equiv \mathcal{X}$
- **Output:** Approve credit or not; $y \in \{-1, +1\} \equiv \mathcal{Y}$
- **Target:** True relationship between $\mathbf{x}$ and y; $f : \mathcal{X} \to \mathcal{Y}$
- Customer data; $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

$\mathcal{X}, \mathcal{Y}$ and $\mathcal{D}$ are *given* by the learning problem;
The target f is fixed but unknown

The Key Players

Credit Approval Problem

- **Input:** Salary, debt, years in residence; $\mathbf{x} \in \mathbb{R}^d \equiv \mathcal{X}$
- **Output:** Approve credit or not; $y \in \{-1, +1\} \equiv \mathcal{Y}$
- **Target:** True relationship between $\mathbf{x}$ and y; $f : \mathcal{X} \to \mathcal{Y}$
- Customer data; $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

$\mathcal{X}, \mathcal{Y}$ and $\mathcal{D}$ are *given* by the learning problem;
The target f is fixed but unknown

Goal: learn the function f from the data $\mathcal{D}$.

A Simple Learning Model

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^T$, Give importance weights to the different features and compute a "Credit Score"

$$\text{“Credit Score”} = \sum_{i=1}^{d} w_i x_i.$$

A Simple Learning Model

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^T$, Give importance weights to the different features and compute a "Credit Score"

$$\text{"Credit Score"} = \sum_{i=1}^{d} w_i x_i.$$

- Approve credit if the "Credit Score" is acceptable.

$$\text{Approve credit if } \sum_{i=1}^{d} w_i x_i > \text{threshold}, \text{("Credit score" is good)}$$

$$\text{Deny credit if } \sum_{i=1}^{d} w_i x_i < \text{threshold}, \text{("Credit score" is bad)}$$

A Simple Learning Model

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^T$, Give importance weights to the different features and compute a "Credit Score"

$$\text{“Credit Score”} = \sum_{i=1}^{d} w_i x_i.$$

- Approve credit if the "Credit Score" is acceptable.

$$\text{Approve credit if } \sum_{i=1}^{d} w_i x_i > \text{threshold, (“Credit score” is good)}$$

$$\text{Deny credit if } \sum_{i=1}^{d} w_i x_i < \text{threshold, (“Credit score” is bad)}$$

- How to choose the importance weights w_i?

$$x_i \text{ is important} \Rightarrow \text{large weight}$$

$$x_i \text{ is beneficial for credit} \Rightarrow \text{positive weight}$$

$$x_i \text{ is detrimental for credit} \Rightarrow \text{negative weight}$$

A Simple Learning Model

$$\text{Approve credit if } \sum_{i=1}^{d} w_i x_i > \text{threshold, (“Credit score” is good)}$$

$$\text{Deny credit if } \sum_{i=1}^{d} w_i x_i < \text{threshold, (“Credit score” is bad)}$$

can be written formally as

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right)$$

The “bias weight” w_0 correspond to the threshold. (How?)

The Perceptron Hypothesis Set

We have defined a Hypothesis set $\mathcal{H}$

$$\mathcal{H} = \left\{ h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{d} w_i x_i + w_0 \right) = \text{sign}(\mathbf{w}^T \mathbf{x}) \right\}$$

which is uncountably infinite.

The Perceptron Hypothesis Set

We have defined a Hypothesis set $\mathcal{H}$

$$\mathcal{H} = \left\{ h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{d} w_i x_i + w_0 \right) = \text{sign}(\mathbf{w}^T \mathbf{x}) \right\}$$

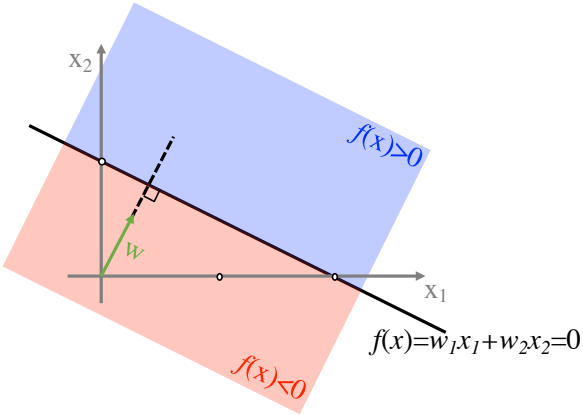
which is uncountably infinite. We have used the "dummy variable":

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_d \end{bmatrix} \in \mathbb{R}^{d+1}, \quad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_d \end{bmatrix} \in \{1\} \times \mathbb{R}^d$$

This hypothesis set is called the *perceptron* or *linear separator*

Geometry of The Perceptron

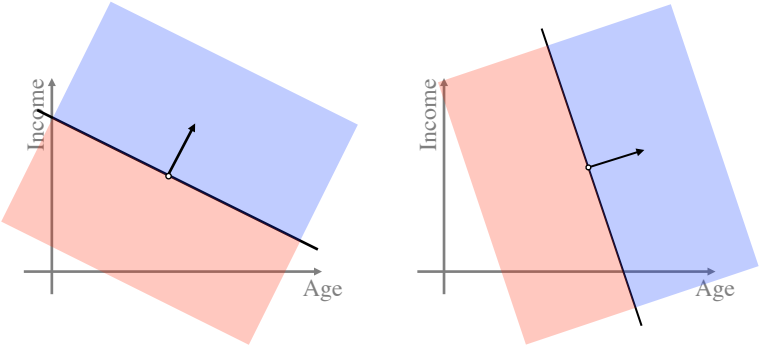
- In 2-d space, $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ defines a line that separates the space.



- In higher dimensional space, this corresponds to a hyperplane.

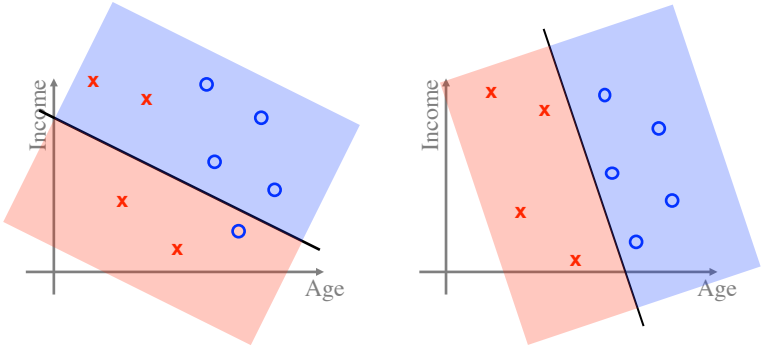
Geometry of The Perceptron

So many choices...



Which one should we pick?

Use the Data to Pick a Line



- A perceptron fits the data by using a line to separate the +1 from -1 data.
- **Fitting the data**: How to find a hyperplane that separates the data?

How to Learn a Final Hypothesis g from $\mathcal{H}$

- We want to select $g \in \mathcal{H}$ so that $g \approx f$.
- We certainly want $g \approx f$ on the data set $\mathcal{D}$. Ideally,

$$g(\mathbf{x}_n) = y_n$$

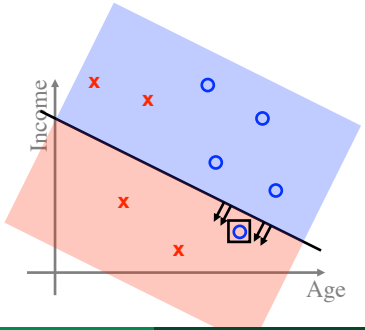
- How do we find such a g in the *infinite* hypothesis set $\mathcal{H}$, if it exists?

How to Learn a Final Hypothesis g from $\mathcal{H}$

- We want to select $g \in \mathcal{H}$ so that $g \approx f$.
- We certainly want $g \approx f$ on the data set $\mathcal{D}$. Ideally,

$$g(\mathbf{x}_n) = y_n$$

- How do we find such a g in the *infinite* hypothesis set $\mathcal{H}$, if it exists?
- **Idea!** Start with some weight vector and try to improve it.



The Perceptron Learning Algorithm (PLA)

A simple iterative method:

1. $\mathbf{w}(1) = 0$
2. for iteration $t = 1, 2, 3, \ldots$
3. the weight vector is $\mathbf{w}(t)$
4. From $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ pick any misclassified sample.
5. Call the misclassified example $(\mathbf{x}_*, y_*)$, i.e.,

$$\text{sign}(\mathbf{w}(t)^T \mathbf{x}_*) \neq y_*$$

6. Update the weight:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + y_* \mathbf{x}_*$$

7. $t \leftarrow t + 1$

PLA implements our idea: start at some weights and try to improve: **"incremental learning" on a single sample at a time.**

The Perceptron Learning Algorithm (PLA)

- Invented by Frank Rosenblatt in 1957, funded by ONR.
- The perceptron was intended to be a *machine*, rather than a program.

"The embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

— New York Times, 1958

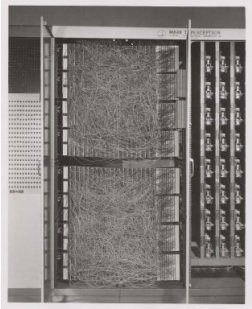
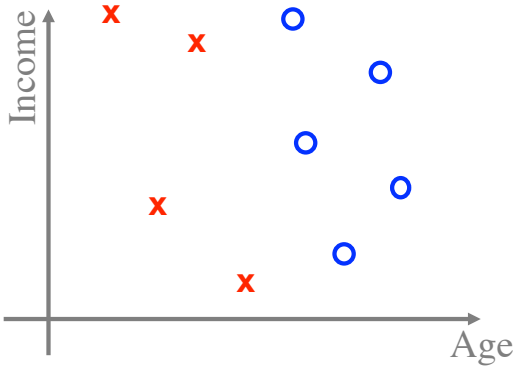


Figure: Mark I

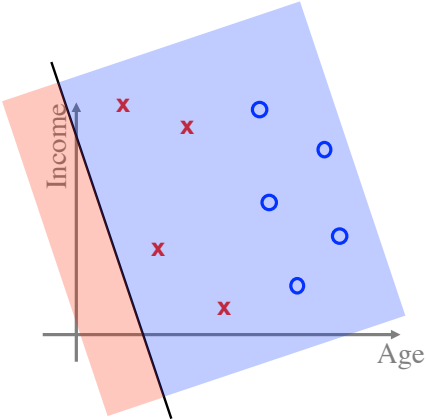
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



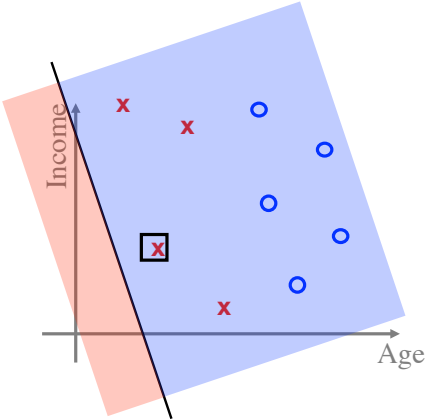
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



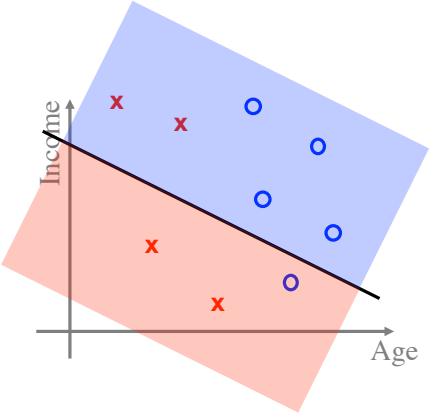
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



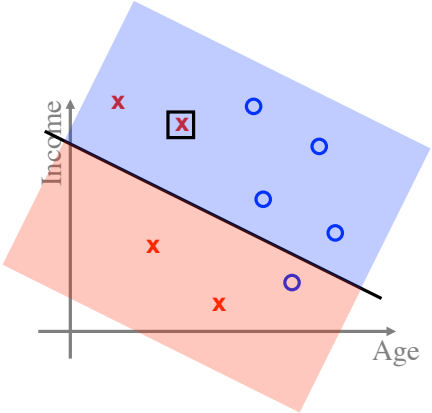
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



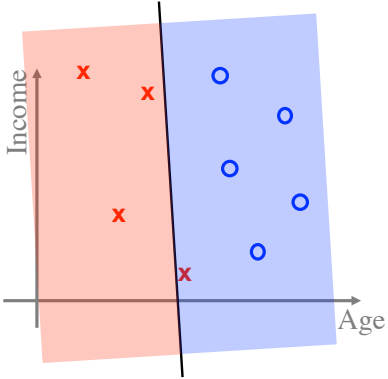
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



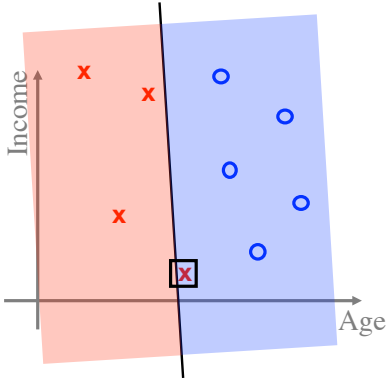
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



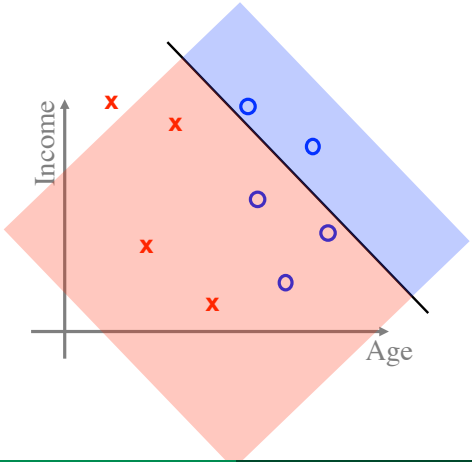
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



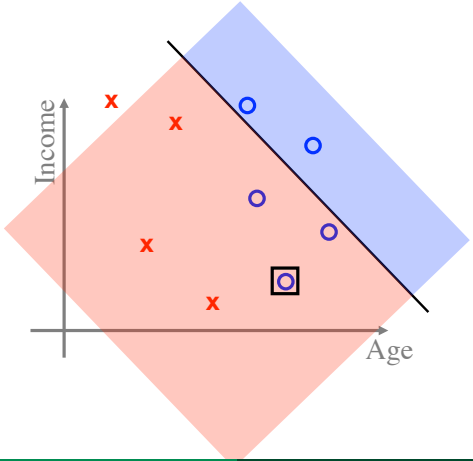
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



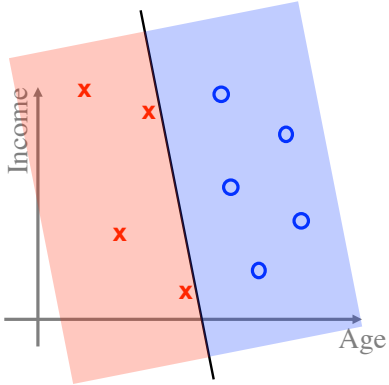
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



Convergence of Perceptron

- We have the following convergence of PLA:

$$t \leq \frac{\|\mathbf{w}^*\|_2^2 \cdot \max \|\mathbf{x}(.)\|_2^2}{\delta^2}$$

- What does this bound tell us about the Perceptron?
  - δ measures how close the solution decision boundary is to the input patterns.
  - If the input classes are difficult to separate (are close to the decision boundary) it will take many iterations for the algorithm to converge.
  - Additional assumption required: $\max \|\mathbf{x}(.)\|_2^2$ is bounded.

Convergence of Perceptron

- We have the following convergence of PLA:

$$t \leq \frac{\|\mathbf{w}^*\|_2^2 \cdot \max \|\mathbf{x}(.)\|_2^2}{\delta^2}$$

- What does this bound tell us about the Perceptron?
  - δ measures how close the solution decision boundary is to the input patterns.
  - If the input classes are difficult to separate (are close to the decision boundary) it will take many iterations for the algorithm to converge.
  - Additional assumption required: $\max \|\mathbf{x}(.)\|_2^2$ is bounded.
- PLA algorithm is a hard version of stochastic logistic regression.
  - Logistic Regression (stochastic gradient descent)

$$\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) + y_* \mathbf{x}_* \left(\frac{\eta}{1 + \exp\left(y_* \mathbf{w}^T \mathbf{x}_*\right)} \right)$$

Recall PLA:

$$\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) + y_* \mathbf{x}_*$$

Deep Learning

- Deep learning: multi-layer perceptron

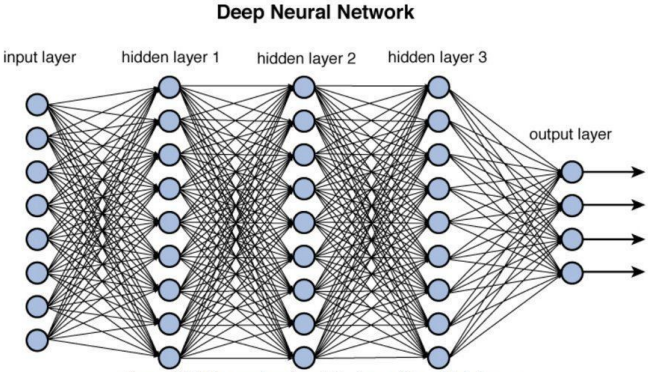


Figure 12.2 Deep network architecture with multiple layers.

- Issues: Local optimals, efficiency, generalization

Tree Methods

Running Example: Wait for a table?

- Problem: decide whether to wait for a table at a restaurant,

Running Example: Wait for a table?

- Problem: decide whether to wait for a table at a restaurant, based on the following features:
  1. *Alternate*: is there an alternative restaurant nearby?
  2. *Bar*: is there a comfortable bar area to wait in?
  3. *Fri/Sat*: is today Friday or Saturday?
  4. *Hungry*: are we hungry?
  5. *Patrons*: number of people in the restaurant (None, Some, Full)
  6. *Price*: price range ($, $$, $$$)
  7. *Raining*: is it raining outside?
  8. *Reservation*: have we made a reservation?
  9. *Type*: kind of restaurant (French, Italian, Thai, Burger)
  10. *WaitEstimate*: estimated waiting time (0-10, 10-30, 30-60, >60)

Running Example: Wait for a table? (cont.)

- Examples described by attribute values (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

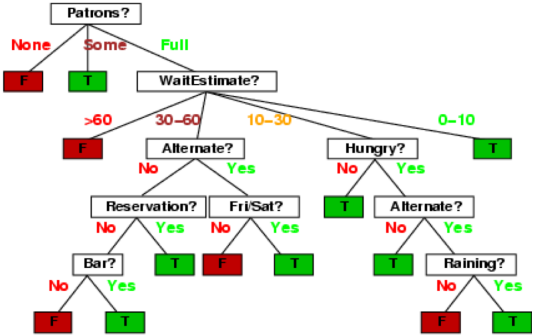
- Classification of examples is positive (T) or negative (F)
- General form for data: a number N of instances, each with attributes

Decision Trees

- One possible representation for hypotheses
- We imagine someone taking a sequence of decisions.

Decision Trees

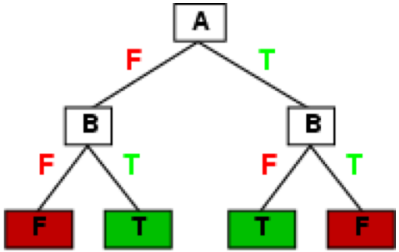
- One possible representation for hypotheses
- We imagine someone taking a sequence of decisions.
- E.g., here is the "true" tree for deciding whether to wait:



Expressiveness

- Decision trees can express any function of the input features.
- E.g., for Boolean functions, truth table row → path to leaf:

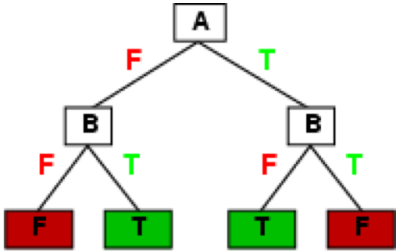
A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Expressiveness

- Decision trees can express any function of the input features.
- E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F

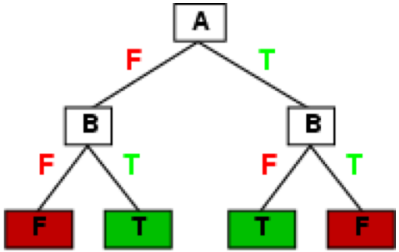


- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example but it probably won't generalize to new examples.

Expressiveness

- Decision trees can express any function of the input features.
- E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example but it probably won't generalize to new examples.
- Prefer to find more compact decision trees: we don't want to memorize the data, we want to find the structure in the data!

Hypothesis space

- How many distinct decision trees with n Boolean attributes?

Hypothesis space

- How many distinct decision trees with n Boolean attributes?
  = number of Boolean functions

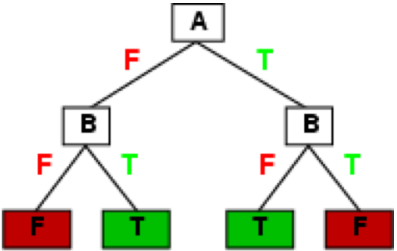
Hypothesis space

- How many distinct decision trees with n Boolean attributes?
  = number of Boolean functions
  = number of distinct truth tables with 2^n rows = 2^{2^n}

Hypothesis space

- How many distinct decision trees with n Boolean attributes?
  = number of Boolean functions
  = number of distinct truth tables with 2^n rows = 2^{2^n}
- With 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees
- Example:

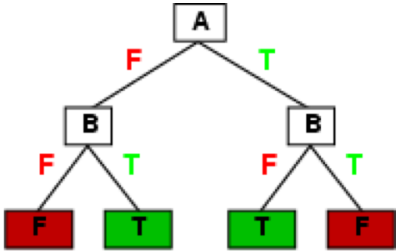
A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Hypothesis space

- How many distinct decision trees with n Boolean attributes?
  = number of Boolean functions
  = number of distinct truth tables with 2^n rows = 2^{2^n}
- With 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees
- Example:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



$n = 2$: $2^2 = 4$ rows. For each row we can choose T or F: 2^4 functions.

Decision tree learning

- If there are so many possible trees, can we actually search this space?

Decision tree learning

- If there are so many possible trees, can we actually search this space? (solution: greedy search).

Decision tree learning

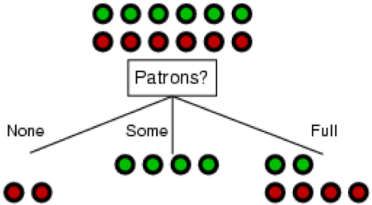
- If there are so many possible trees, can we actually search this space? (solution: greedy search).
- *Aim*: find a small tree consistent with the training examples

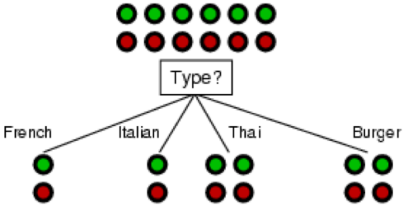
Decision tree learning

- If there are so many possible trees, can we actually search this space? (solution: greedy search).
- *Aim*: find a small tree consistent with the training examples
- *Idea*: (recursively) choose **most significant** feature as root of (sub)tree.

Choosing an attribute

- What is a good attribute?

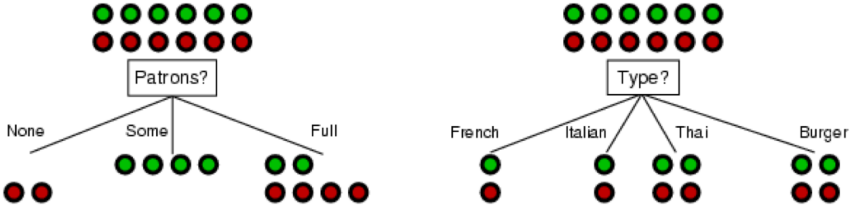




Patrons or Type?

Choosing an attribute

- What is a good attribute?



Patrons or Type?

- *Idea*: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative".

Using information theory

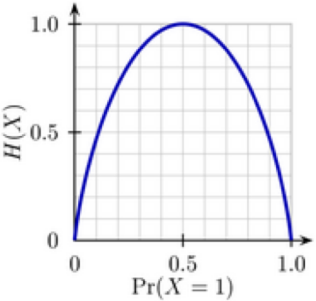
Entropy measures the amount of uncertainty in a probability distribution:

- Consider tossing a biased coin. If you toss the coin VERY often, the frequency of heads is, say, p, and hence the frequency of tails is $1 - p$. (fair coin $p = 0.5$).

Using information theory

Entropy measures the amount of uncertainty in a probability distribution:

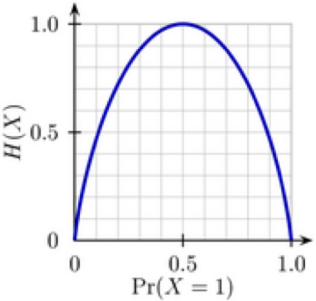
- Consider tossing a biased coin. If you toss the coin VERY often, the frequency of heads is, say, p, and hence the frequency of tails is $1 - p$. (fair coin $p = 0.5$).
- The uncertainty in any actual outcome is given by the entropy



Using information theory

Entropy measures the amount of uncertainty in a probability distribution:

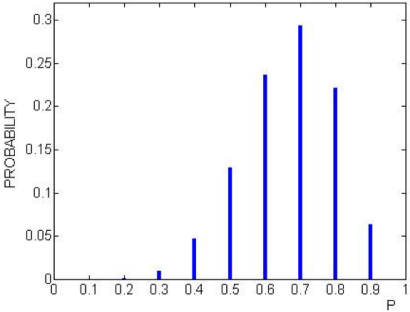
- Consider tossing a biased coin. If you toss the coin VERY often, the frequency of heads is, say, p, and hence the frequency of tails is $1 - p$. (fair coin $p = 0.5$).
- The uncertainty in any actual outcome is given by the entropy



Note, the uncertainty is zero if p=0 or 1 and maximal if we have $p = 0.5$.

Using information theory

If there are more than two states $s = 1, 2, \ldots, n$ we have (e.g. a die):



$$
\begin{aligned}
\textit{Entropy}(\mathrm{p}) = & -p(s=1)\log(p(s=1)) \\
& -p(s=2)\log(p(s=2)) \\
& \ldots \\
& -p(s=n)\log(p(s=n))
\end{aligned}
$$

where

$$\sum_{s=1}^{n} p(s) = 1$$

Using information theory

- Imagine we have p examples which are true (positive) and n examples which are false (negative).

Using information theory

- Imagine we have p examples which are true (positive) and n examples which are false (negative).
- Our best estimate of true or false (prior) is given by:

$$p(\text{true}) \approx p/(p+n), p(\text{false}) \approx n/(p+n).$$

Using information theory

- Imagine we have p examples which are true (positive) and n examples which are false (negative).
- Our best estimate of true or false (prior) is given by:

$$p(\text{true}) \approx p/(p+n), p(\text{false}) \approx n/(p+n).$$

- Hence the entropy is given by

$$\textit{Entropy}(\frac{p}{p+n}, \frac{n}{p+n}) \approx -\frac{p}{p+n}\log(\frac{p}{p+n}) - \frac{n}{p+n}\log(\frac{n}{p+n})$$

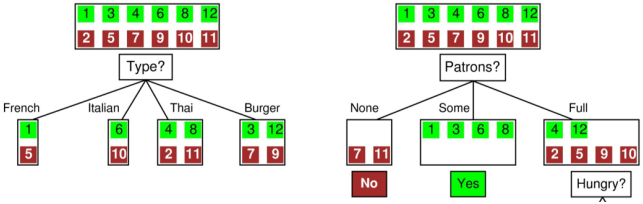
Using Information Theory

- *Question:* How much information do we gain if we disclose the value of some attribute?

Using Information Theory

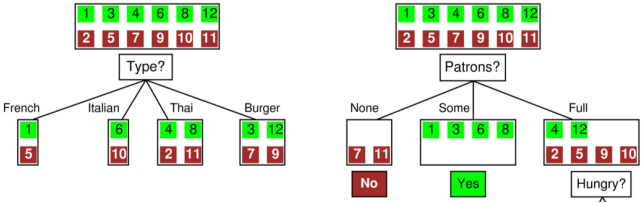
- *Question:* How much information do we gain if we disclose the value of some attribute?
- *Answer: uncertainty before - uncertainty after*

Example



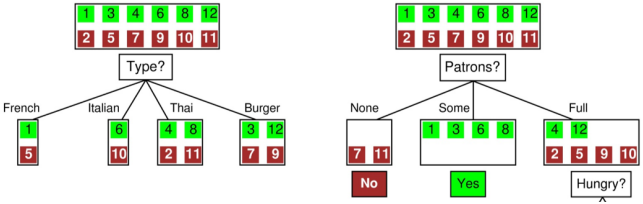
- **Before split**:

Example



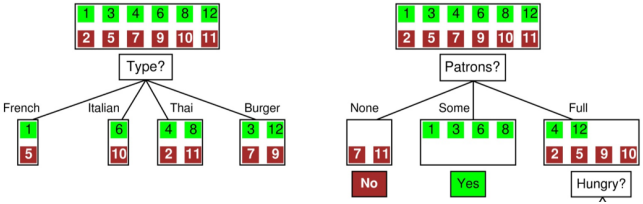
- **Before split**: $E = -1/2\log(1/2) - 1/2\log(1/2) = \log(2) = 1$
  There is "1 bit of information to be discovered".
- *Split* $\textit{TYPE}$:

Example



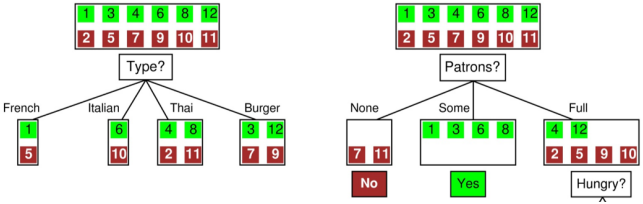
- **Before split**: $E = -1/2\log(1/2) - 1/2\log(1/2) = \log(2) = 1$
  There is "1 bit of information to be discovered".
- *Split* TYPE: If we go into branch FRENCH we have 1 bit, similarly for the others. On average: 1 bit. We gained nothing.

Example



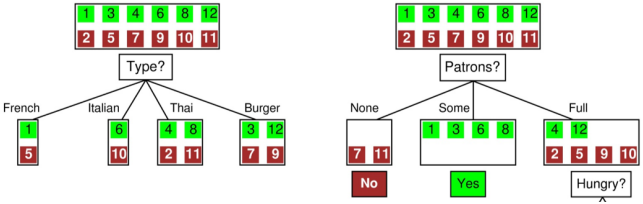
- **Before split**: $E = -1/2\log(1/2) - 1/2\log(1/2) = \log(2) = 1$
  There is "1 bit of information to be discovered".
- *Split* TYPE: If we go into branch FRENCH we have 1 bit, similarly for the others. On average: 1 bit. We gained nothing.
- *Split* PATRONS:

Example



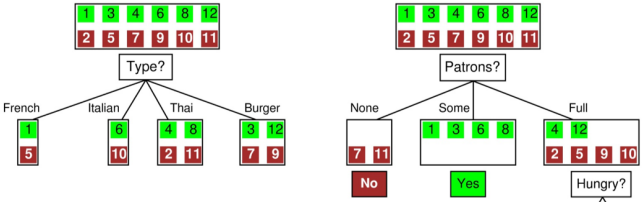
- **Before split**: $E = -1/2\log(1/2) - 1/2\log(1/2) = \log(2) = 1$
  There is "1 bit of information to be discovered".
- *Split* TYPE: If we go into branch FRENCH we have 1 bit, similarly for the others. On average: 1 bit. We gained nothing.
- *Split* PATRONS: Branch NONE and SOME, entropy =0; Branch FULL,

Example



- **Before split**: $E = -1/2\log(1/2) - 1/2\log(1/2) = \log(2) = 1$
  There is "1 bit of information to be discovered".
- *Split* TYPE: If we go into branch FRENCH we have 1 bit, similarly for the others. On average: 1 bit. We gained nothing.
- *Split* PATRONS: Branch NONE and SOME, entropy =0; Branch FULL, entropy = $-1/3\log(1/3) - 2/3\log(2/3)$.

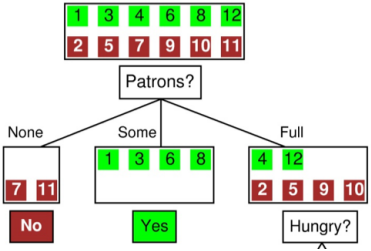
Example



- **Before split**: $E = -1/2\log(1/2) - 1/2\log(1/2) = \log(2) = 1$
  There is "1 bit of information to be discovered".
- *Split* Type: If we go into branch French we have 1 bit, similarly for the others. On average: 1 bit. We gained nothing.
- *Split* Patrons: Branch None and Some, entropy =0; Branch Full, entropy = $-1/3\log(1/3) - 2/3\log(2/3)$.
  So Patrons *gains more information!*

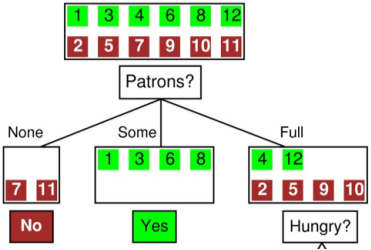
Information Gain

- How do we combine branches?



Information Gain

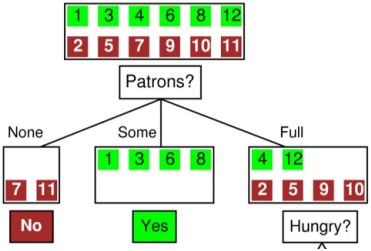
- How do we combine branches?



- An intuitive *weighting* scheme: 1/6 of the time we enter NONE, so we weight NONE with 1/6. Similarly: SOME has weight: 1/3 and FULL has weight 1/2.

Information Gain

- How do we combine branches?



- An intuitive *weighting* scheme: 1/6 of the time we enter NONE, so we weight NONE with 1/6. Similarly: SOME has weight: 1/3 and FULL has weight 1/2.
-

$$\text{Entropy}(A) = \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} \text{Entropy}\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

Information gain

- Information Gain (IG) or reduction in entropy from the attribute test:

$$\text{IG}(A) = \text{Entropy Before} - \text{Entropy After}$$

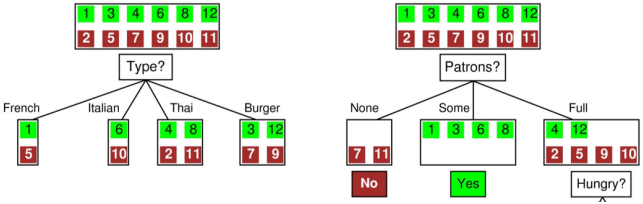
Information gain

- Information Gain (IG) or reduction in entropy from the attribute test:

$$\text{IG}(A) = \text{Entropy Before} - \text{Entropy After}$$

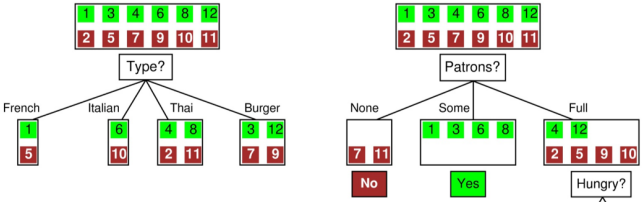
- Choose the attribute with the largest IG as the next split.

Information Gain



- For the training set, $p = n = 6$, $E(6/12, 6/12) = 1$bit.
- Compute $IG(Patrons)$ and $IG(Type)$.

Information Gain



- For the training set, $p = n = 6$, $E(6/12, 6/12) = 1$bit.
- Compute $IG(Patrons)$ and $IG(Type)$.

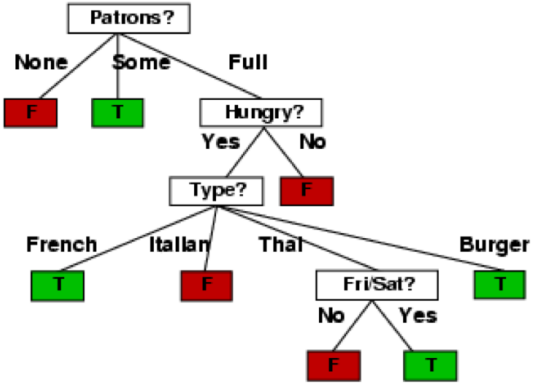
$$IG(P) = 1 - \left[\frac{2}{12}E(0,1) + \frac{4}{12}E(1,0) + \frac{6}{12}E(\frac{2}{6}, \frac{4}{6})\right] = 0.54$$

$$IG(T) = 1 - \left[\frac{2}{12}E(\frac{1}{2}, \frac{1}{2}) + \frac{2}{12}E(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12}E(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12}E(\frac{1}{2}, \frac{1}{2})\right] = 0bits$$

- Patrons has the highest IG of all attributes and so is chosen.

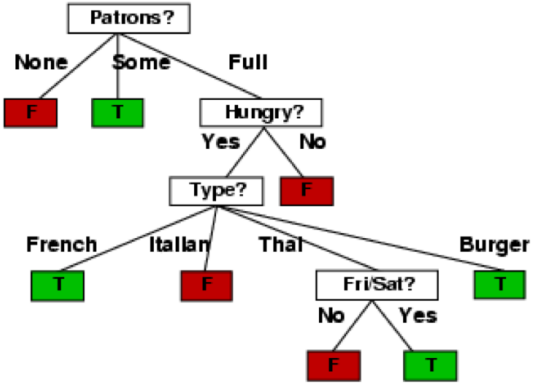
Example cont'd.

- Decision tree learned from the 12 examples:



Example cont'd.

- Decision tree learned from the 12 examples:



- Substantially simpler than "true" tree—a more complex hypothesis isn't justified by small amount of data

Technical Issues

- Continues variables (quantization)
- Data sparsity at certain splits
- Stoping criteria (e.g., minimal support available to a node)
- Pruning (tree size and generalization)

Regression Trees

- Decision trees where the target has continuous values.
- Splitting criteria: Residual sum of squares

$$\text{RSS} = \sum_{\text{left}} (y_i - y_L^*)^2 + \sum_{\text{right}} (y_i - y_R^*)^2$$

where:

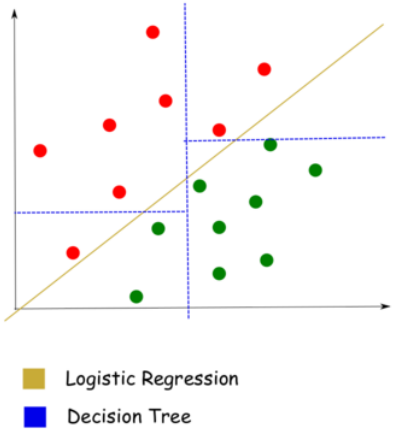
- y_L^* = mean y-value for left node
- y_R^* = mean y-value for right node

Classification and Regression Trees

Advantages

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

Comparison Between Logistic Regression and Decision Tree Decision Boundaries



https://www.quora.com/In-what-circumstances-or-properties-of-the-data-will-decision-trees-perform-worse-than-logistic-regression

Classification and Regression Trees

Disadvantages

- **Accuracy** - current methods, such as support vector machines and ensemble classifiers often have 30% lower error rates than classification and regression trees (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.

Classification and Regression Trees

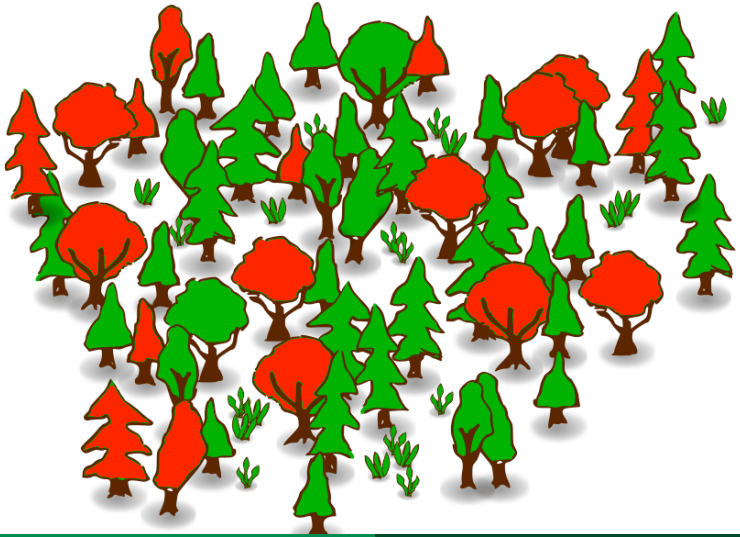
Disadvantages

- **Accuracy** - current methods, such as support vector machines and ensemble classifiers often have 30% lower error rates than classification and regression trees (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.

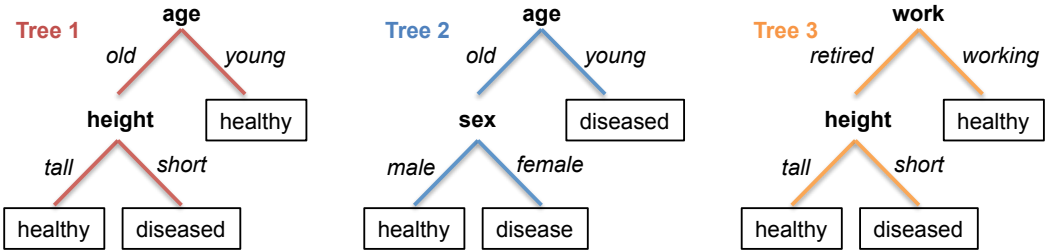
We can do better!
Random Forests

Random Forest

Random Forest

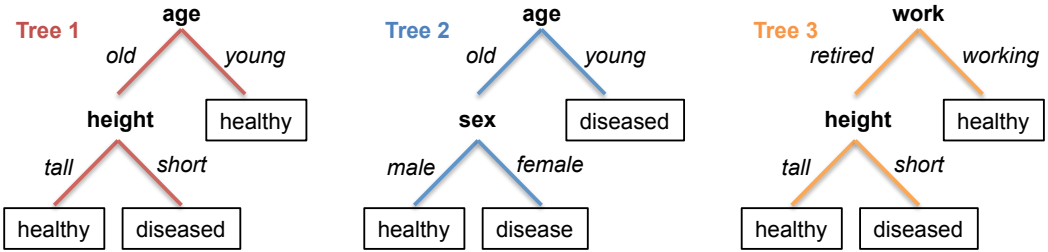


Example of Random Forest



- New sample: {old, retired, male, short}. Prediction?

Example of Random Forest



- New sample: {old, retired, male, short}. Prediction?
- Tree predictions: diseased, healthy, diseased
- Majority Rule: Diseased

The Random Forest Algorithm

1. For $b = 1$ to B
   1. Draw a bootstrap sample of size N from the training data (randomly sample with replacement)
   2. Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps, until the minimum node size $n_{\min}$ is reached.
      1. Select m variables at random from the p variables.
      2. Pick the best variable/split-point among the m
      3. Split the node into two daughter nodes
2. Output the ensemble of trees $\{T_b\}_1^B$

The Random Forest Algorithm

1. For $b = 1$ to B
   1. Draw a bootstrap sample of size N from the training data (randomly sample with replacement)
   2. Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps, until the minimum node size $n_{\min}$ is reached.
      1. Select m variables at random from the p variables.
      2. Pick the best variable/split-point among the m
      3. Split the node into two daughter nodes
2. Output the ensemble of trees $\{T_b\}_1^B$

To make a prediction at a new point x:

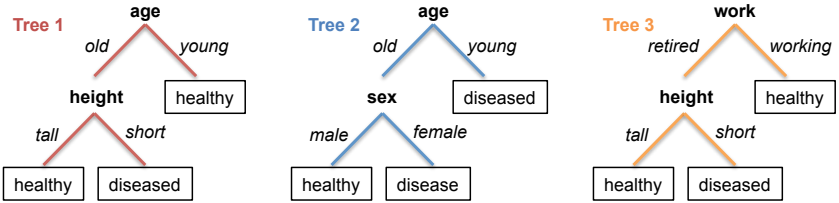
- *Regression:* $\hat{f}_{\text{rf}}^B = \frac{1}{B}\sum_{b=1}^B T_b(x)$
- *Classification:* Let $\hat{C}_b(x)$ be the class prediction of the b-th random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \text{majority vote}\{\hat{C}_b(x)\}_1^B$.

Differences to standard tree

- Train each tree on bootstrap resample of data
  Bootstrap resample of data set with N samples: Make new data set by drawing with replacement N samples; i.e., some samples will probably occur multiple times in new data set.
- For each split, consider only m randomly selected variables. (Feature bagging)
- Don't prune
- Fit a forest of B trees in such a way and use average or majority voting to aggregate results.

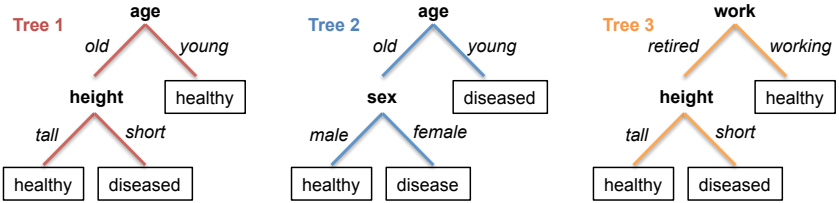
Variable Importance

- Advantage of tree methods is the intepretability.
- But we lose this property in random forest: what are the important variables in the forest?



Variable Importance

- Advantage of tree methods is the intepretability.
- But we lose this property in random forest: what are the important variables in the forest?



Good news! There are ways to identify important variables in the model, e.g., Out-of-Bag error.

Tree and Random Forest

Trees

- \+ Trees yield insight into decision rules
- \+ Rather fast
- \+ Easy to tune parameters
- – Prediction of trees tend to have a high variance

Random Forest

- \+ RF as smaller prediction variance and therefore usually a better general performance
- \+ Easy to tune parameters
- – Rather slow
- – "Black Box": Rather hard to get insights into decision rules

Adaboost

Motivation

- Easy to come up with rough rules of thumb for classifying data
  - email contains more than 50% capital letters.
  - email contains expression “buy now”.
- Each alone isn’t great, but better than random.
- Boosting converts rough rules of thumb into an accurate classifier.
  - Adaboost
  - Gradient Boosting

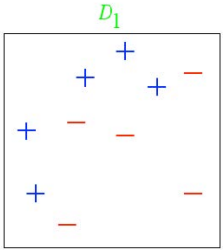
Adaboost

Given examples $(x_1, y_1) \ldots (x_n, y_n)$ with $y_i = \pm 1$.

- Let $D_1(i) = 1/n$ for $i = 1 \ldots n$.
- For $t = 1 \ldots T$ do
  - Run weak learner using examples with weights D_t.
  - Get weak (base) classifier h_t
  - Compute error: $\varepsilon_t = \sum_i D_t(i) \mathbb{I}(h_t(\mathbf{x}_i) \neq y_i)$
  - Compute magic coefficient $\alpha_t = \frac{1}{2} \log \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$
  - Update weights $D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{Z_t}$
- Output the final classifier $f_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$, and class label is given by

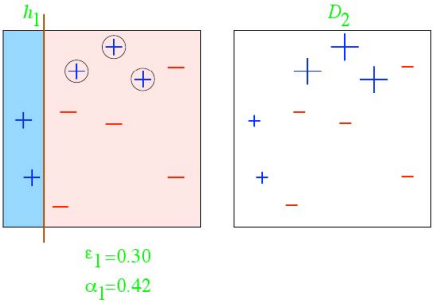
  $H(\mathbf{x}) = \text{sign}(f_T(\mathbf{x}))$

Toy example



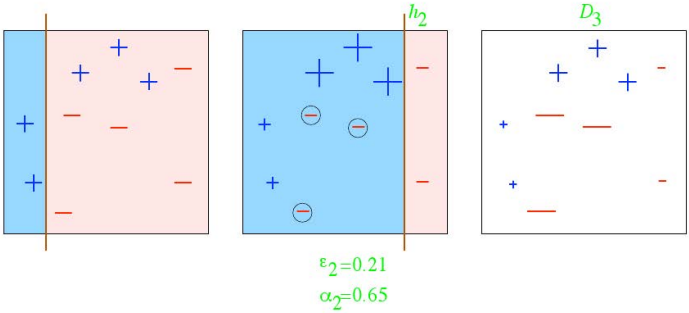
Weak classifiers: vertical or horizontal half-planes.

Adaboost round 1



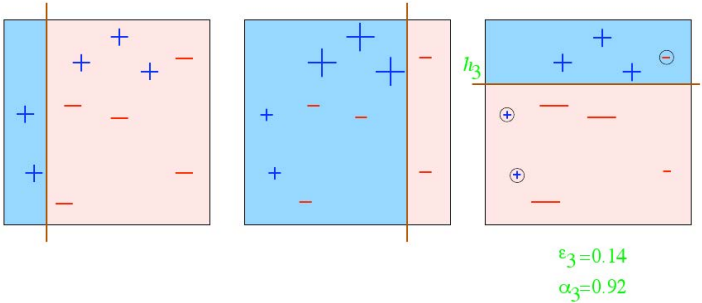
- $\varepsilon_t = \sum_i D_t(i)\mathbb{I}(h_t(\mathbf{x}_i) \neq y_i)$: error
- $\alpha_1 = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$: magic coefficient

Adaboost round 2



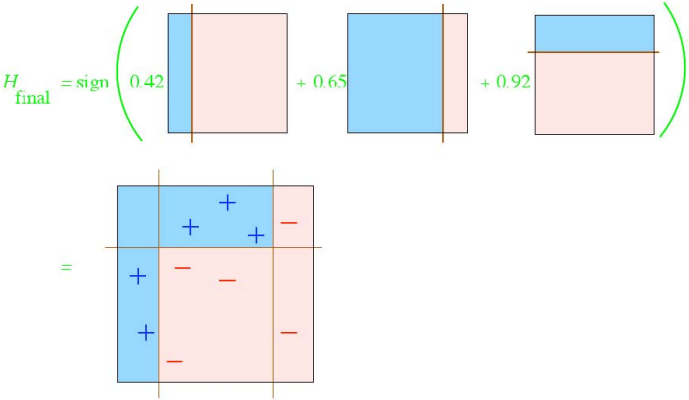
- $\varepsilon_t = \sum_i D_t(i)\mathbb{I}(h_t(\mathbf{x}_i) \neq y_i)$: error
- $\alpha_1 = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$: magic coefficient

Adaboost round 3



- $\varepsilon_t = \sum_i D_t(i)\mathbb{I}(h_t(\mathbf{x}_i) \neq y_i)$: error
- $\alpha_1 = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$: magic coefficient

Adaboost final classifier



Gradient Boosting

What is Gradient Boosting

- **Gradient Boosting = Gradient Descent + Boosting**
- Gradient Boosting
  - Fit an additive model (ensemble) $\sum_t \rho_t h_t(x)$ in a forward stage-wise manner.
  - 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.

Gradient Boosting for Regression

Let's play a game...

You are given $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, and the task is to fit a model $F(x)$ to minimize 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: $F(x_1) = 0.8$, while $y_1 = 0.9$, and $F(x_2) = 1.4$ 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) h to F, so the new prediction will be $F(x) + h(x)$.

Gradient Boosting for Regression

Simple solution: You wish to improve the model such that

$$F(x_1) + h(x_1) = y_1$$

$$F(x_2) + h(x_2) = y_2$$

$$\cdots$$

$$F(x_n) + h(x_n) = y_n$$

Gradient Boosting for Regression

Simple solution: Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\dots$$
$$h(x_n) = y_n - F(x_n)$$

Can any regression model achieve goal perfectly?

Gradient Boosting for Regression

Simple solution: Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\dots$$
$$h(x_n) = y_n - F(x_n)$$

Can any regression model achieve goal perfectly?
Maybe not ...

Gradient Boosting for Regression

Simple solution: Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\dots$$
$$h(x_n) = y_n - F(x_n)$$

Can any regression model achieve goal perfectly?
Maybe not ...
But some regression models might be able to do this approximately.

Gradient Boosting for Regression

Simple solution: Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\dots$$
$$h(x_n) = y_n - F(x_n)$$

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

Gradient Boosting for Regression

Simple solution: Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\dots$$
$$h(x_n) = y_n - F(x_n)$$

Can any regression model achieve goal perfectly?
Maybe not ...
But some regression models might be able to do this approximately.
How?
Just fit a regression model h to data

$$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), ..., (x_n, y_n - F(x_n))$$

Gradient Boosting for Regression

Simple solution: Or, equivalently, you wish

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\dots$$
$$h(x_n) = y_n - F(x_n)$$

Can any regression model achieve goal perfectly?
Maybe not ...
But some regression models might be able to do this approximately.
How?
Just fit a regression model h to data

$$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), ..., (x_n, y_n - F(x_n))$$

Congratulations, you get a better model!

Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$ 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:

$y_i - F(x_i)$ 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,

Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$ 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 model...

Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$ 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 model...

We are improving the predictions of training data, is the procedure also useful for test data?

Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$ 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 model...

We are improving the predictions of training data, is the procedure also useful for test data?

Yes! Because we are building a model, and the model can be applied to test data as well.

Gradient Boosting for Regression

Simple solution:

$y_i - F(x_i)$ 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 model...

We are improving the predictions of training data, is the procedure also useful for test data?

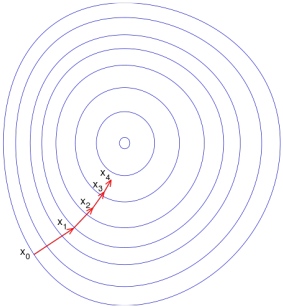
Yes! Because we are building a model, and the model can be applied to test data as well.

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.

$$\theta_i = \theta_i - \rho \frac{\partial J}{\partial \theta_i}, \quad \boldsymbol{\theta} = \boldsymbol{\theta} - \rho \nabla_{\boldsymbol{\theta}} J$$



Gradient Boosting for Regression

How is this related to gradient descent?

Loss function $L(y, F(x)) = (y - F(x))^2/2$

We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), \ldots, F(x_n)$.

Notice that $F(x_1), F(x_2), \ldots, F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives.

Gradient Boosting for Regression

How is this related to gradient descent?

Loss function $L(y, F(x)) = (y - F(x))^2/2$

We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), \ldots, F(x_n)$.

Notice that $F(x_1), F(x_2), \ldots, F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives.

$$\frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients:

$$-\frac{\sum_i L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

Gradient Boosting for Regression

How is this related to gradient descent?

$$F(x_i) = F(x_i) + h(x_i)$$
$$F(x_i) = F(x_i) + y_i - F(x_i)$$
$$F(x_i) = F(x_i) - 1\frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)}$$
$$\theta_i = \theta_i - \rho\frac{\partial J}{\partial \theta_i}$$

Gradient Boosting for Regression

How is this related to gradient descent?

For regression with **square loss**,

$$\text{residual} = \text{negative gradient}$$

$$\text{fit } h \text{ to residual} = \text{fit } h \text{ to negative gradient}$$

$$\text{update } F \text{ based on residual} = \text{update } F \text{ based on negative gradient}$$

Gradient Boosting for Regression

How is this related to gradient descent?

For regression with **square loss**,

$$\text{residual} = \text{negative gradient}$$

$$\text{fit } h \text{ to residual} = \text{fit } h \text{ to negative gradient}$$

$$\text{update } F \text{ based on residual} = \text{update } F \text{ based on negative gradient}$$

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**,

$$\text{residual} = \text{negative gradient}$$
$$\text{fit } h \text{ to residual} = \text{fit } h \text{ to negative gradient}$$
$$\text{update } F \text{ based on residual} = \text{update } F \text{ based on negative gradient}$$

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**. So from now on, let's stick with gradients. The reason will be explained later.

Gradient Boosting for Regression

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

$$-g(x_i) = -\frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

1. start with an initial model, say, $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
2. iterate until converge:
   - calculate negative gradients $-g(x_i)$
   - fit a regression model h to negative gradients $-g(x_i)$
   - $F = F + \rho h$, where $\rho = 1$

Gradient Boosting for Regression

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

$$-g(x_i) = -\frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

1. start with an initial model, say, $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
2. iterate until converge:
   - calculate negative gradients $-g(x_i)$
   - fit a regression model h to negative gradients $-g(x_i)$
   - $F = F + \rho h$, 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

Loss Functions for Regression Problem

Why do we need to consider other loss functions? Isn't square loss good enough?

Gradient Boosting for Regression

Loss Functions for Regression Problem

Why do we need to consider other loss functions? Isn't square loss good enough?

Square loss is not robust to outliers

- Outliers are heavily punished because the error is squared.

y_i	0.5	1.2	2	5*
$F(x_i)$	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

Why do we need to consider other loss functions? Isn't square loss good enough?

Square loss is not robust to outliers

- Outliers are heavily punished because the error is squared.

y_i	0.5	1.2	2	5*
$F(x_i)$	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)

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

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)

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

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)

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

y_i	0.5	1.2	2	5*
$F(x_i)$	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 loss function L: general procedure

Given any (sub)differentiable loss function L

1. start with an initial model, say, $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
2. iterate until converge:
   - calculate negative gradients $-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$
   - fit a regression model h to negative gradients $-g(x_i)$
   - $F = F + \rho h$, where $\rho = 1$

Gradient Boosting for Regression

Regression with loss function L: general procedure

Given any (sub)differentiable loss function L

1. start with an initial model, say, $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
2. iterate until converge:
   - calculate negative gradients $-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$
   - fit a regression model h to negative gradients $-g(x_i)$
   - $F = F + \rho h$, where $\rho = 1$

In general,

$$\text{negative gradients} \neq \text{residuals}$$

We should follow negative gradients rather than residuals. Why?

Gradient Boosting for Regression

Summary of the Section

- Fit an additive model $F = \sum_t \rho_t h_t$ in a forward stage-wise manner.
- In each stage, introduce a new regression tree h to compensate the shortcomings of existing model.
- The "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.

Things not covered

How to choose a proper learning rate for each gradient boosting algorithm. See [Friedman, 2001]

Conclusion Remarks

- Machine learning
  - Focus on generalization (testing)
  - Not necessarily base on probability
  - A rich amount of linear and non-linear models for typical classification and regression
- Advanced topics
  - Sparse learning
  - Transfer learning and multi-task learning
  - Multi-modality learning
  - Robust learning
  - ...