

# Advanced Business Data Mining

MSIS 522 – Lesson 3

---

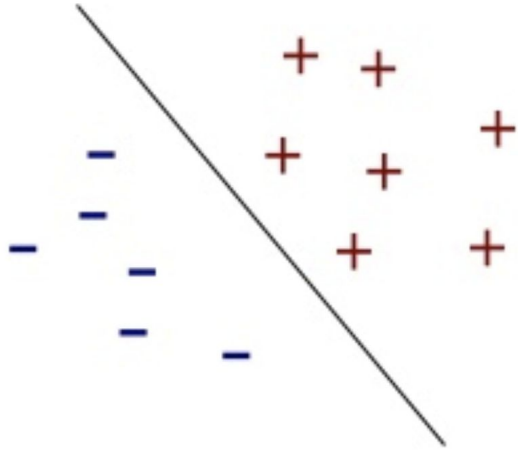
# Course Overview

- Lecture 1 - Fundamentals of Machine Learning
- Lecture 2 - Decision Tree
- **Lecture 3 - Ensemble Learning**
- Lecture 4 - Clustering
- Lecture 5 - Recommendation Systems

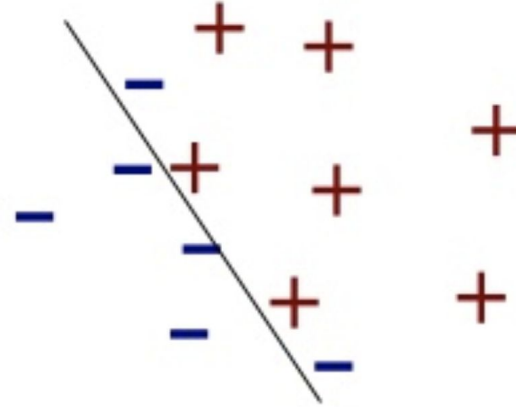
# Recap of Lesson 2

---

# Linear Separable



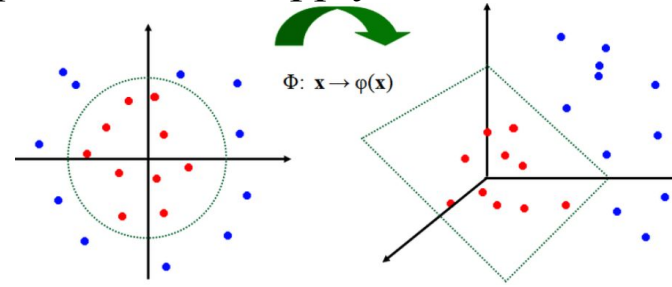
Linear Separable



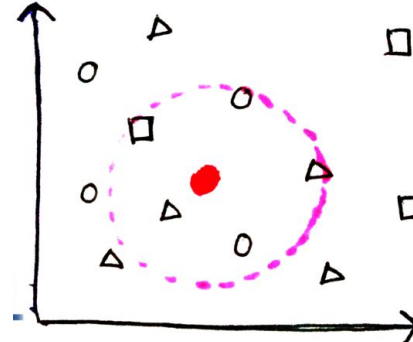
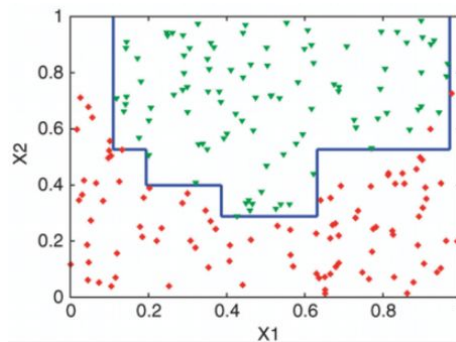
Not Linear Separable

# Handle Nonlinear Separable Data

- Project existing data into other dimensional space so that data becomes linear separable in that space and then apply a linear model.

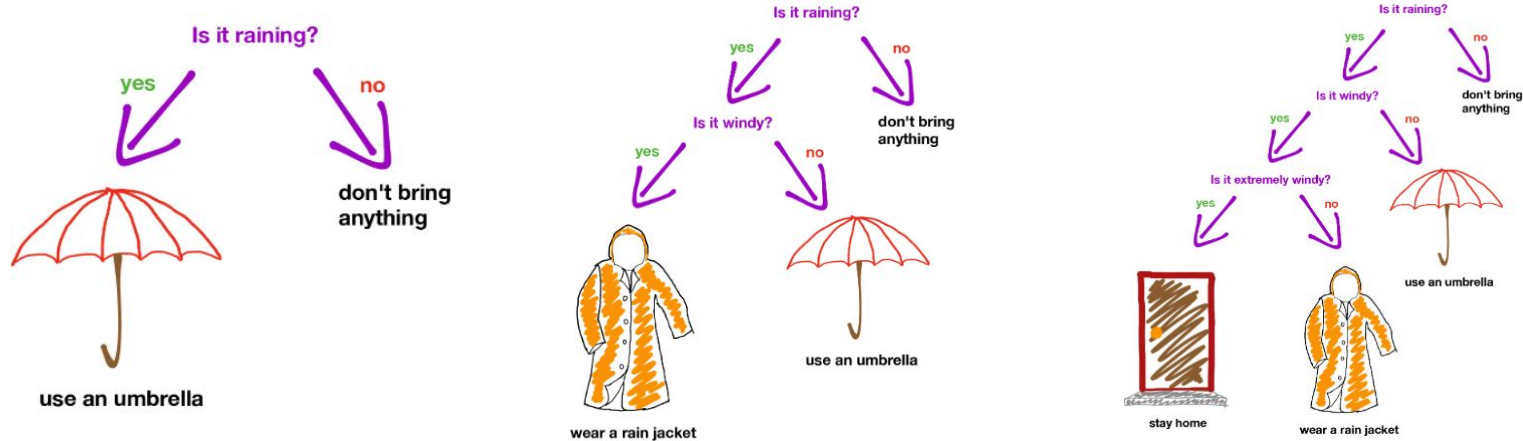


- Use a more powerful model which can model non-linearity in the data by itself.



# How to construct a Decision Tree?

- Choose an attribute (i.e. a feature) for root.
- Split data using chosen attribute into disjoint subsets.
- Recursive partitioning for each subset.



# Split of a Categorical Variable

- Examine all possible ways in which the categories can be split into two groups.

- E.g. categories A, B, C can be split 3 ways.

- {A} and {B, C}
- {B} and {A, C}
- {C} and {A, B}

- In theory, we have an exponential number of different splits.

- In practice, we often use one vs the rest.



# Classification Impurity Measure: Entropy

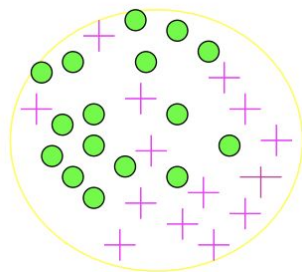
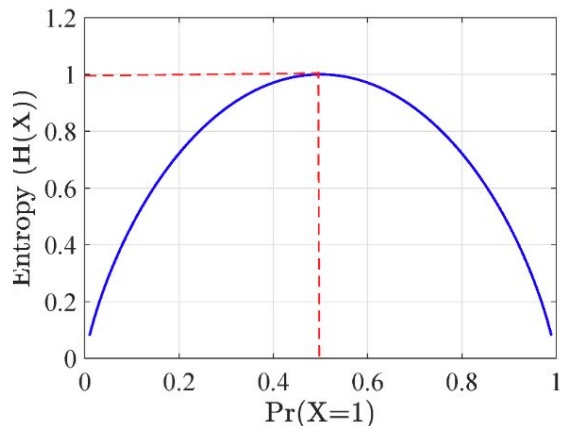
- **Entropy** measures the level of **impurity** in a group of examples for classification problems.

$$H(x) = - \sum_i p_i \log(p_i)$$

16/30 are green circles; 14/30 are pink crosses

$\log_2(16/30) = -.9$ ;  $\log_2(14/30) = -1.1$

Entropy =  $-(16/30)(-.9) - (14/30)(-1.1) = .99$





# Classification Impurity Measure: Entropy

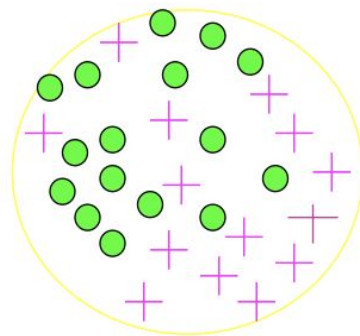
- Entropy measures the level of impurity in a group of examples.

$$H(x) = - \sum_i p_i \log(p_i)$$

16/30 are green circles; 14/30 are pink crosses

$\log_2(16/30) = -.9$ ;  $\log_2(14/30) = -1.1$

Entropy =  $-(16/30)(-.9) - (14/30)(-1.1) = .99$



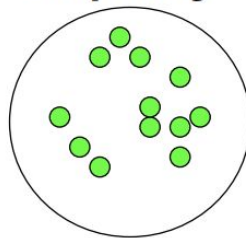
## Entropy for 2-class Cases

- What is the entropy of a group in which all examples belong to the same class?

– entropy =  $-1 \log_2 1 = 0$

not a good training set for learning

Minimum  
impurity

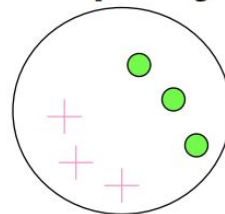


- What is the entropy of a group with 50% in either class?

– entropy =  $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$

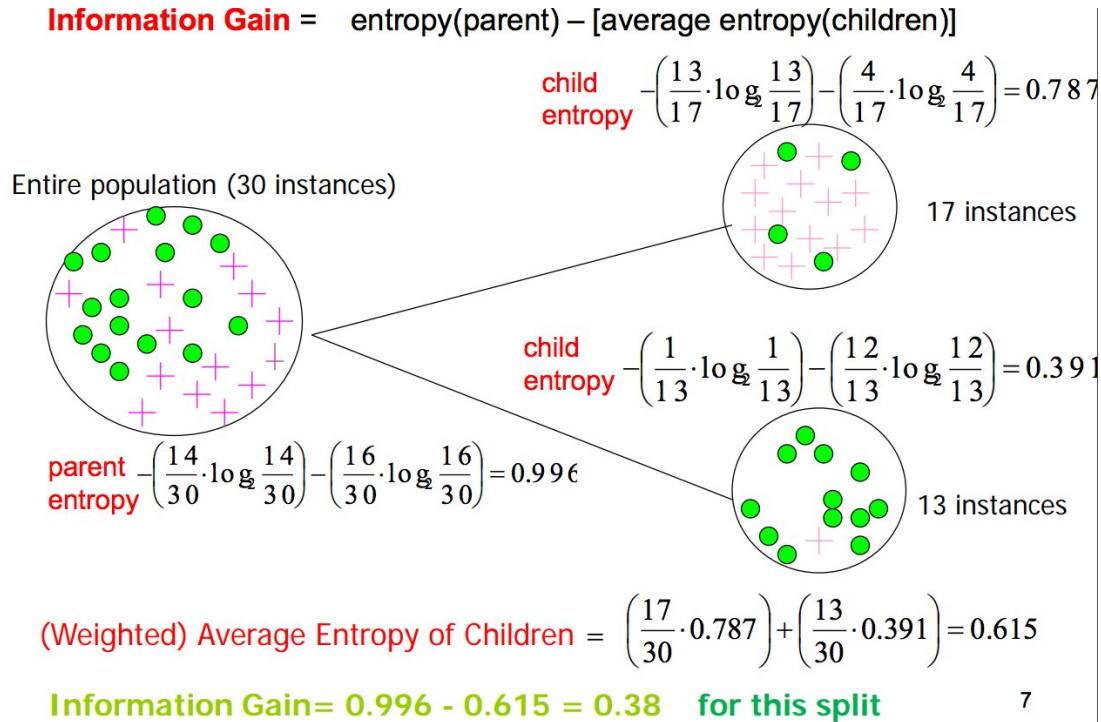
good training set for learning

Maximum  
impurity



# Information Gain Example

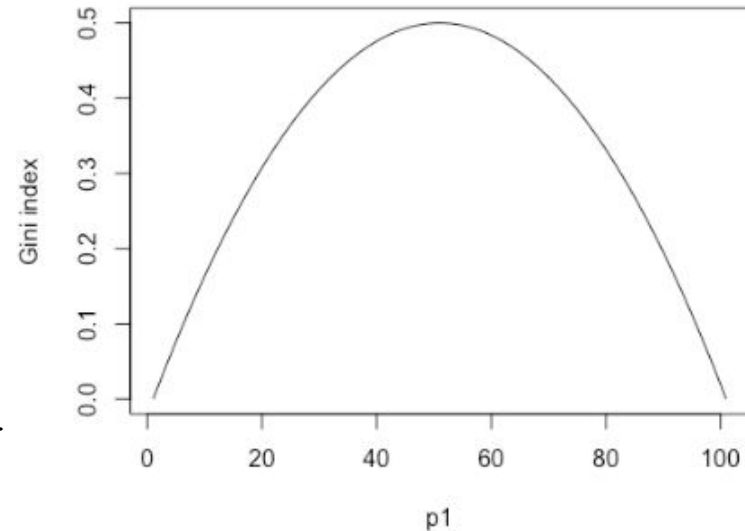
Information Gain determines **which attribute** is most useful for discriminating between the classes.



# Classification Impurity Measure: Gini Impurity

$$I_G = 1 - \sum_{j=1}^c p_j^2$$

- **Gini impurity/index** is a measure to quantify the level of impurity in a group of examples.
  - $I(A) = 0$  when all cases belong to the same class.
  - Max value when all classes are equally represented.



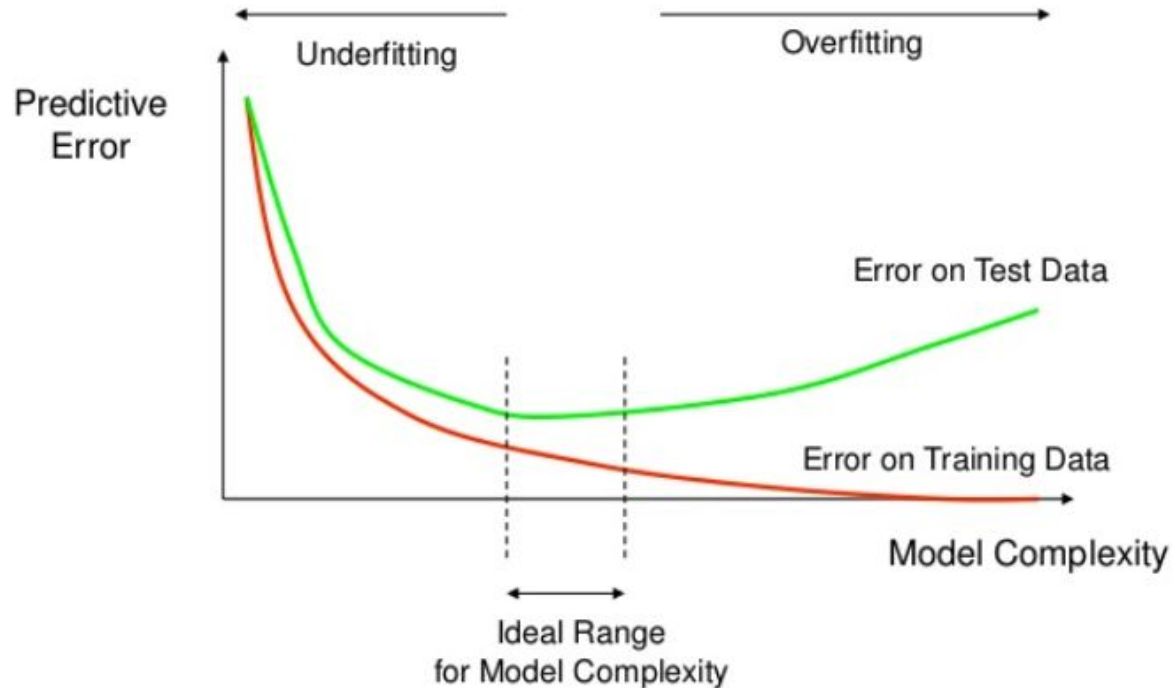
# Split of a Numerical Variable

- For each numerical attribute:
  - Sort the attribute from the smallest to the largest.
  - Linearly scan these values and choose the split position leading to the maximum impurity reduction (i.e. information gain).

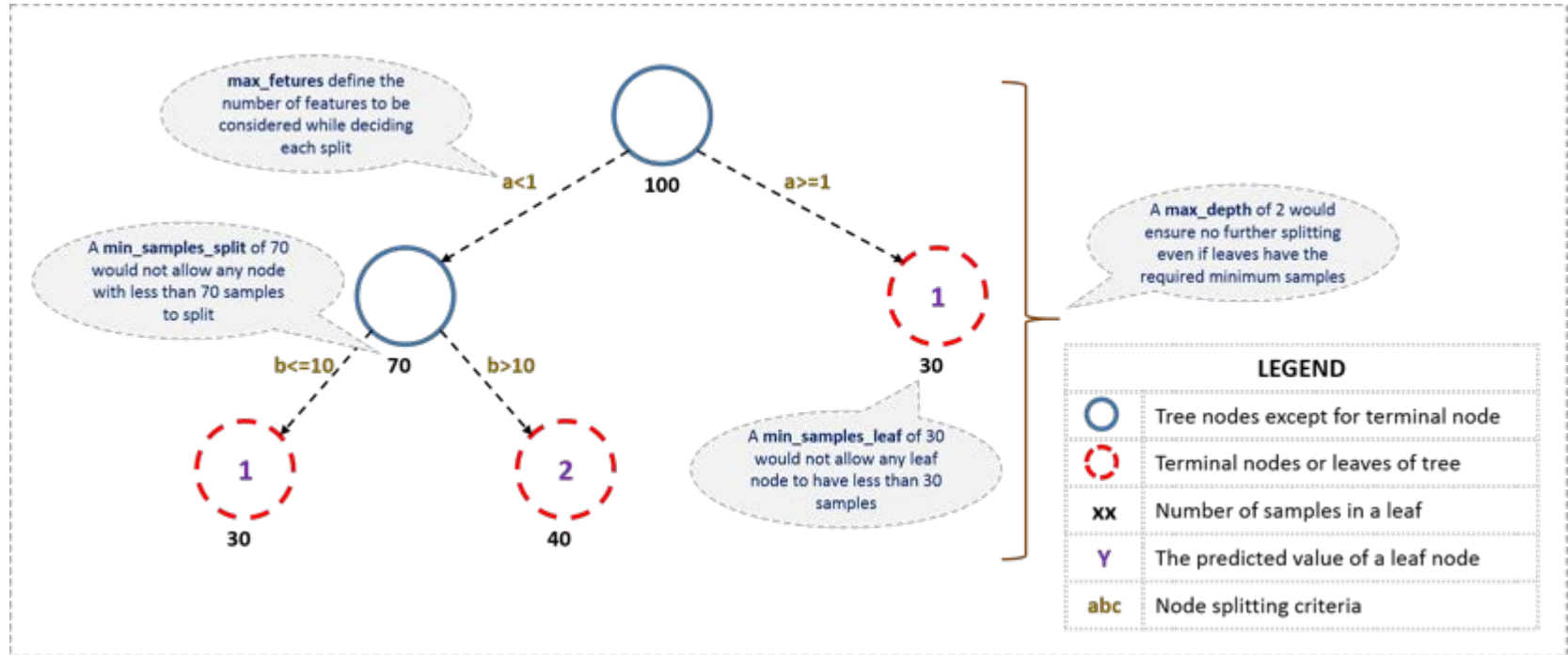
		Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
		Taxable Income										
Sorted Values	→	60	70	75	85	90	95	100	120	125	220	
Split Positions	→	55	65	72	80	87	92	97	110	122	172	230
		≤ >	≤ >	≤ >	≤ >	≤ >	≤ >	≤ >	≤ >	≤ >	≤ >	≤ >
	Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	3 0
	No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0
	Gini	0.420	0.400	0.375	0.343	0.417	0.400	0.300	0.343	0.375	0.400	0.420

# Generalization

Machine Learning is all about **generalization** to future unseen data points.

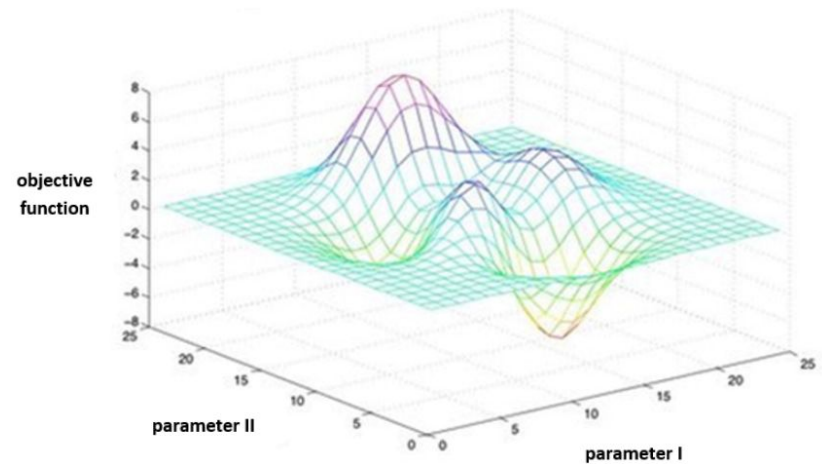
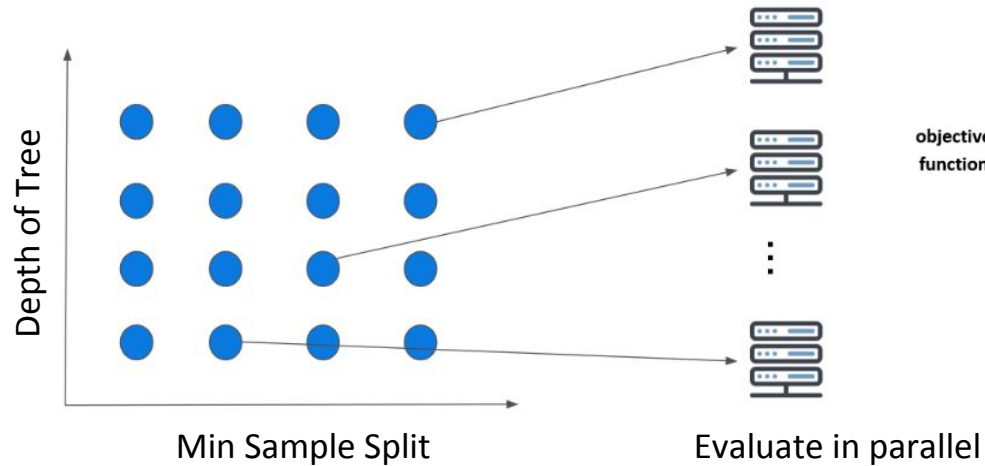


# Hyper-parameters of Decision Tree



# Grid Search

Find the best configuration for the hyper-parameters used in a ML model.





# Outline

---

- Weak Learner vs. Strong Learner
- Ensemble Learning
  - Bagging
  - Boosting
- Lab

# Ensemble Learning



# Weak Learner & Strong Learner

- A **weak learner**: it can make predictions (slightly) **better than random guessing**.
  - Weak learners has high bias and cannot solve hard learning problems.
  - e.g., naïve Bayes, logistic regression, decision stumps (decision trees of depth 1)
- A **strong learner**: it has **arbitrarily small error rate**.
  - Strong learners are our goal of machine learning.
  - e.g. random forest, deep neural networks

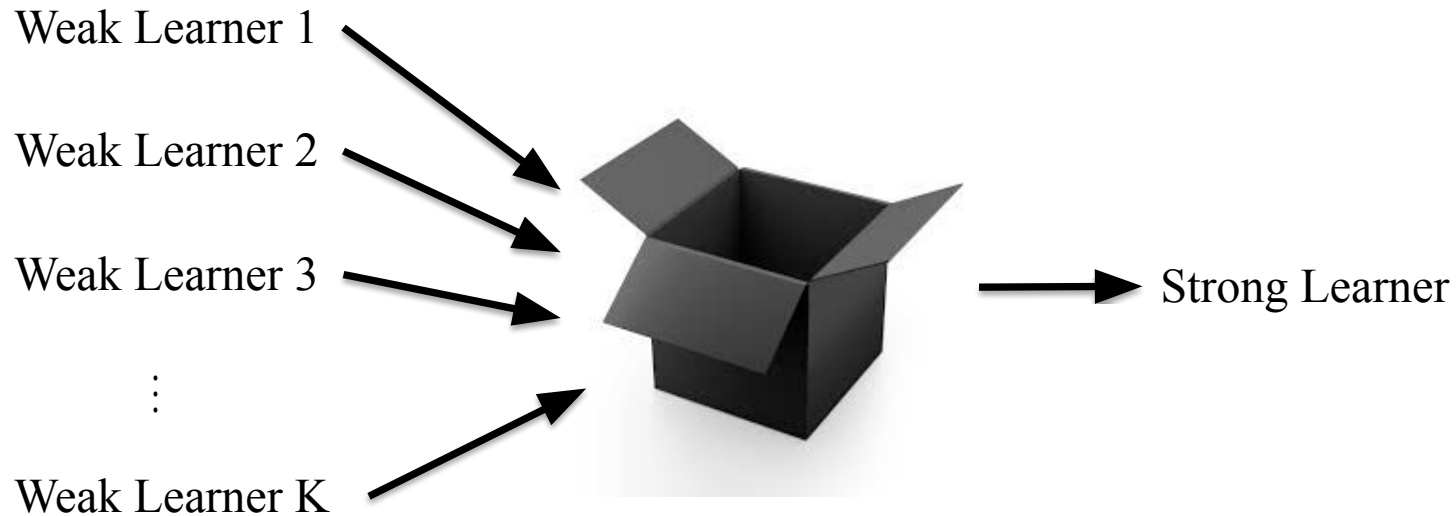
## Example

**Goal:** Automatically categorize type of call requested  
(Collect, Calling card, Person-to-person, etc.)

- yes I'd like to place a collect call long distance please (Collect)
- operator I need to make a call but I need to bill it to my office (ThirdNumber)
- yes I'd like to place a call on my master card please (CallingCard)

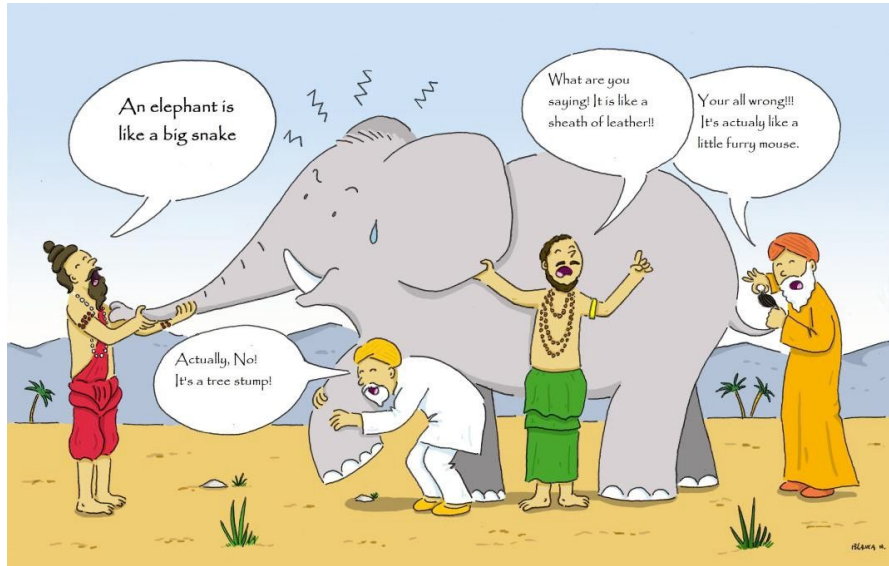
- **Easy to find “rules of thumb” that are often better than random guessing (i.e. weak learner).**
  - E.g. predict *Calling card* if ‘card’ occurs in utterance.
- **Hard to find a single highly accurate prediction rule (i.e. strong learner).**

# Can we turn weak learners into a strong one?

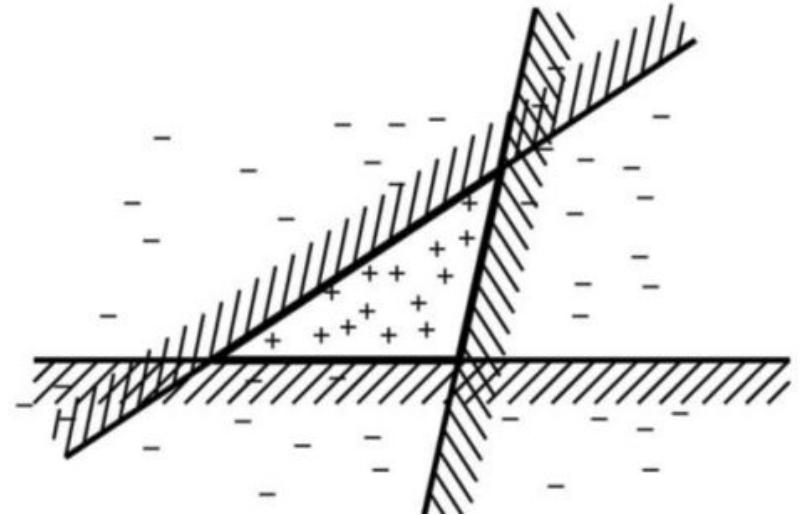


**YES, ENSEMBLE LEARNING**

# Ensemble Learning Intuition



Fable of blind men and elephant



Combine 3 linear classifiers

# Ensemble Learning

- Instead of learning a single classifier, we learn a set of classifiers.

How do we learn a set of classifiers?

- Combine the predictions of multiple classifiers to produce the final prediction.

How do we combine all the classifiers?

# Why do ensembles work?

- Suppose there are 25 classifiers where each classifier has an error rate of 0.35.
  - Assume classifiers are **independent**: a mistake from one classifier does not depend on the predictions from other classifiers.
  - In practice they are NOT completely independent.
- *Majority Voting*: The ensemble makes a wrong prediction if the majority of the classifiers predict the wrong prediction.
- What is the probability that the ensemble makes a wrong prediction? (hint: 13 or more classifiers make wrong predictions).

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1 - \varepsilon)^{25-i} \approx 0.06 \quad \text{vs. } 0.35 \text{ from any base classifiers}$$



# Ensemble Learning in Netflix Prize

Machine learning competition with a \$1 million prize

**Leaderboard** Display top 20 leaders.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	<b>The Ensemble</b>	0.8553	10.10	2009-07-26 18:38:22
2	PragmaticTheory	0.8554	10.09	2009-07-26 18:18:28
<b>Grand Prize - RMSE &lt;= 0.8563</b>				
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	Qace	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	Feedz	0.8613	9.47	2009-07-24 20:06:46
<b>Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos</b>				
13	Wanliang	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
<b>Progress Prize 2007 - RMSE = 0.8712 - Winning Team: KorBell</b>				
<b>Cinematch score on quiz subset - RMSE = 0.9514</b>				



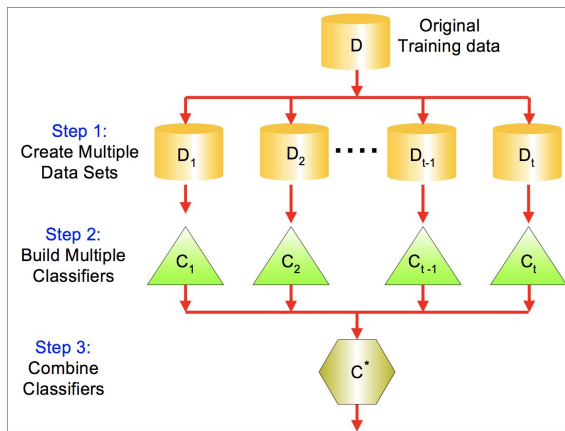
**The Ensemble** was an ensemble solution of teams which had been competing individually for the prize.

# Bagging

---

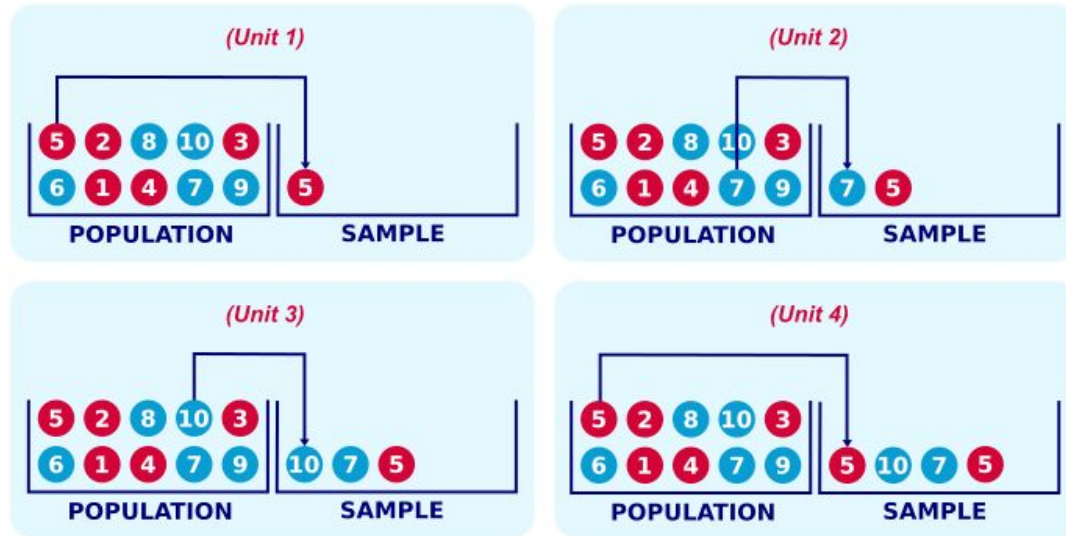
# Bagging

- **Bootstrap aggregation**, or **bagging**, is a general-purpose procedure for **reducing the variance** of a machine learning method by combining the result of multiple classifiers trained on different sub-samples of the same data set.
  - E.g. random forest.
- **Bagging:**
  - Step 1: Create bootstrap samples (sample with replacement).
  - Step 2: Train separate classifier on each sample.
  - Step 3: Classify new data point by majority vote or average.

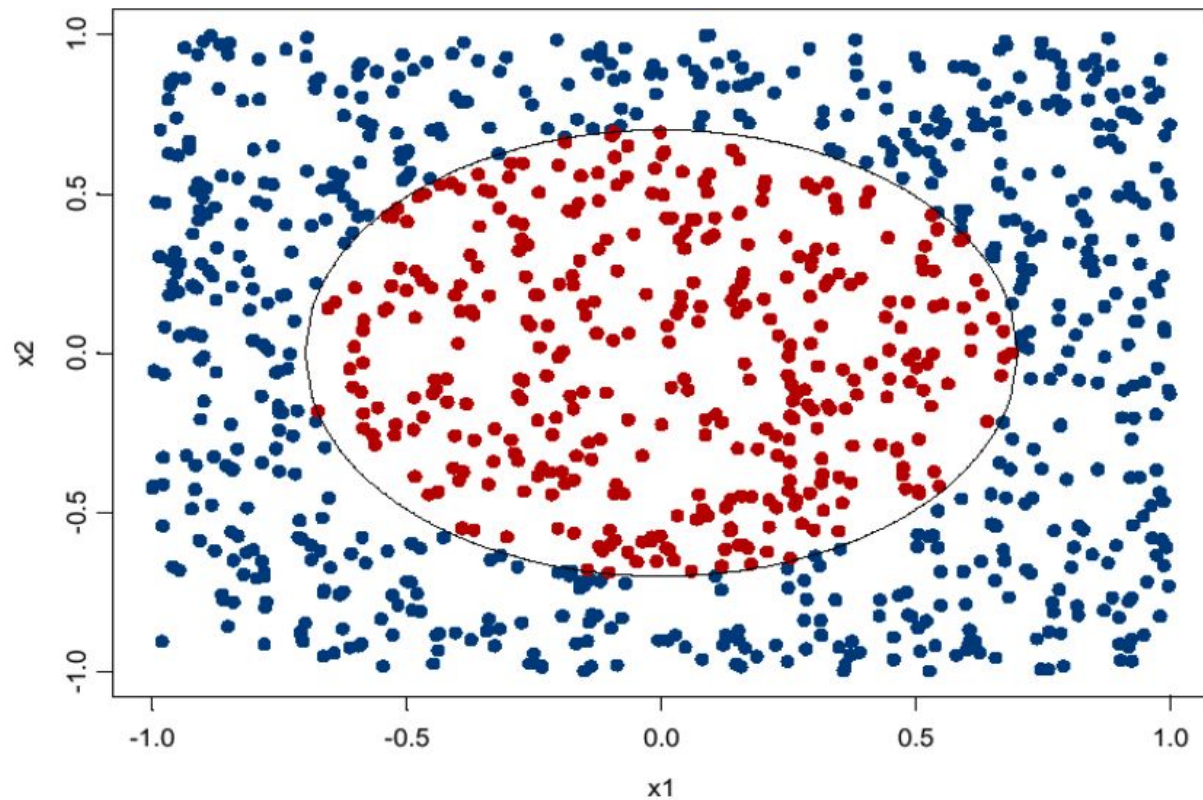


# Bootstrapping

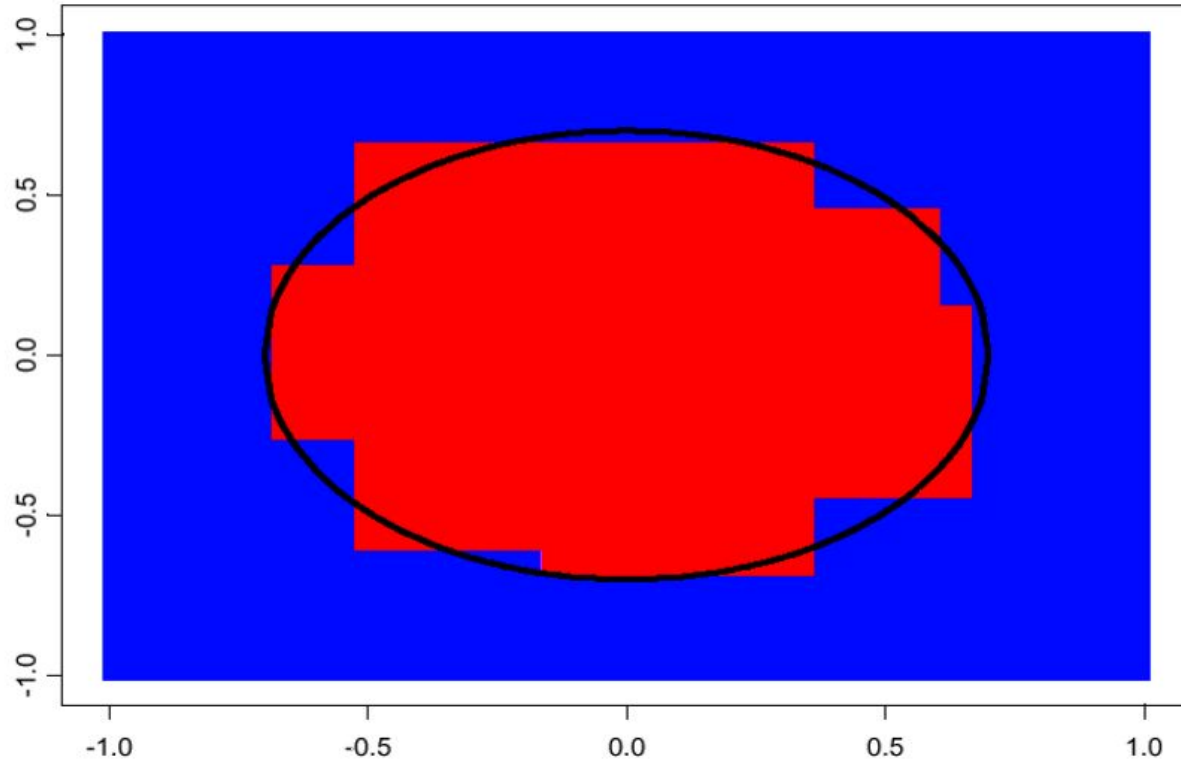
Bootstrapping: sample with replacement.



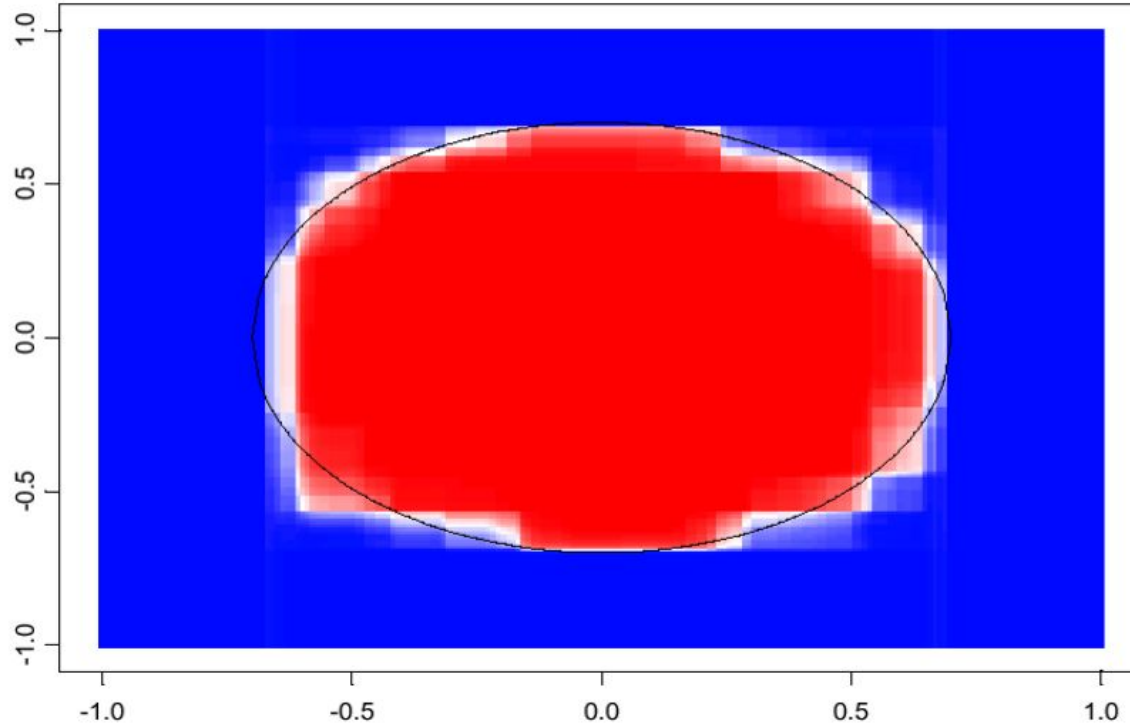
# Ground Truth



# Decision Boundary from a Single Decision Tree



# Decision Boundary from 100 Decision Tree



shades of blue/red indicate strength of vote for particular classification

# Random Forest

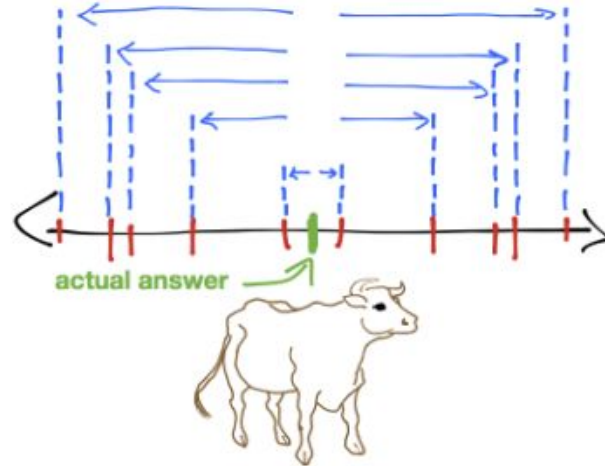




# Intuition behind Random Forest

There was a fair where visitors would guess the weight of a bull, and whoever got the actual weight won a prize. The organizer noticed that although none of the individual guesses were exactly correct, the *average* of the guesses was surprisingly accurate.

Inaccurate guesses cancel each other out



# Random Forest Algorithm

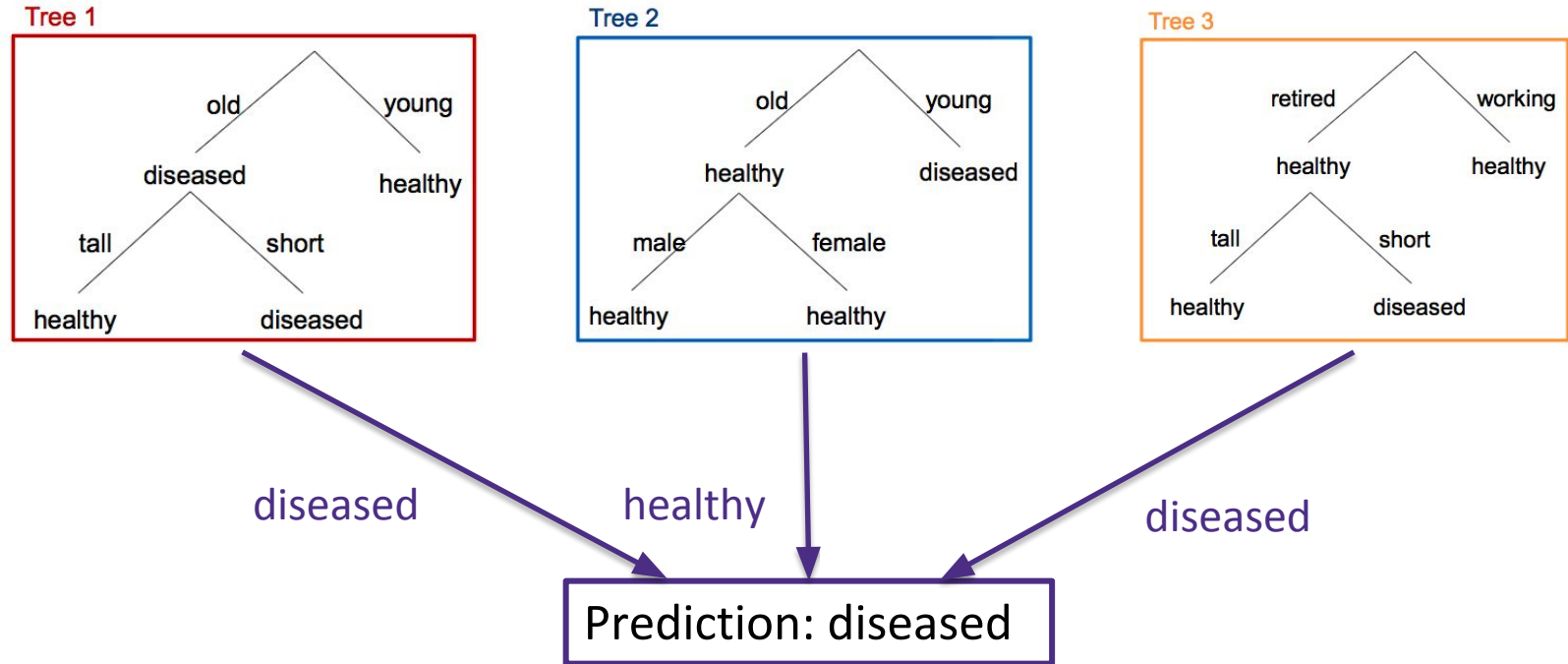
## Training

- Draw  $M$  bootstrap samples (sample with replacement) of size  $N$  from the training data.
- Build a decision tree for each bootstrap sample.
  - Select  $m$  variables at random from all variables
  - Pick the best variable and split-point among those  $m$  variables.
  - Split the node into two child nodes.
- Output the ensemble of trees (one decision tree from each bootstrap sample).

## Making a prediction

- **Regression:** output the average of all  $M$  predictions, one from each decision tree.
- **Classification:** use majority voting where the class with the most votes is the final prediction.

# Random Forest Quiz



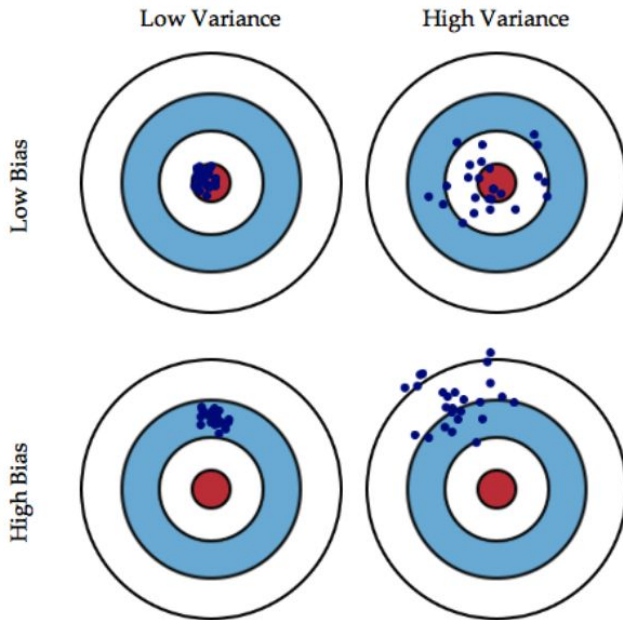
**New Sample:**

- old, retired, male, short

# Bias/Variance Tradeoff

$$\text{ERROR} = \text{BIAS} + \text{VARIANCE} + \text{NOISE}$$

- **Bias:** the difference between the predicted value and the actual value.
  - Model underfits the training data and fails to capture the underlying pattern within the data.
  - e.g. Linear model
- **Variance:** the variability of a model prediction for a given data point.
  - Learning is not stable. A small change in the training data or hyper-parameter can lead to a very different model/prediction.
  - E.g. decision tree



## Math behind Random Forest

$$\begin{aligned} E_P [(y - h(\mathbf{x}))^2] &= E_P [(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2] + \\ &\quad \bar{h}(\mathbf{x})^2 - 2f(\mathbf{x})\bar{h}(\mathbf{x}) + f(\mathbf{x})^2 + \\ &\quad E_P [(y - f(\mathbf{x}))^2] \\ &= E_P [(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2] + \quad (\text{variance}) \\ &\quad (h(\mathbf{x}) - f(\mathbf{x}))^2 + \quad (\text{bias})^2 \\ &\quad E_P [(y - f(\mathbf{x}))^2] \quad (\text{noise}) \\ &= \text{Var}[h(\mathbf{x})] + \text{Bias}[h(\mathbf{x})]^2 + E_P[\varepsilon^2] \\ &= \text{Var}[h(\mathbf{x})] + \text{Bias}[h(\mathbf{x})]^2 + \sigma^2 \end{aligned}$$

Expected prediction error = Variance + Bias<sup>2</sup> + Noise<sup>2</sup>

## Math behind Random Forest Continued

- > For each data point  $x$ , we now have the observed corresponding value  $y$  and  $K$  predicted values  $\{z_1, z_2, \dots, z_K\}$  from each decision tree in the forest.
- > Compute the average prediction  $\bar{z}$ .
- > Estimate bias as  $(\bar{z} - y)$
- > Estimate variance as  $\sum_k (z_k - \bar{z}) / (K - 1)$

Bias does not  
change.

Variance decreases  
by  $K-1$  fold.

# Main Ideas of Random Forest

Introduce two sources of randomness.

- **Bagging:** Bootstrap the data.
- **Random input vector:** At each node, best split is chosen from a random sample of attributes instead of all attributes.

1. For  $b = 1$  to  $B$ :

- (a) Draw a **bootstrap sample**  $\mathbf{Z}^*$  of size  $N$  from the training data.
- (b) Grow a random-forest tree  $T_b$  to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size  $n_{min}$  is reached.
  - i. Select  **$m$  variables at random** from the  $p$  variables.
  - ii. Pick the best variable/split-point among the  $m$ .
  - iii. 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:*  $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x).$

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

# Random Forest options in Scikit-learn

Parameter	Description
n_estimators	number of tree
criterion	"gini" or "entropy"
max_features	The number of features to consider when looking for the best split
max_depth	The maximum depth of the tree
min_samples_split	The minimum number of samples required to split an internal node
min_samples_leaf	The minimum number of samples required to be at a leaf node
min_weight_fraction_leaf	The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node.
max_leaf_nodes	Grow trees with max_leaf_nodes in best-first fashion.
min_impurity_split	Threshold for early stopping in tree growth.
bootstrap	Whether bootstrap samples are used when building trees.
oob_score	Whether to use out-of-bag samples to estimate the generalization accuracy.
warm_start	When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.



## Why does bagging work?

- > Bagging reduces **variance** by averaging the predictions from multiple classifiers.

$$Var(\bar{X}) = \frac{Var(X)}{N} \quad \text{(when prediction are independent)}$$

- > Bagging has little effect on **bias**.
- > Can we average and reduce both **bias** and **variance**?

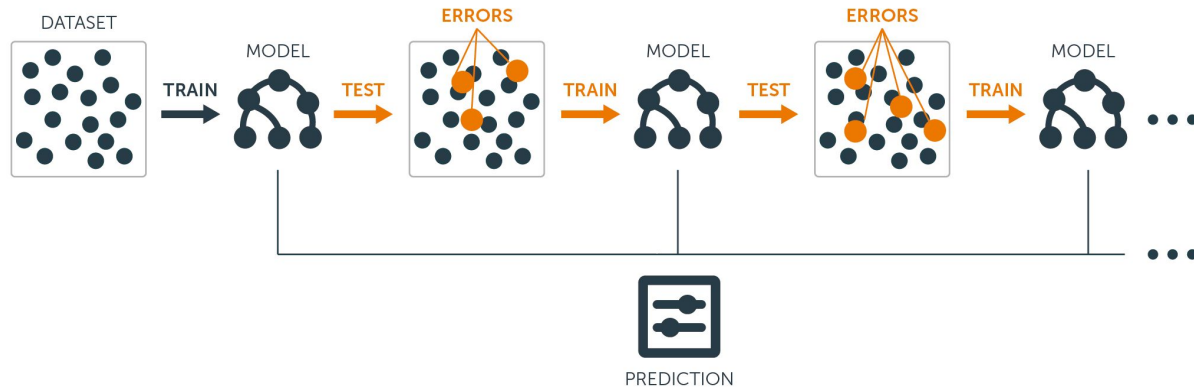
Yes: Boosting

# Boosting

---

# Main Idea of Boosting

- **Learn classifiers in sequence:** later classifiers focus on examples that were misclassified by earlier classifiers.
- **Weighted voting:** weight the predictions of the classifiers according to their prediction accuracy.

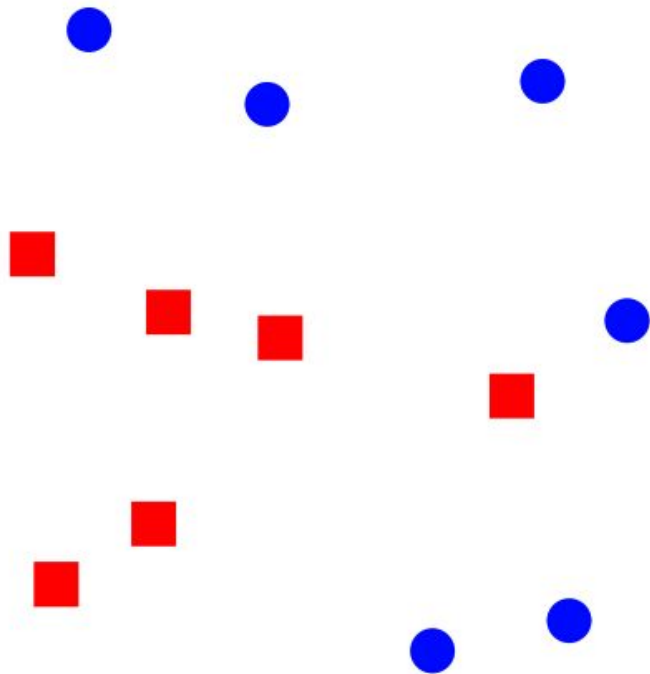


# Boosting Realization

- On each iteration  $t$ :
  - Weight each training example by how incorrectly it was classified by previous classifiers.
    - increasing the weight of incorrectly classified examples ensures that they will become more important in the next iteration.
  - Learn a classifier  $h_t(x)$  based on the weighted training data.
  - Calculate a strength factor  $\alpha_t$  for  $h_t(x)$  based on its accuracy.
- Final classifier:
  - Weighted voting of different classifiers where weight is their strength factor.

# Boosting Intuition

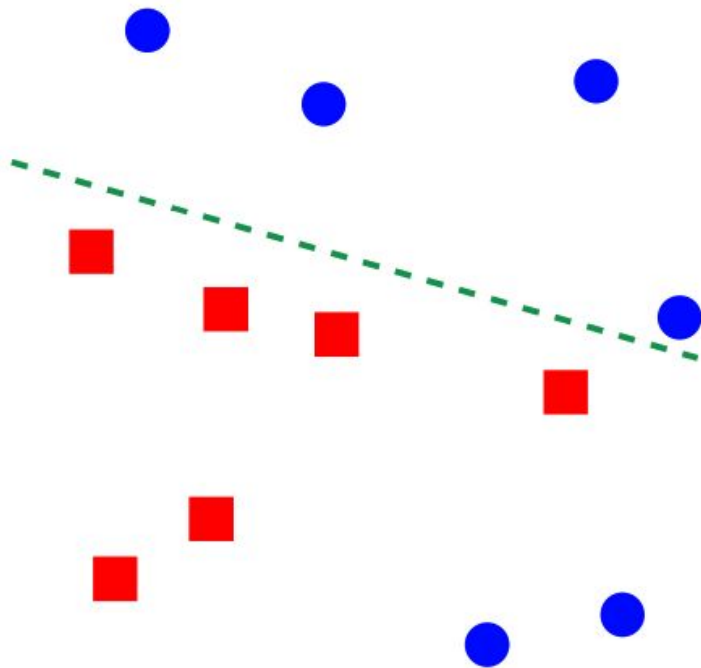
Goal: turn weak learners (e.g. linear model) into a strong learner.



- Pick a weak linear classifier  $h_t(x)$ .
- Adjust weights: misclassified examples get heavier weight.
- Calculate strength  $\alpha_t$  according to the weighted error of  $h_t(x)$ .

# Boosting Intuition

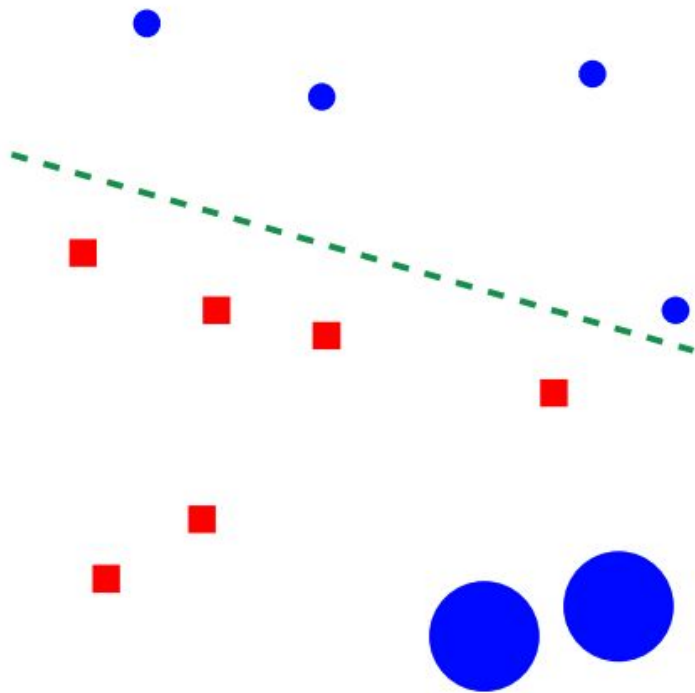
Goal: turn weak learners (e.g. linear model) into a strong learner.



- Pick a weak linear classifier  $h_t(x)$ .
- Adjust weights: misclassified examples get heavier weight.
- Calculate strength  $\alpha_t$  according to the weighted error of  $h_t(x)$ .

# Boosting Intuition

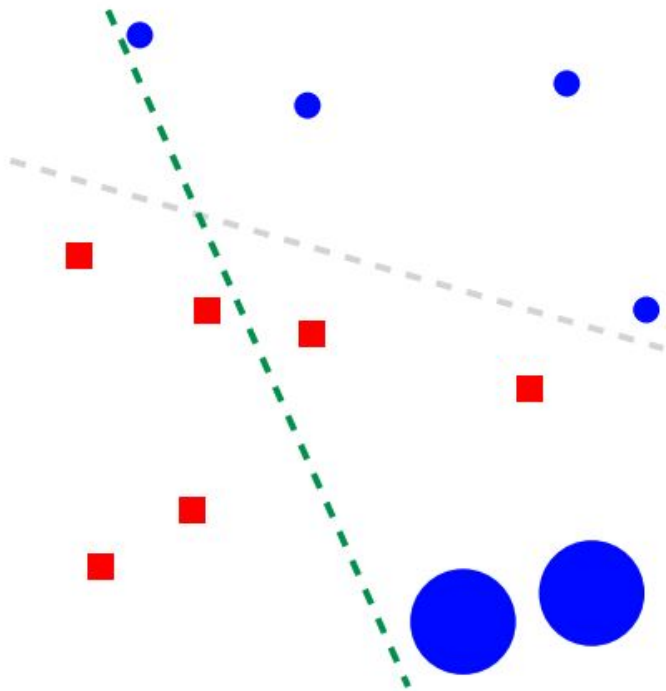
Goal: turn weak learners (e.g. linear model) into a strong learner.



- Pick a weak linear classifier  $h_t(x)$ .
- **Adjust weights: misclassified examples get heavier weight.**
- Calculate strength  $\alpha_t$  according to the weighted error of  $h_t(x)$ .

# Boosting Intuition

Goal: turn weak learners (e.g. linear model) into a strong learner.

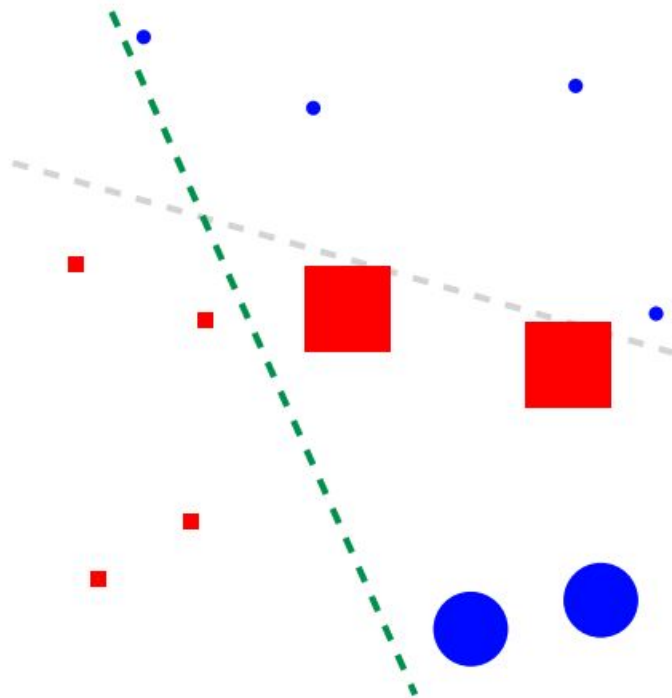


- Pick a weak linear classifier  $h_t(x)$ .
- Adjust weights: misclassified examples get heavier weight.
- Calculate strength  $\alpha_t$  according to the weighted error of  $h_t(x)$ .



# Boosting Intuition

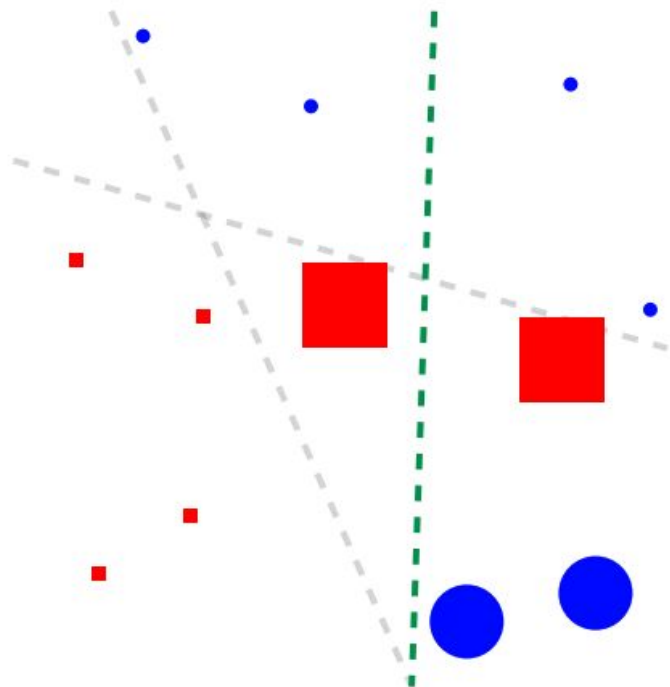
Goal: turn weak learners (e.g. linear model) into a strong learner.



- Pick a weak linear classifier  $h_t(x)$ .
- **Adjust weights: misclassified examples get heavier weight.**
- Calculate strength  $\alpha_t$  according to the weighted error of  $h_t(x)$ .

# Boosting Intuition

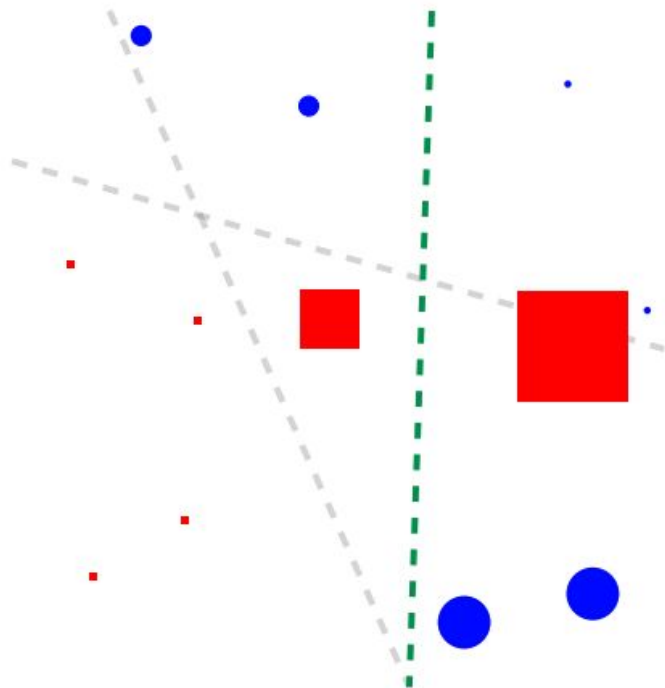
Goal: turn weak learners (e.g. linear model) into a strong learner.



- Pick a weak linear classifier  $h_t(x)$ .
- Adjust weights: misclassified examples get heavier weight.
- Calculate strength  $\alpha_t$  according to the weighted error of  $h_t(x)$ .

# Boosting Intuition

Goal: turn weak learners (e.g. linear model) into a strong learner.



- Pick a weak linear classifier  $h_t(x)$ .
- **Adjust weights: misclassified examples get heavier weight.**
- Calculate strength  $\alpha_t$  according to the weighted error of  $h_t(x)$ .

# Boosting history

[Schapire '89]:

- first provable boosting algorithm

[Freund '90]:

- “optimal” algorithm that “boosts by majority”

[Drucker, Schapire & Simard '92]:

- first experiments using boosting
- limited by practical drawbacks

[Freund & Schapire '95]:

- introduced “**AdaBoost**” algorithm
- strong practical advantages over previous boosting algorithms

# AdaBoost

Given:  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}$ ,  $y_i \in \{-1, +1\}$ .

Initialize:  $D_1(i) = 1/m$  for  $i = 1, \dots, m$ .

For  $t = 1, \dots, T$ :

- Train weak learner using distribution  $D_t$ .
- Get weak hypothesis  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$ .
- Aim: select  $h_t$  with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

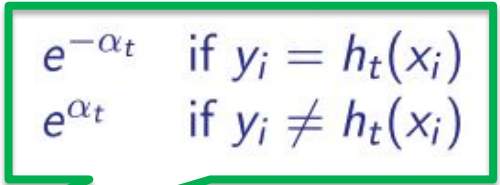
- Choose  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$ .
- Update, for  $i = 1, \dots, m$ :

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

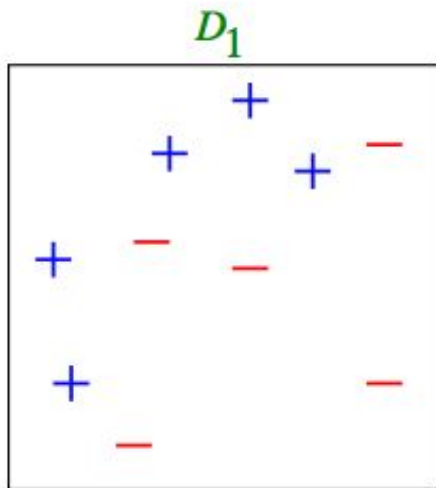
where  $Z_t$  is a normalization factor (chosen so that  $D_{t+1}$  will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right).$$


$$\begin{array}{ll} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{array}$$

# Toy Example



Minimize the error

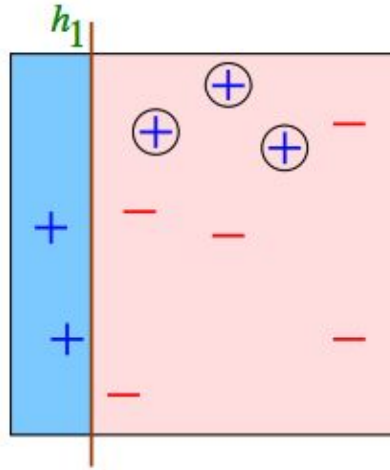
$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i]$$

For binary  $h_t$  , typically use

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

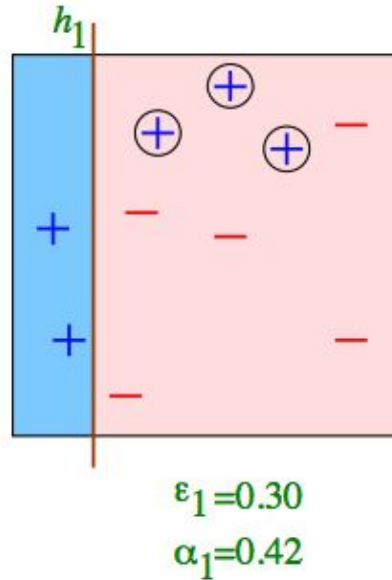
**weak classifiers = vertical or horizontal half-planes**

# Toy Example – Round 1



# Toy Example – Round 1

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

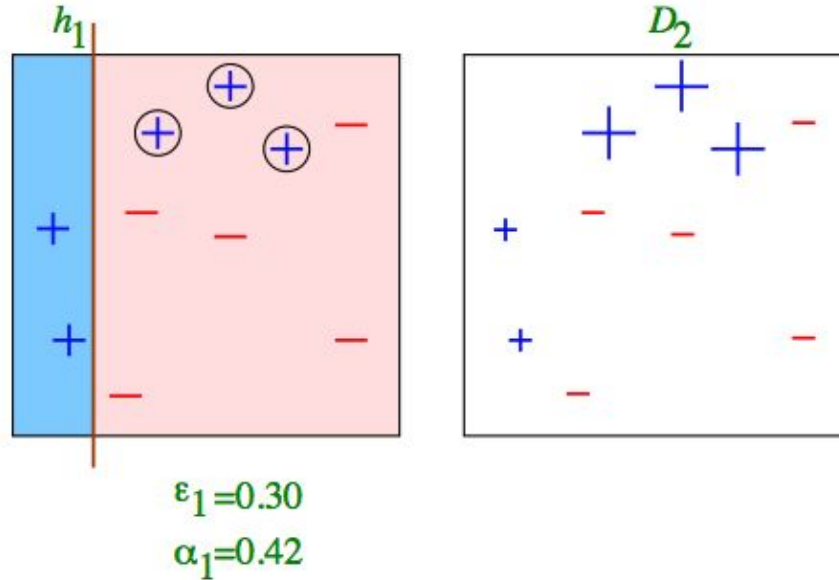




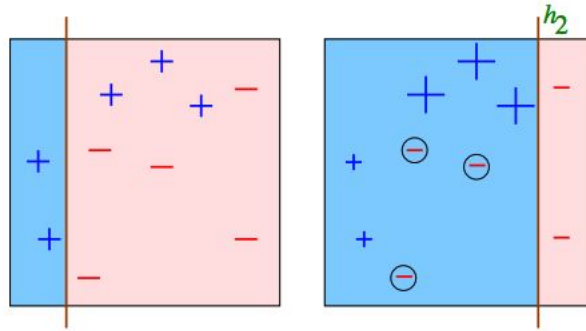
# Toy Example – Round 1

$$D_{t+1}(i) = \frac{D_t(i) \exp(\alpha_t)}{Z_t} \text{ if } y_i \neq h_t(x_i)$$

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t)}{Z_t} \text{ if } y_i = h_t(x_i)$$

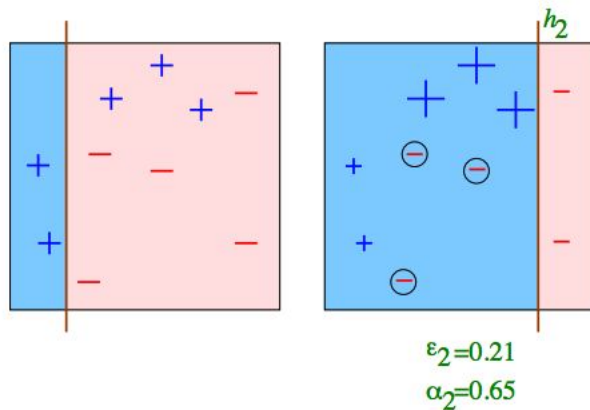


## Toy Example – Round 2



## Toy Example – Round 2

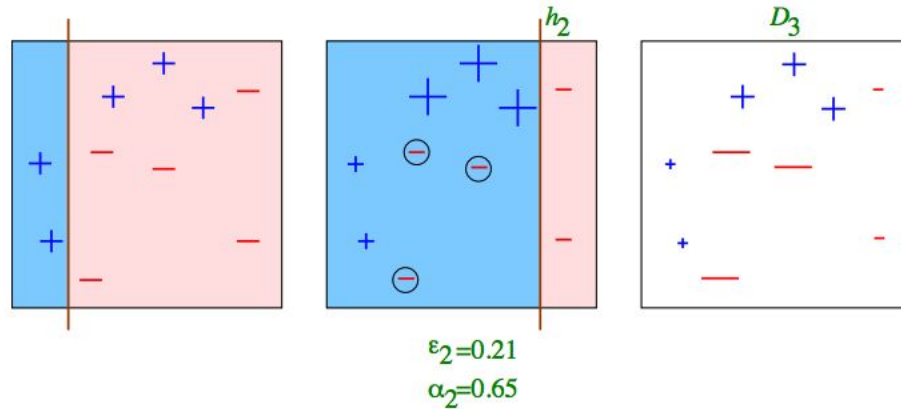
$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$



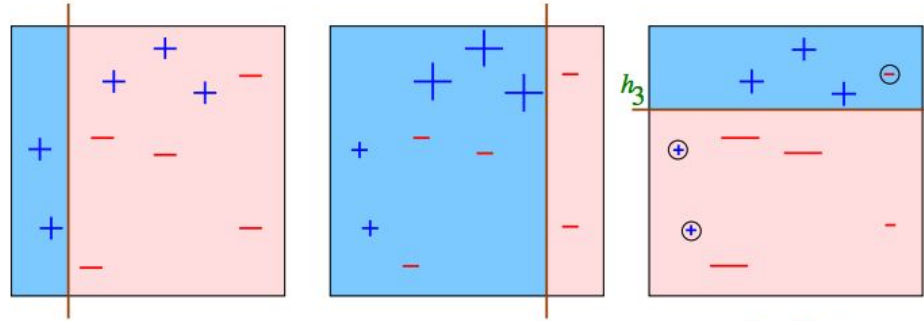
## Toy Example – Round 2

$$D_{t+1}(i) = \frac{D_t(i) \exp(\alpha_t)}{Z_t} \text{ if } y_i \neq h_t(x_i)$$

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t)}{Z_t} \text{ if } y_i = h_t(x_i)$$

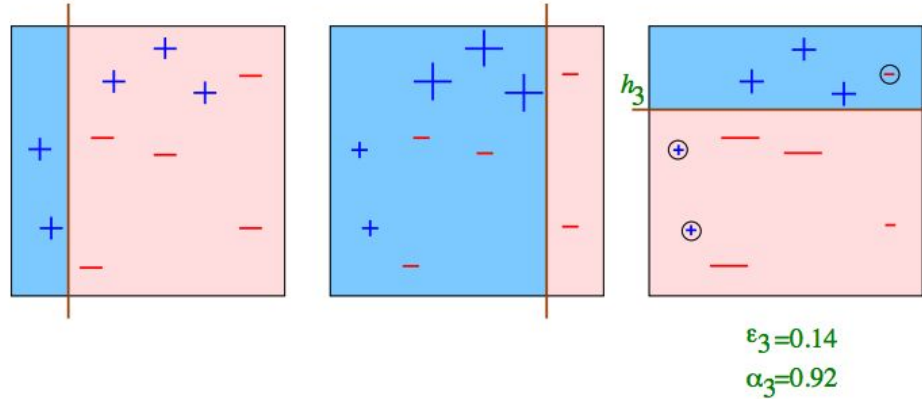


## Toy Example – Round 3



## Toy Example – Round 3

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



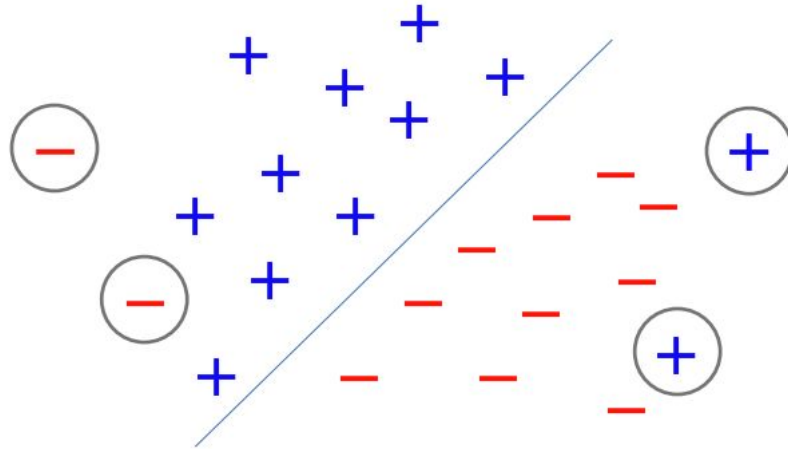
# Toy Example

$$H_{\text{final}} = \text{sign} \left( 0.42 \begin{array}{|c|c|} \hline \text{blue} & \text{red} \\ \hline \end{array} + 0.65 \begin{array}{|c|c|} \hline \text{blue} & \text{blue} \\ \hline \end{array} + 0.92 \begin{array}{|c|c|} \hline \text{blue} & \text{red} \\ \hline \end{array} \right)$$

=


## Effect of Outliers

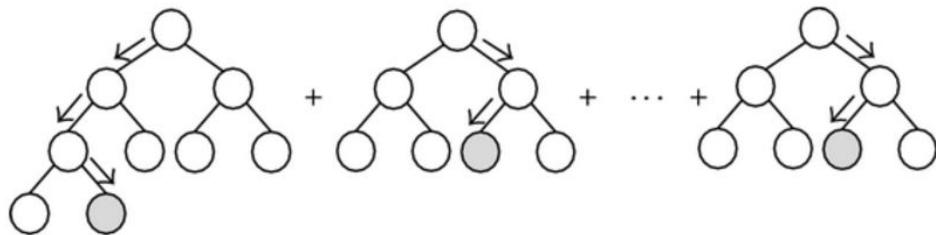
Too many outliers can degrade classification performance dramatically  
increase time to convergence.





# Gradient Boosted Decision Trees

- **Adaboost**: redistribute the weights in the data so that later classifier focuses more on the misclassified examples by previous classifiers.
- **Gradient boosting**:
  - Step 1: Calculate **the residual** for all examples in the training data (the difference between the outcome of the first learner and the real value).
  - Step 2: Build learner to predict/fit **the residual** left from the previous classifiers.
  - These two steps continues until certain threshold is met.



# XGBoost

- XGBoost stands for e**X**treme **G**radient **B**oosting.
- XGBoost is an implementation of gradient boosted decision trees, created by Tianqi Chen (UW).



*The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms. Which is the reason why many people use xgboost.*

- The two advantages of XGBoost:
  - Execution Speed (written in C++).
  - Model Performance (a more regularized model formalization to control over-fitting).

# XGBoost on Kaggle



[KDnuggets Home](#) » [News](#) » [2016](#) » [Mar](#) » [Software](#) » [XGBoost: Implementing the Winningest Kaggle Algorithm in Spark and Flink \( 16:n11 \)](#)

**Latest News, Stories**

- When Will We Bow To Our Machine Overlords?
- The ABC's of Insurance: Analytics, business transfor...
- Mind of a Data Scientist – Part 1
- SAP: Reimagine Predictive Analytics for the Digital En...
- Top Stories, Oct 17-23: Top 10 Data Science Videos on ...

More [News & Stories](#) | [Top Stories](#)

**Visual Statistics**

Your data.  
Powerful predictive modeling.



**XGBoost: Implementing the Winningest Kaggle Algorithm in Spark and Flink**

[Previous post](#)

[f](#) [in](#) [G+](#) [6](#) [Share](#) [71](#)

Tags: [Apache Spark](#), [Distributed System](#)

An overview of XGBoost4J, a JVM-based implementation of the most successful recent machine learning competitions, with distributed support for Spark and Flink.

By Nan Zhu, McGill University, and Tianyi Chen, Microsoft Research, Washington.

**Introduction**

XGBoost is a library designed and optimized for tree boosting. Gradient boosting trees model is originally proposed by Friedman et al. By embracing multi-threads and introducing regularization, XGBoost delivers higher computational power and more accurate prediction. More than half of the winning solutions in machine learning challenges hosted at Kaggle adopt XGBoost (incomplete list). XGBoost has provided native interfaces for C++, R, python, Julia and Java users. It is used by both data exploration and production scenarios to solve real world machine learning problems.

***More than half of the winning solutions in machine learning challenges hosted at Kaggle adopt XGBoost***

# Boosting: Pros and Cons

- Pros:
  - Often best off-the-shelf accuracy on many problems.
  - Using model for prediction requires only modest memory and is fast.
  - Does not require careful normalization of features to perform well.
  - Like decision trees, handles a mixture of feature types.
- Cons:
  - The models are often difficult for humans to interpret.
  - Requires careful tuning of the learning rate and other parameters.
  - Not easy to parallel (unlike random forest) since each classifier can only be trained after the previous one has been trained.

# Bagging vs. Boosting

- Bagging
  - Resamples data points
  - Weight of each classifier is the same
  - Only variance reduction
- Boosting
  - Reweights data points
  - Weight is dependent on classifier accuracy.
  - Both bias and variance reduced.



# Lab

