Machine Learning

AIML CZG565

Ensemble Learning

BITS Pilani

Pilani Campus

Course Faculty of M.Tech Cluster

BITS – CSIS - WILP

Machine Learning

Disclaimer and Acknowledgement

- These content of modules & context under topics are planned by the course owner Dr. Sugata, with grateful acknowledgement to many others who made their course materials freely available online
- We here by acknowledge all the contributors for their material and inputs.
- We have provided source information wherever necessary
- Students are requested to refer to the textbook w.r.t detailed content of the presentation deck shared over canvas
- We have reduced the slides from canvas and modified the content flow to suit the requirements of the course and for ease of class presentation

Slide Source / Preparation / Review:
From BITS Pilani WILP: Profs.Sugata, Chetana, Monali, Rajavadhana, Seetha, Anita

External: CS109 and CS229 Stanford lecture notes, Dr.Andrew NG and many others who made their course materials freely available online

Course Plan

M1	Introduction & Mathematical Preliminaries
M2	Machine Learning Workflow
M3	Linear Models for Regression
M4	Linear Models for Classification
M5	Decision Tree
M6	Instance Based Learning
M7	Support Vector Machine
M8	Bayesian Learning
M9	Ensemble Learning
M10	Unsupervised Learning
M11	Machine Learning Model Evaluation/Comparison

Ensemble Learning

Ensemble Philosophy

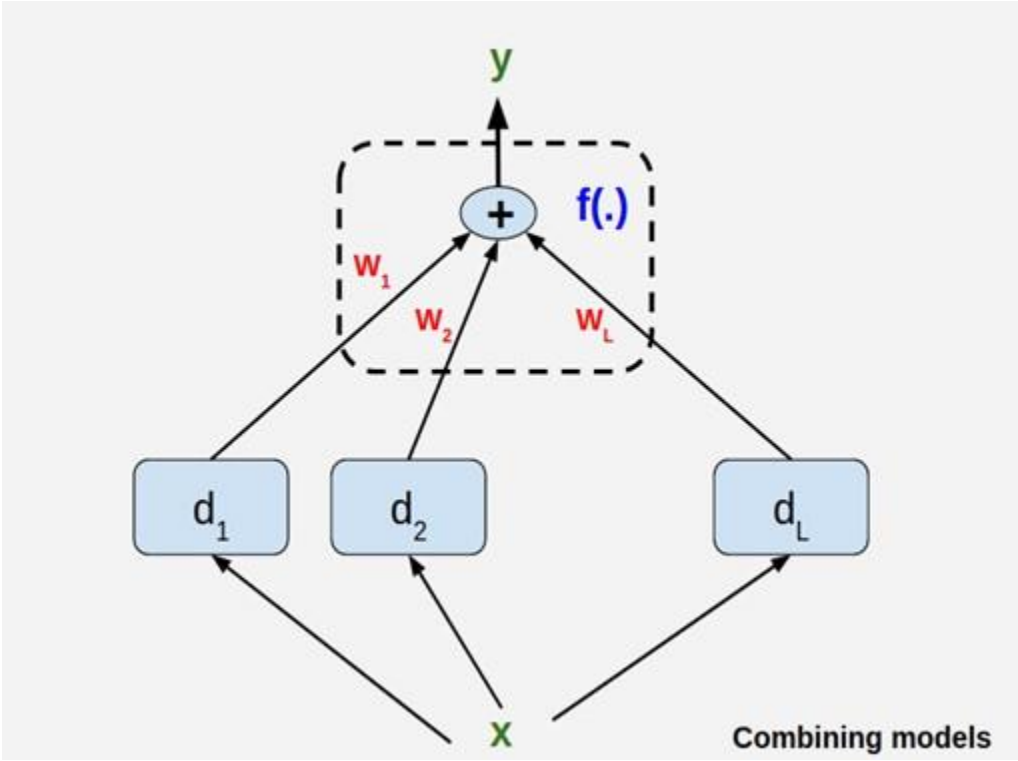
- No Free Lunch Theorem: There is no algorithm that is always the most accurate

- Each learning algorithm dictates a certain model that comes with a set of assumptions
  - Each algorithm converges to a different solution and fails under different circumstances
    - The best tuned learners could miss some examples and there could be other learners which works better on (may be only) those !
  - In the absence of a single expert (a superior model) , a committee (combinations of models) can do better !
    - A committee can work in many ways ...

Weak learner does only slightly better than random guessing

Committee of Models

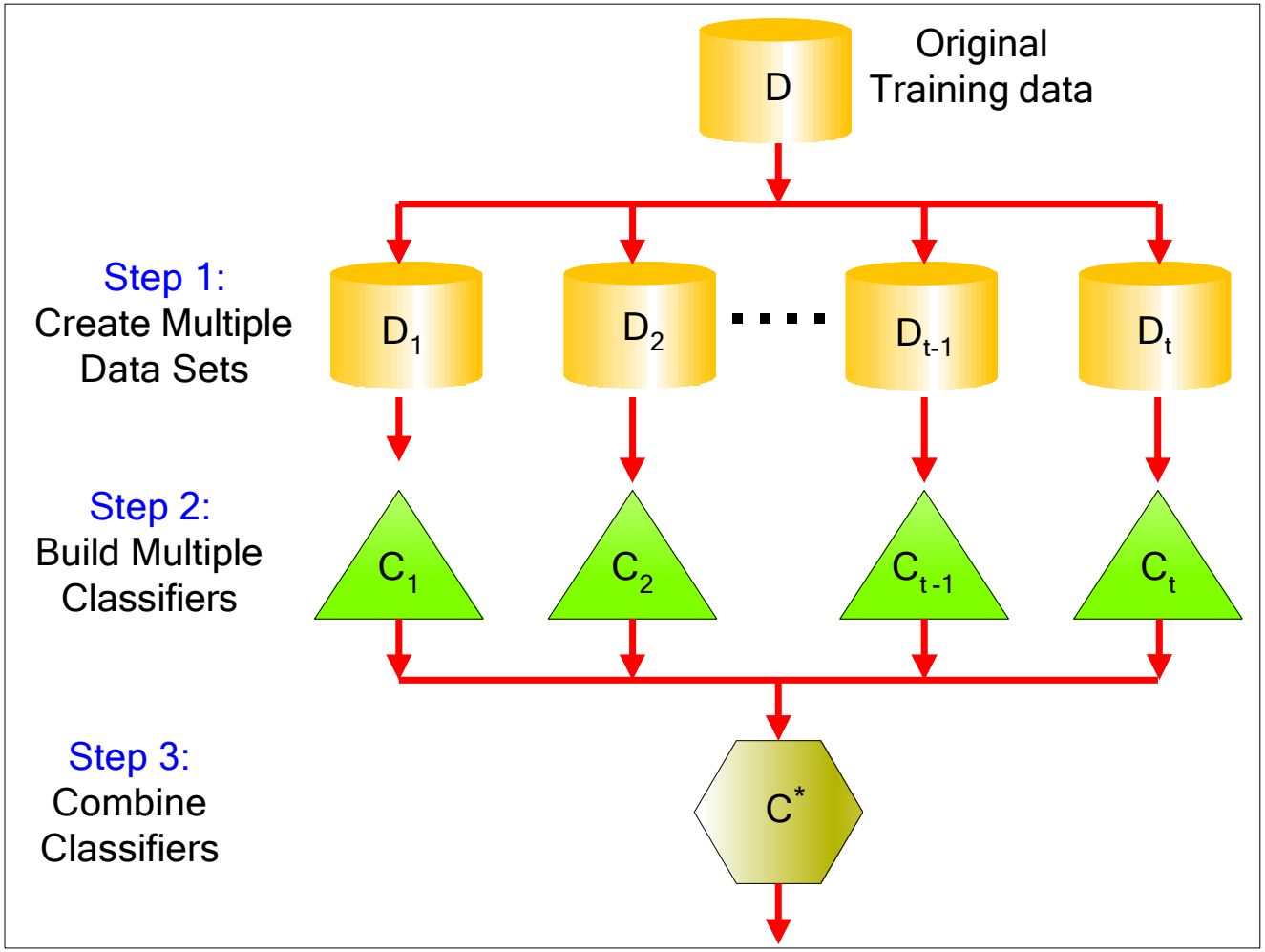
- Committee Members are base learners !
- Major challenges dealing with this committee
  - Expertise of each of the members (Does it help / not?)
  - Combining the results from the members for better performance



Ensemble Methods

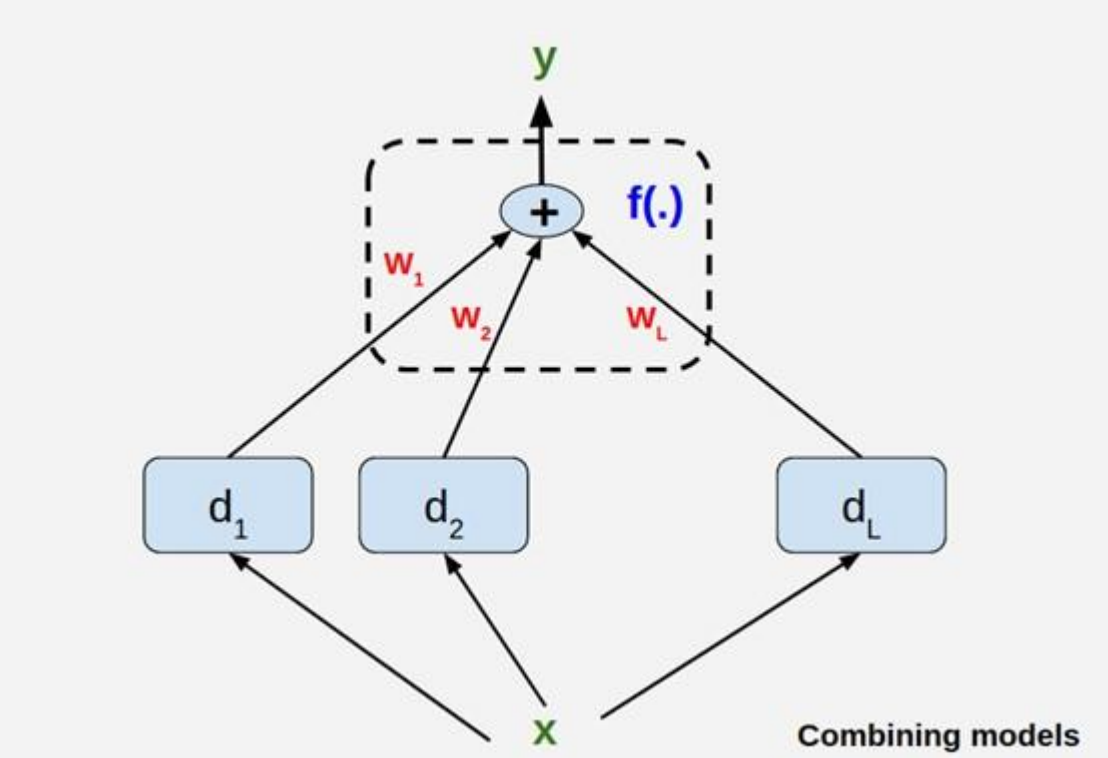
- Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone
- Construct a set of classifiers from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers
- Tend to reduce problems related to over-fitting of the training data.

General Approach



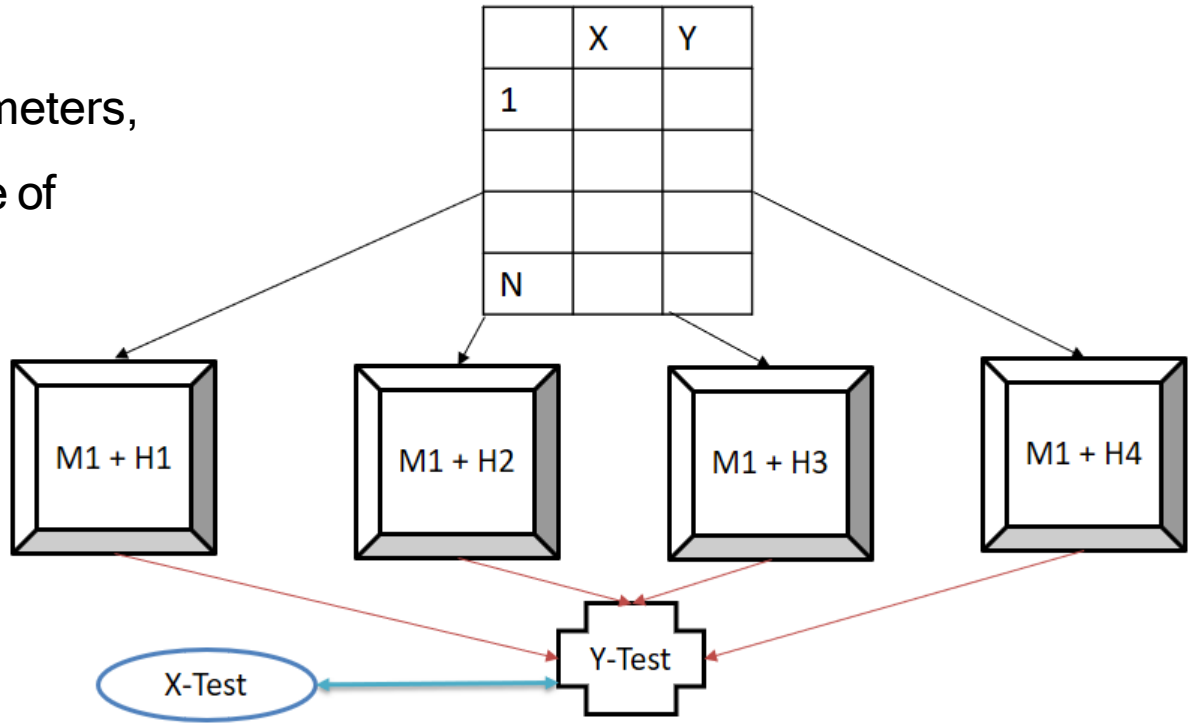
Issue 1 : On the members (Base Learners)

- It does not help if all learners are good/bad at roughly same thing
  - Need Diverse Learners



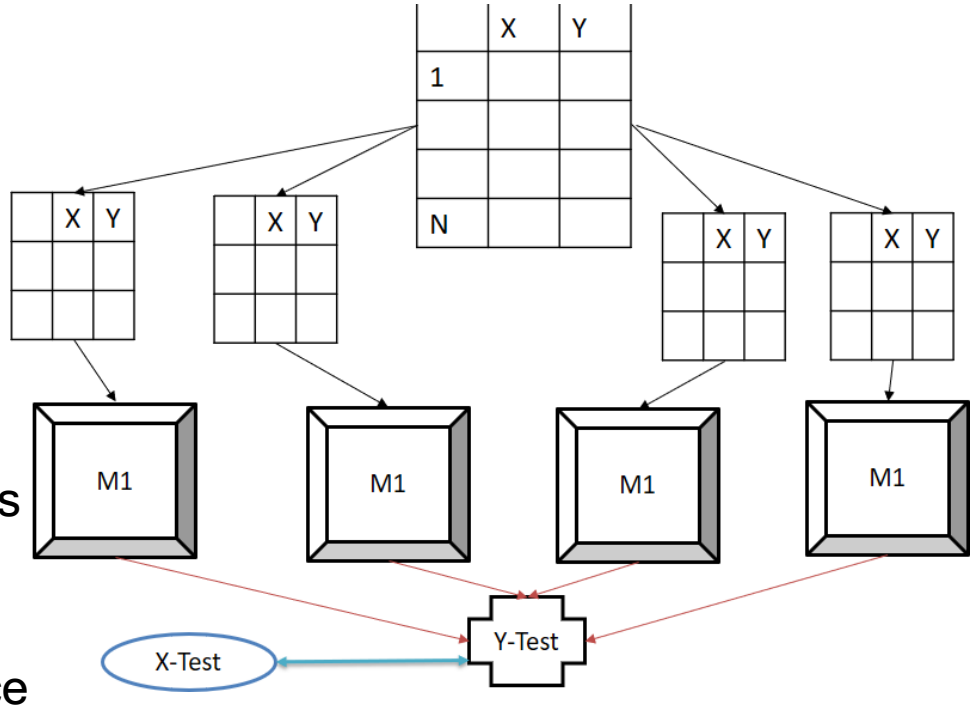
Issue 1 : On the members (Base Learners)

- Use Different Algorithms
  - Different algorithms make different assumptions
- Use Different Hyper parameters,
  - E.g. vary the structure of neural nets



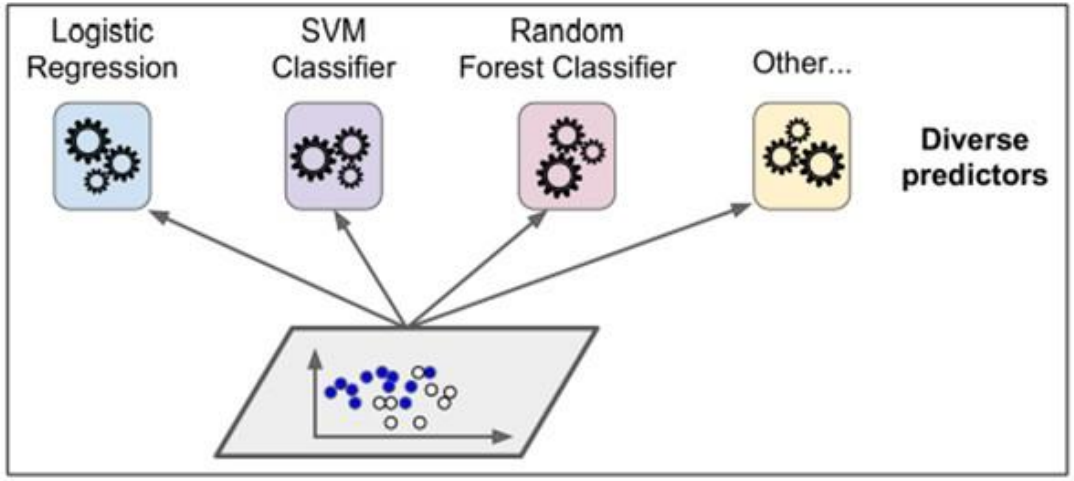
Issue 1 : On the members (Base Learners)

- Different input representations
  - Uttered words + video information of speakers clips
  - image + text annotations
- Different training sets
  - Draw different random samples of data
  - Partition data in the input space and have learners specialized in those spaces (mixture of experts)



Issue -2 : Combining Results Base Learners

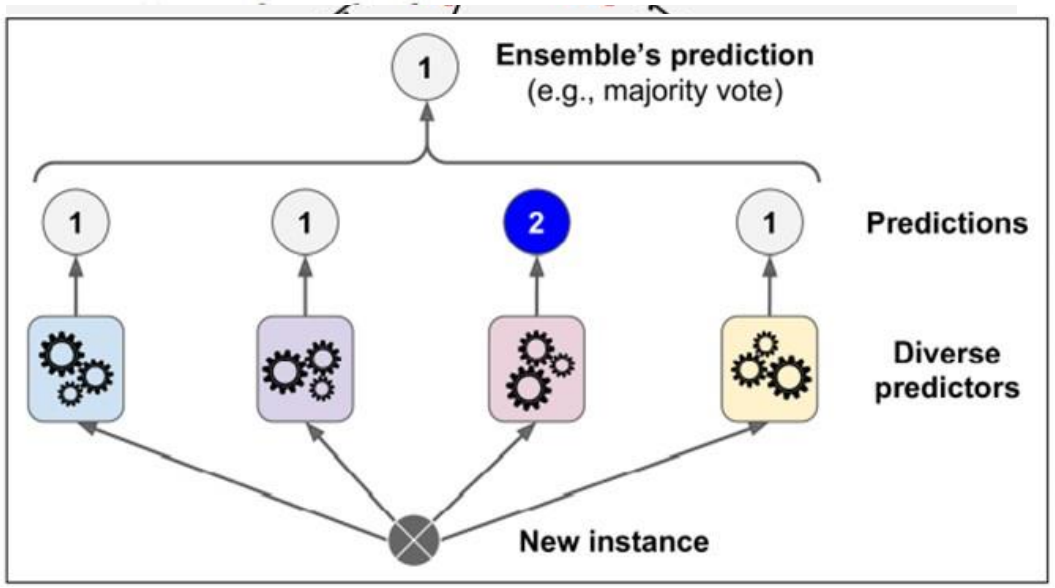
$$y = f(d_1, d_2, \ldots, d_L | \Phi)$$



A Simple Combination Scheme:

$$y = \sum_{j=1}^{L} w_j d_j$$

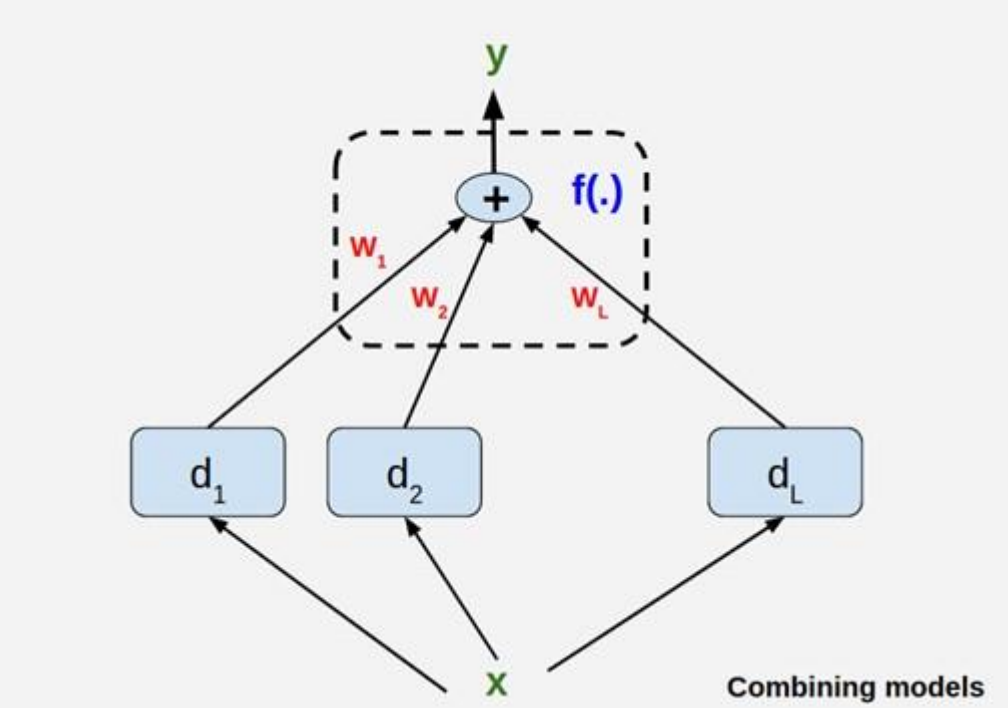
$$w_j \geq 0 \text{ and } \sum_{j=1}^{L} w_j = 1$$



Issue -2 : Combining Results Base Learners

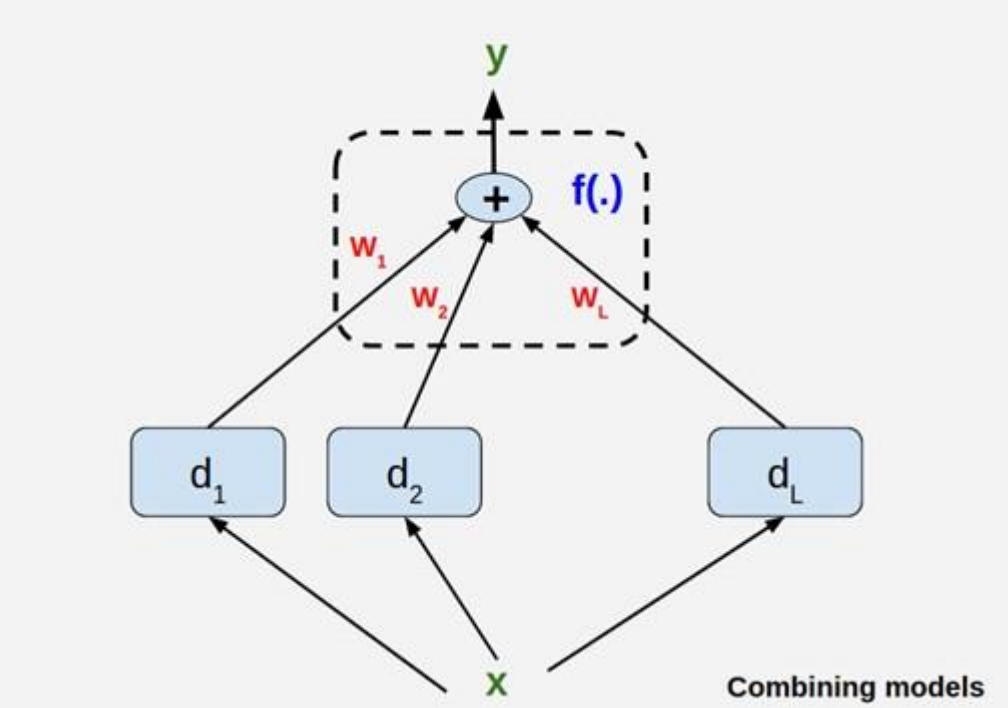
$$y = f(d_1, d_2, \ldots, d_L | \Phi)$$

Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$



Issue -2 : Combining Results Base Learners

	C_1	C_2	C_3
d_1	0.2	0.5	0.3
d_2	0.0	0.6	0.4
d_3	0.4	0.4	0.2
~~Sum~~ **Avg**	0.2	**0.5**	0.3
Median	0.2	**0.5**	0.4
Minimum	0.0	**0.4**	0.2
Maximum	0.4	**0.6**	0.4
Product	0.0	**0.12**	0.032

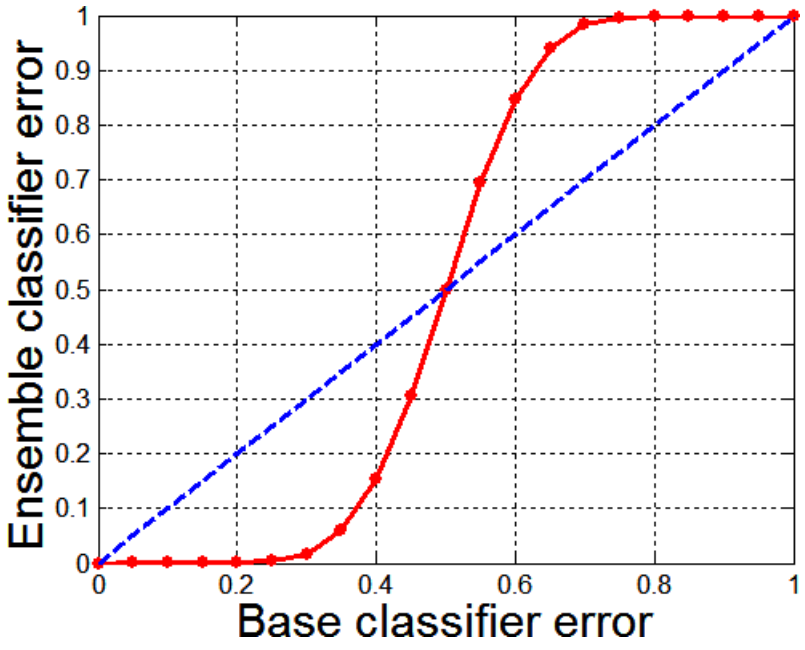


When Ensemble Methods Work?

- Ensemble classifier performs better than the base classifiers when e is smaller than 0.5
- Necessary conditions for an ensemble classifier to perform better than a single classifier:
  - Base classifiers should be independent of each other
  - Base classifiers should do better than a classifier that performs random guessing

Necessary Conditions for Ensemble Methods

- Ensemble Methods work better than a single base classifier if:
  - All base classifiers are independent of each other
  - All base classifiers perform better than random guessing (error rate < 0.5 for binary classification)



Classification error for an ensemble of 25 base classifiers, assuming their errors are uncorrelated.

Why Ensemble Methods work?

- 25 base classifiers. Each classifier has error rate, $\varepsilon = 0.35$
- If base classifiers are identical, then the ensemble will misclassify the same examples predicted incorrectly by the base classifiers depicted by dotted line
- Assume errors made by classifiers are uncorrelated
- Ensemble makes a wrong prediction only if base classifiers error is more than 0.5

Classification error for an ensemble of 25 base classifiers, assuming their errors are uncorrelated.

Types of Ensemble Methods

- By manipulating training set
  - Example: bagging, boosting, random forests

- By manipulating input features
  - Example: random forests

Bagging

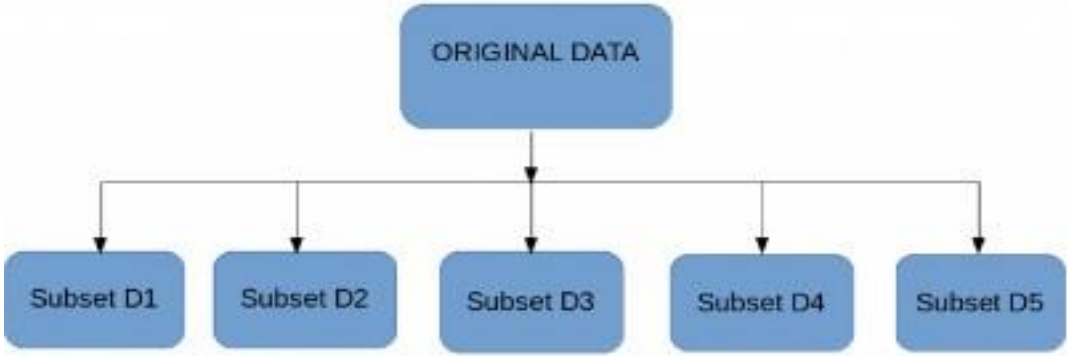
Bootstrap Aggregating

Bagging (Bootstrap Aggregating)

- Technique uses these subsets (bags) to get a fair idea of the distribution (complete set).
- The size of subsets created for bagging may be less than the original set.
- Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, with replacement.
- When you sample with replacement, items are independent. One item does not affect the outcome of the other. You have 1/7 chance of choosing the first item and a 1/7 chance of choosing the second item.
- If the two items are dependent, or linked to each other. When you choose the first item, you have a 1/7 probability of picking a item. Assuming you don't replace the item, you only have six items to pick from. That gives you a 1/6 chance of choosing a second item.

Bagging (Bootstrap Aggregating)

- Multiple subsets are created from the original dataset, selecting observations with replacement.
- A base model (weak model) is created on each of these subsets.
- The models run in parallel and are independent of each other.
- The final predictions are determined by combining the predictions from all the models.



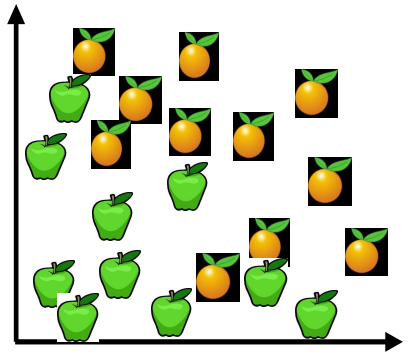
Bagging at training time

For m=1:M, Obtain bootstrap sample D_m from the training data D

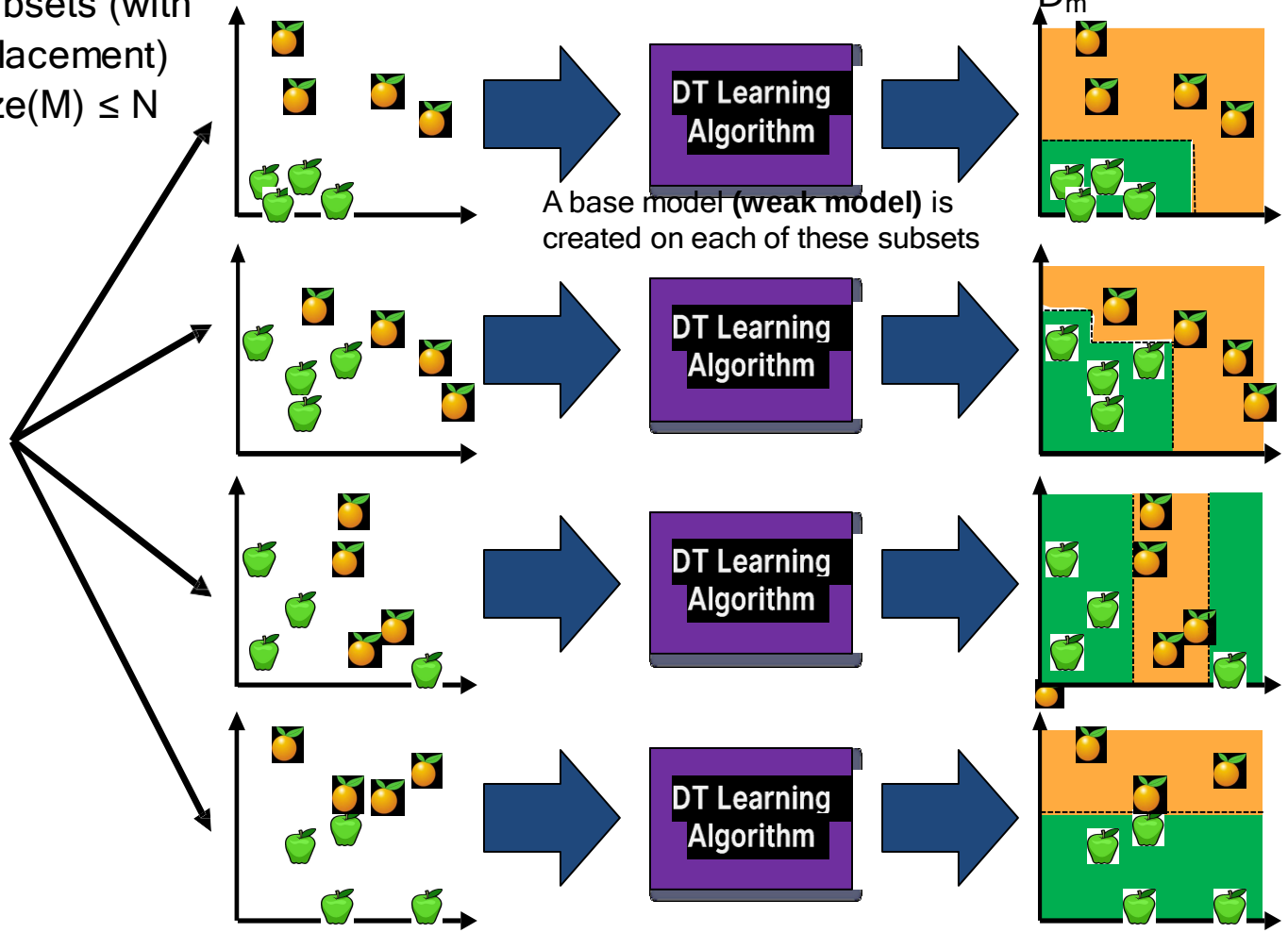
M subsets (with replacement) Size(M) ≤ N

Build a model $G_m(x)$ from bootstrap data D_m

Training set



$D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

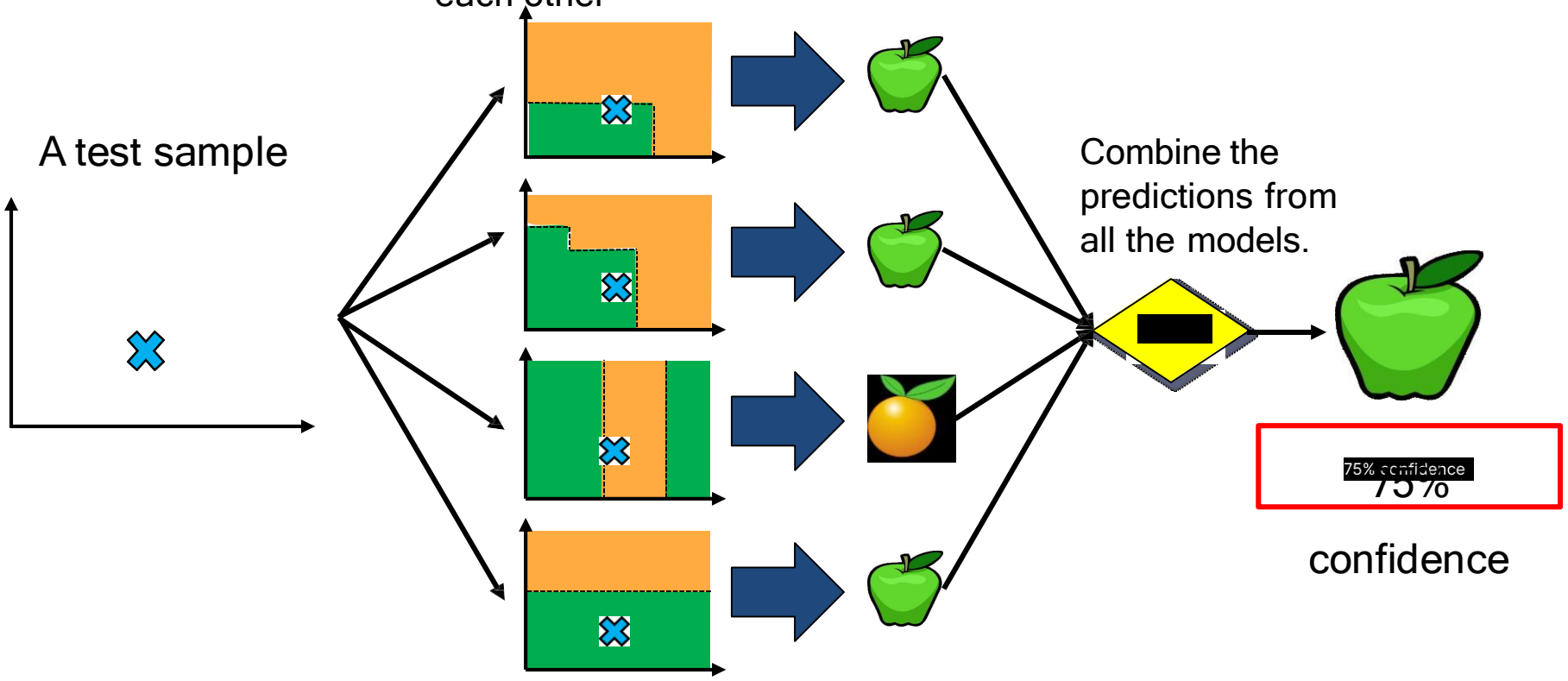


A base model **(weak model)** is created on each of these subsets

Bagging at inference time

The models run in parallel and are independent of each other

A test sample



Combine the predictions from all the models.

75% confidence

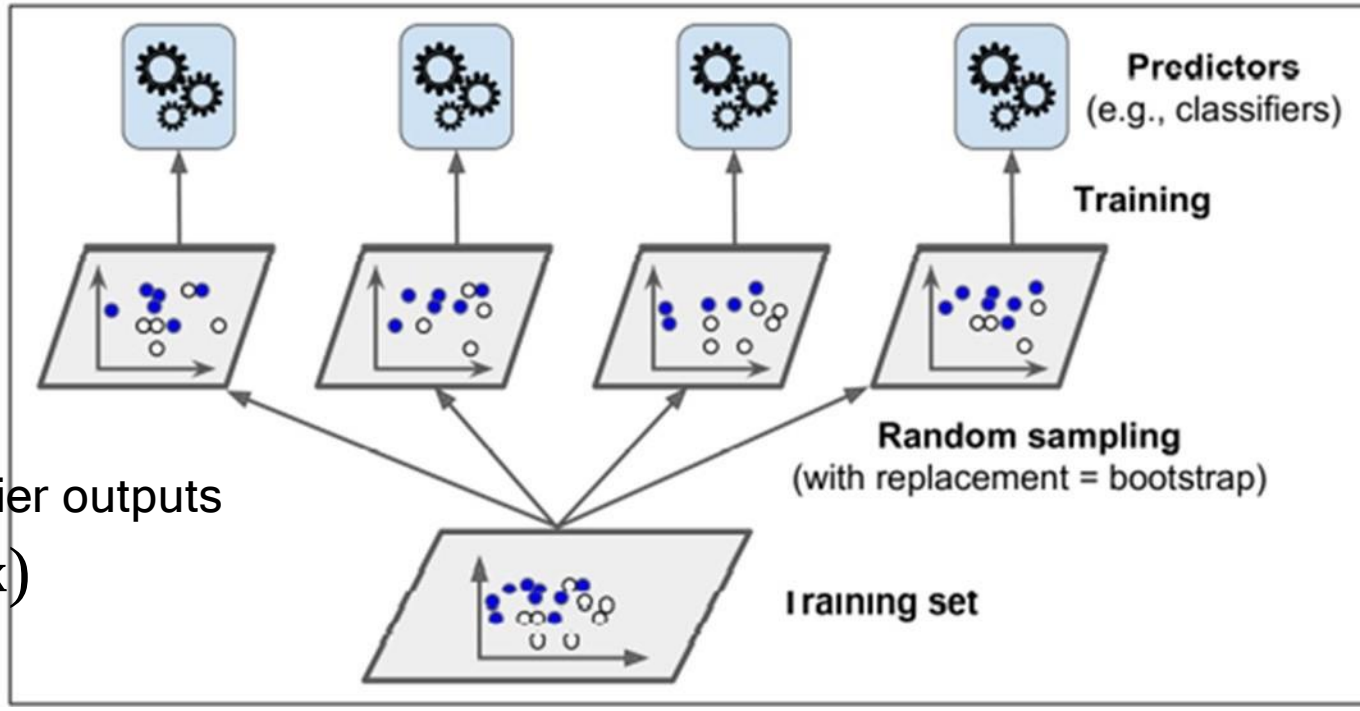
The Bagging Model

- Regression

$$y = \frac{1}{M}\sum_{m=1}^{M} G_m(\mathbf{x})$$

- Classification:
  - Vote over classifier outputs

$$G_1(\mathbf{x}),\ldots,G_M(\mathbf{x})$$

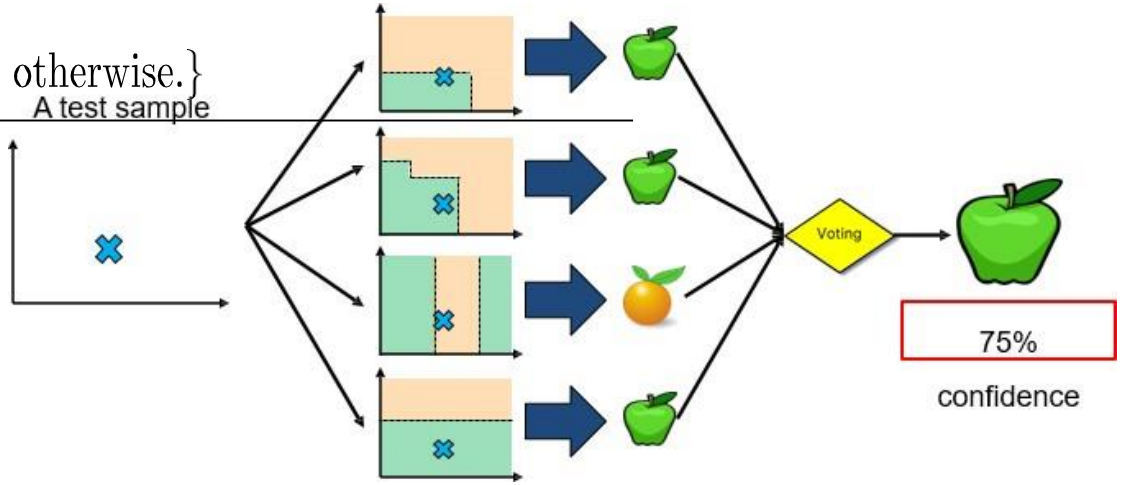


Bagging Algorithm

Algorithm

1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size N.
4: Train a base classifier C_i on the bootstrap sample.
5: **end for**
6: $C^*(x) = \underset{y}{\operatorname{argmax}} \sum_i \delta\big(C_i(x) = y\big)$.

$\{\delta(\cdot) = 1$ if its argument is true and 0 otherwise.$\}$



Bagging Example

Consider 1-dimensional data set:

Original Data:

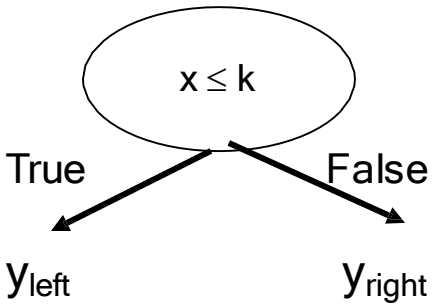
x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Classifier is a decision stump

Decision rule: $x \leq k$ versus $x > k$

$x < 0.35$ or $X >= 0.75$.

Split point k is chosen based on entropy



Decision tree with one internal node (the root) which is immediately connected to the terminal nodes (its leaves). **Decision stump** makes a prediction based on the value of just a single input feature. Sometimes they are also called 1-rules

Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

X < = 0.7 □ y= 1
X > 0.7 □ y= 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging Example

Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging Example

- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

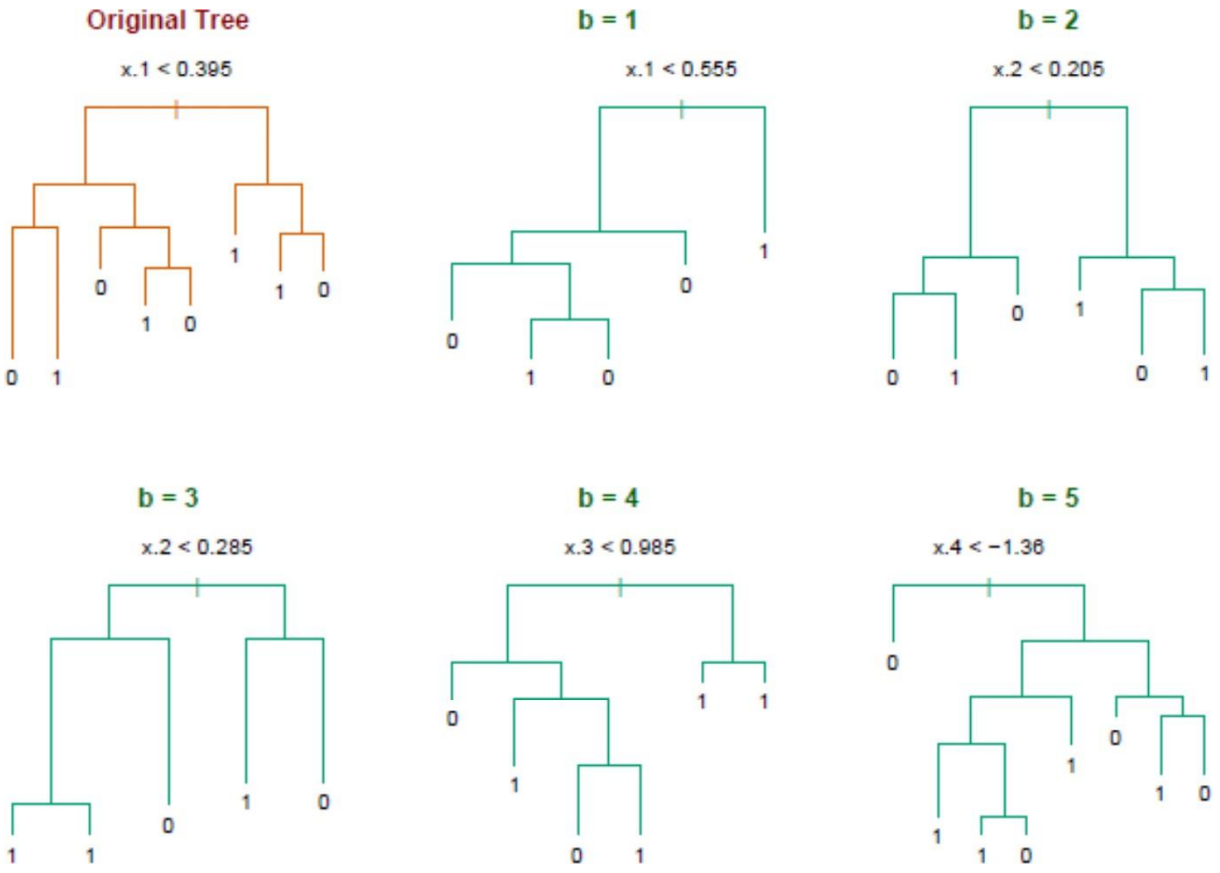
Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Bagging as Decision Tree



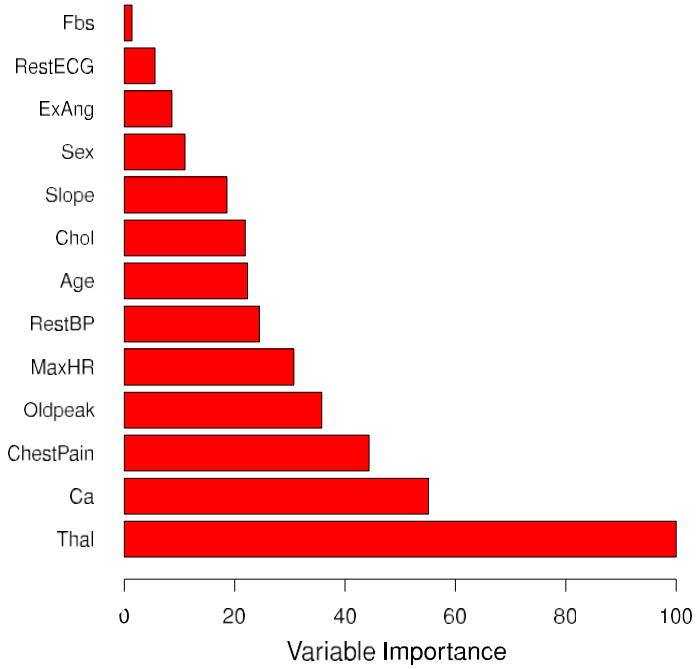
Source : Hastie et al.,"The Elements of Statistical Learning: Data Mining, Inference, and Prediction", Springer (2009

Bagging - Sampling Process

- No cross validation?
- Remember, in bootstrapping we sample with replacement, and therefore **not all observations are used for each bootstrap sample. Its observed that on average 1/3 observation are not used!**
- We call them out-of-bag samples (OOB)
- We can predict the response for the i-th observation using each of the trees in which that observation was OOB and do this for n observations
- Calculate overall OOB MSE or classification error

Bagging - Variable Importance

- Bagging results in improved accuracy over prediction using a single tree
- Unfortunately, difficult to interpret the resulting model. **Bagging improves prediction accuracy at the expense of interpretability.**
- Calculate the total amount that the RSS or entropy is decreased due to splits over a given predictor, averaged over all trees.
- A visualize of the values of every feature in a sample use case is shown here.



Bagging – Effect on Bias

- If a base classifier is stable, i.e., robust to minor perturbations in the training set, then the error of the ensemble is primarily caused by bias in the base classifier.
- In this situation, bagging may not be able to improve the performance of the base classifiers significantly.
- It may even degrade the classifier's performance because the effective size of each training set is about 37% smaller than the original data.

Additional Reading Material

Source Credit : Sebastian Raschka

Why Majority Voting Works - Proof

- Assume *n* independent classifiers with a base error rate ϵ. Here, independent means that the errors are uncorrelated
- Assume a binary classification task
- Assume the error rate is better than random guessing (i.e., lower than 0.5 for binary classification)

$$\forall \epsilon i \in \{\epsilon 1, \epsilon 2, \ldots, \epsilon n\}, \epsilon i < 0.5$$

- The probability that we make a wrong prediction via the ensemble if k classifiers predict the same class label is given by below PMF of binomial distribution.

$$P(k) = \binom{n}{k} \epsilon^k (1-\epsilon)^{n-} \qquad k > \lceil n/2 \rceil$$

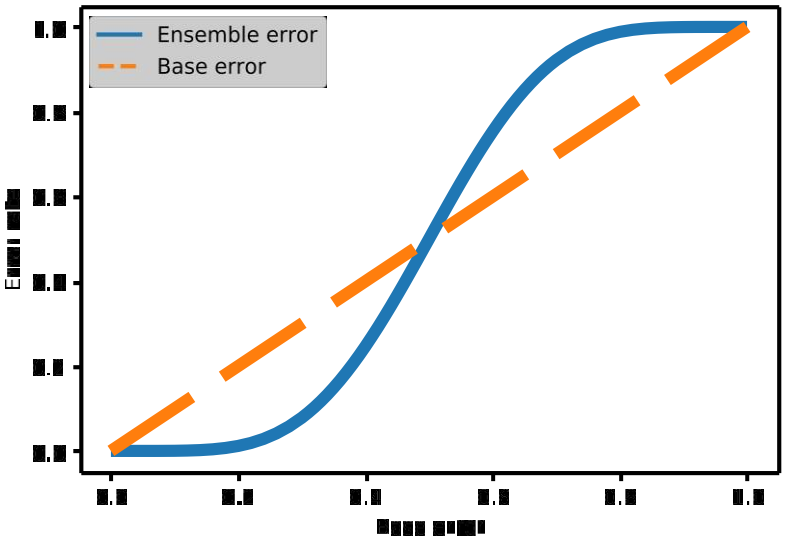
Cont….

- The probability that we make a wrong prediction via the ensemble if k classifiers predict the same class label is given by below PMF of binomial distribution.

$$\mathbf{P(k)} = \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k} \qquad k > \lceil n/2 \rceil$$

- Ensemble error: $\epsilon_{ens} = \sigma_k^n \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k}$
- Eg., If we consider 11 classifier's result then atleast 6 classifier must have produced same label ie.,

$$\epsilon_{ens} = \sigma_{k=6}^{11} \binom{11}{6} 0.25^k (1-0.25)^{11-k} = 0.034$$



Bagging – Advantages

- Reduces overfitting (variance)
- Normally uses one type of classifier
- Decision trees are popular
- Easy to parallelize

Bagging – Limitation

- Each tree is identically distributed (i.d.)
- The expectation of the average of B such trees is the same as the expectation of any one of them
- The bias of bagged trees is the same as that of the individual trees
- Results in a model that is i.d. and not i.i.d

Bagging – Limitation (Cont…)

An average of *B* i.i.d. random variables, each with variance σ^2, has variance: $\boldsymbol{\sigma^2/B}$

If i.d. (identical but not independent) and pair correlation r is present, then the variance is:

$$\rho\,\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

As *B* increases the second term disappears but the first term remains

Why does bagging generate correlated trees?

Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors.

Then all bagged trees will select the strong predictor at the top of the tree and therefore all trees will look similar.

How do we avoid this?

Restriction on the choice of predictors and no.of.trees that can use the same predictor may help. But this leads to too much variance

Bagging – Limitation (Cont…)

Remember we want i.i.d such as the bias to be the same and variance to be less? Other ideas?

What if we consider only a subset of the predictors at each split?

We will still get correlated trees unless …. we **randomly** select the subset !

Algorithms based on Bagging and Boosting

Bagging algorithms:

- Random forest

Boosting algorithms:

- AdaBoost
- Gradient Boosting

Random Forest

Random Forest

- Ensemble method specifically designed for decision tree classifiers
- Random Forests grows many trees
  - Ensemble of unpruned decision trees
  - Each base classifier classifies a "new" vector of attributes from the original data
  - Final result on classifying a new instance: voting.
  - Forest chooses the classification result having the most votes (over all the trees in the forest)

Lower correlation across trees

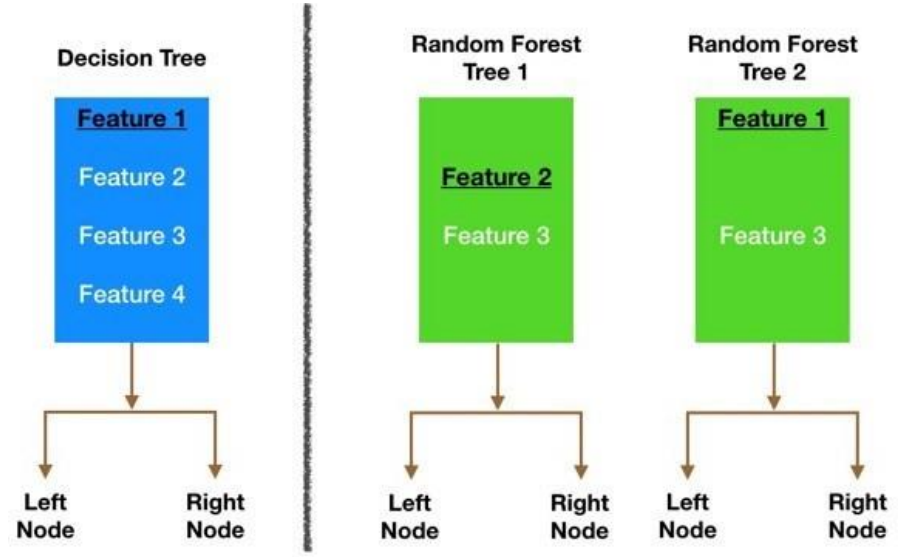
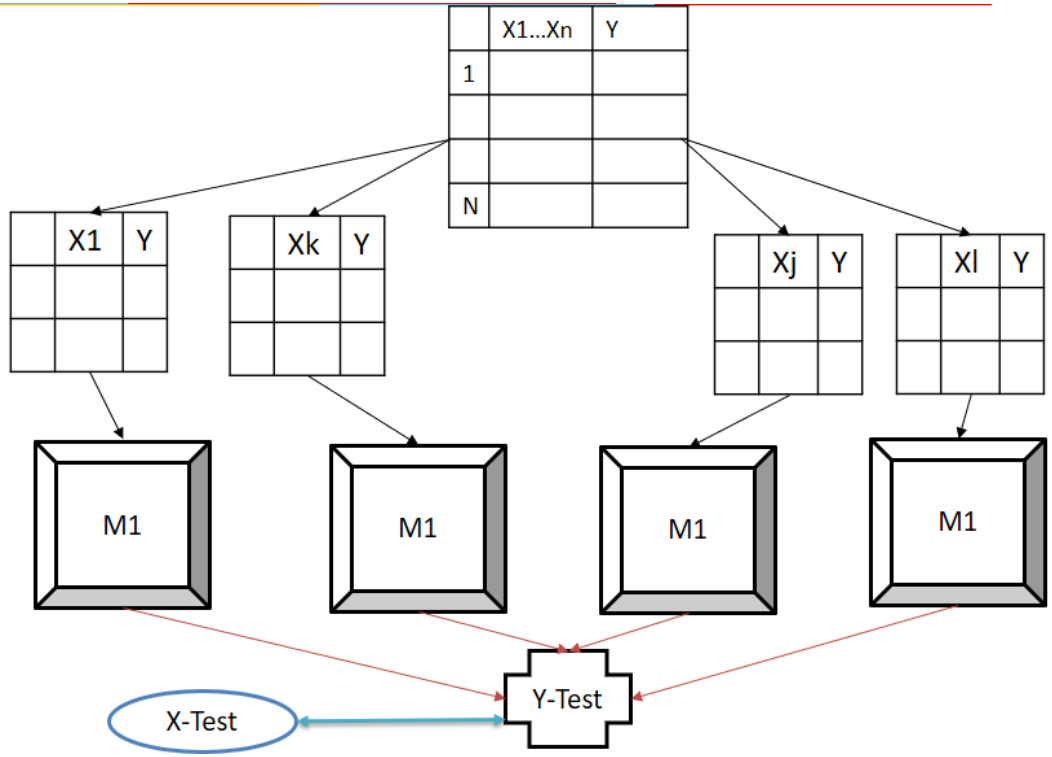


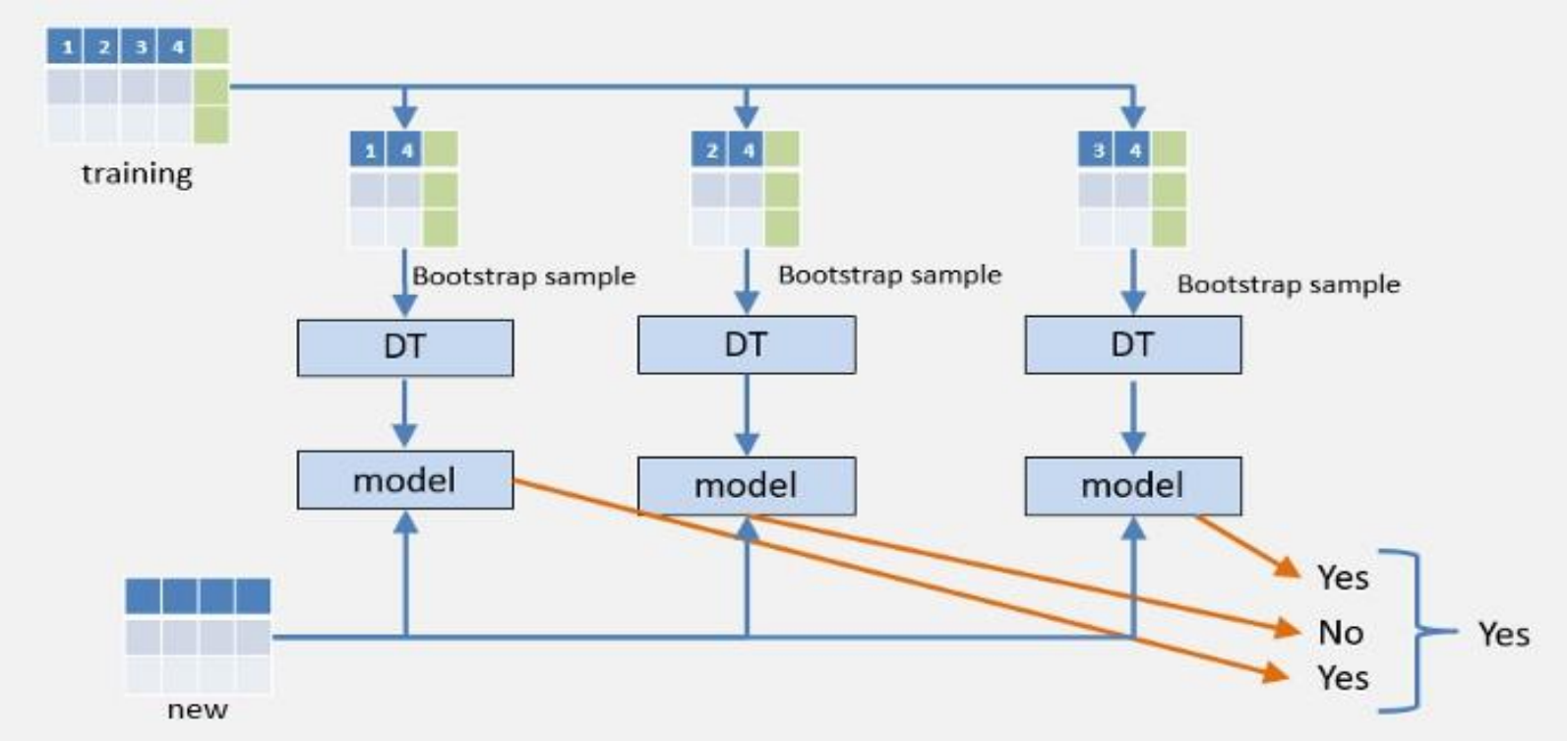
Image credit: https://medium.com

Random Forest Algorithm

- Construct an ensemble of decision trees by manipulating training set as well as features
  - Use bootstrap sample to train every decision tree (similar to Bagging)
  - Use the following tree induction algorithm:
    - At every internal node of decision tree, randomly sample p attributes for selecting split criterion
    - Repeat this procedure until all leaves are pure (unpruned tree)



Random Forest



- Trees that are trained on different sets of data (bagging)
- Trees use different features to make decisions.

Image credit: https://medium.com

Random Forest – Summary

- Random Forest is ensemble machine learning algorithm that follows the bagging technique.
- The base estimators in random forest are decision trees.
- Random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.
- Random subsets are created from the original dataset (bootstrapping).
- At each node in the decision tree, only a random set of features are considered to decide the best split.
- A decision tree model is fitted on each of the subsets.
- The final prediction is calculated by averaging the predictions from all decision trees.

Random Forest

- Random Forest need features that have at least some predictive power.
- The trees of the forest and more importantly their predictions need to be uncorrelated (or at least have low correlations with each other).
- Algorithm can solve both type of problems i.e. classification and regression
- Power to handle large data set with higher dimensionality.
- It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods.

Random Forest

Cheaper Feature Selection

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
>>> rnd_clf.fit(iris["data"], iris["target"])
>>> for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
...     print(name, score)
...
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

Additional Reading Material

Random forests are popular. Leo Breiman's and Adele Cutler maintains a random forest website where the software is freely available, and of course it is included in every ML/STAT package

http://www.stat.berkeley.edu/~breiman/RandomForests/

Source Credit : Original Paper : https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

BITS Pilani

Random Forest – Algorithm

For b = 1 to B:

(a) Draw a bootstrap sample $Z*$ of size N from the training data.

(b) Grow a random-forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $\boldsymbol{n_{min}}$ is reached.

i. Select $\boldsymbol{m}$ variables at random from the p variables.

ii. Pick the best variable/split-point among the m.

iii. Split the node into two child nodes.

Output the ensemble of trees.

To make a prediction at a new point x we do:

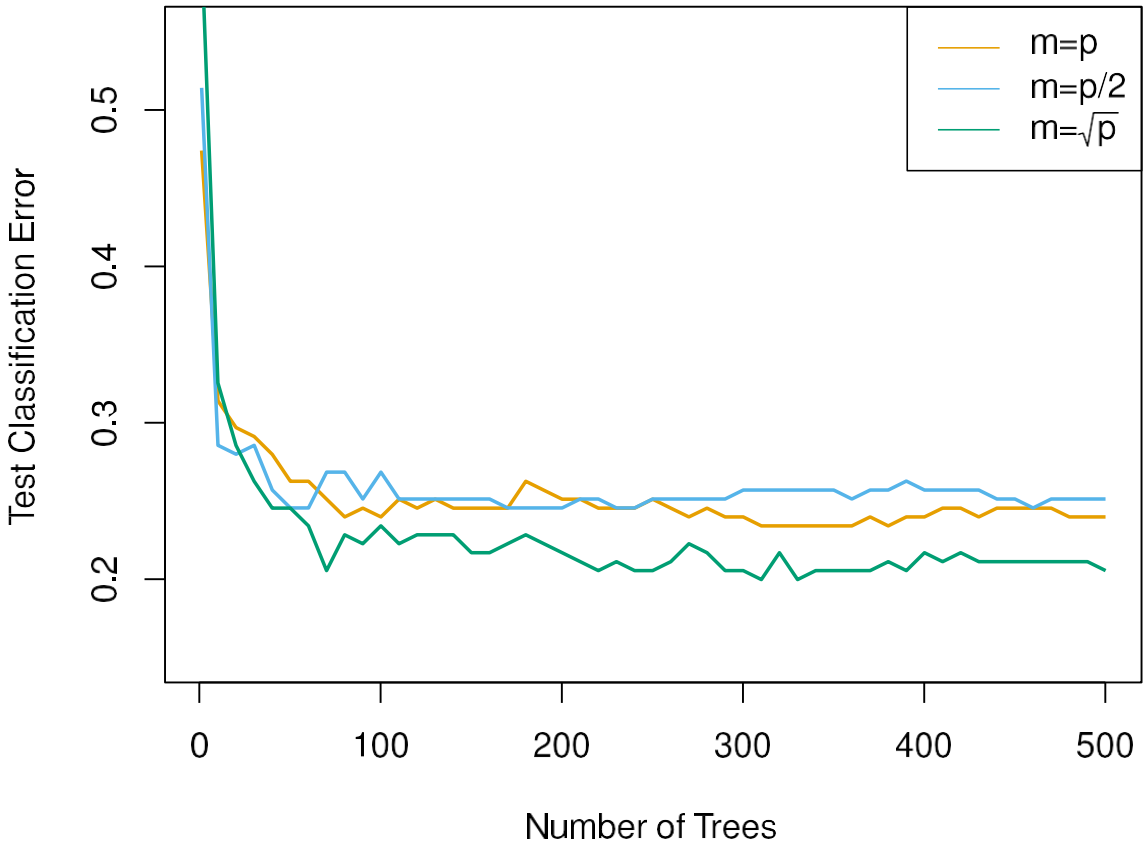
- For regression: average the results
- For classification: majority vote

Random Forest – Algorithm

The inventors make the following recommendations:

- For classification, the default value for m is $\sqrt{p}$ and the minimum node size is one. If m=p then its bagging.
- For regression, the default value for m is p/3 and the minimum node size is five.
- In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.
- Like with Bagging, we can use OOB and therefore RF can be fit in one sequence, with cross-validation being performed along the way. Once the OOB error stabilizes, the training can be terminated.

Random Forest – Algorithm



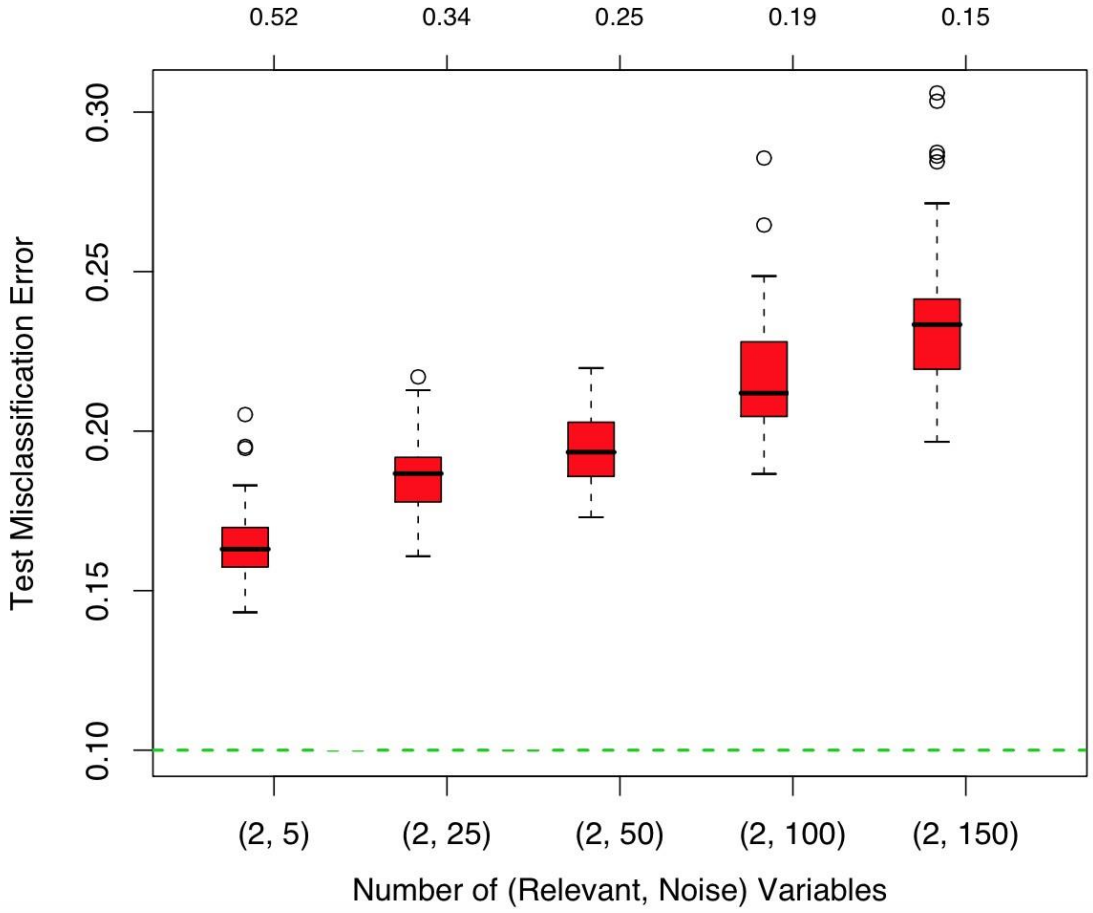
Random Forest – Advantages

- Algorithm can solve both type of problems i.e. classification and regression
- Power to handle large data set with higher dimensionality.
- It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods.
- Random forests "cannot overfit" the data w.r.t to number of trees, since the more number of trees, *B* does not mean there is a increase in the flexibility of the model
- Model outputs **Importance of variable**, which can be a very handy feature (on some random data set).
  - Record the prediction accuracy on the OOB samples for each tree
  - Randomly permute the data for column **j** in the OOB samples and the record the accuracy again.
  - The decrease in accuracy as a result of this permuting is averaged over all trees, and is used as a measure of the importance of variable **j** in the random forest.

Random Forest – Disadvantages

- May over-fit data sets that are particularly noisy.
- Random Forest can feel like a black box approach for statistical modelers – you have very little control on what the model does. You can at best – try different parameters and random seeds!
- When the number of variables is large, but the fraction of relevant variables is small, random forests are likely to perform poorly when m is small. Because at each split the chance can be small that the relevant variables will be selected
- For example, with 3 relevant and 100 not so relevant variables the probability of any of the relevant variables being selected at any split is ~0.25

Random Forest – Disadvantages



Boosting

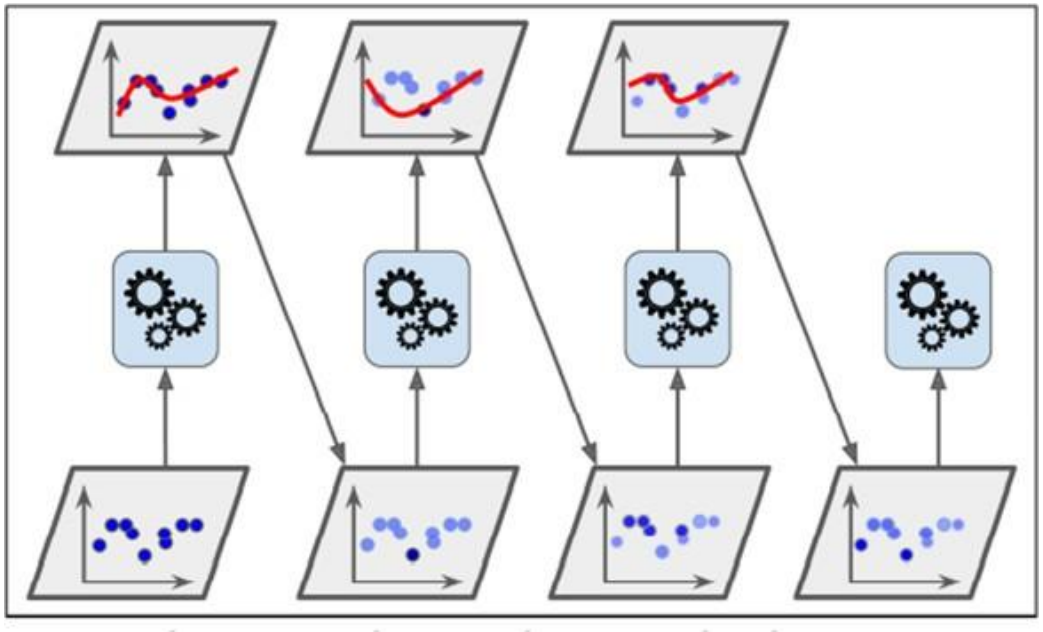
Boosting

- What if a data point is incorrectly predicted by the first model, and then the next (probably all models), will combining the predictions provide better results? Such situations are taken care of by boosting.
- Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model.
- The succeeding models are dependent on the previous model.

Boosting

Train predictors sequentially, each trying to correct its predecessor
Scalability

Ada Boost



An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records

Initially, all N records are assigned equal weights (for being selected for training)

Unlike bagging, weights may change at the end of each boosting round

Boosting

- Records that are wrongly classified will have their weights increased in the next round
- Records that are classified correctly will have their weights decreased in the next round

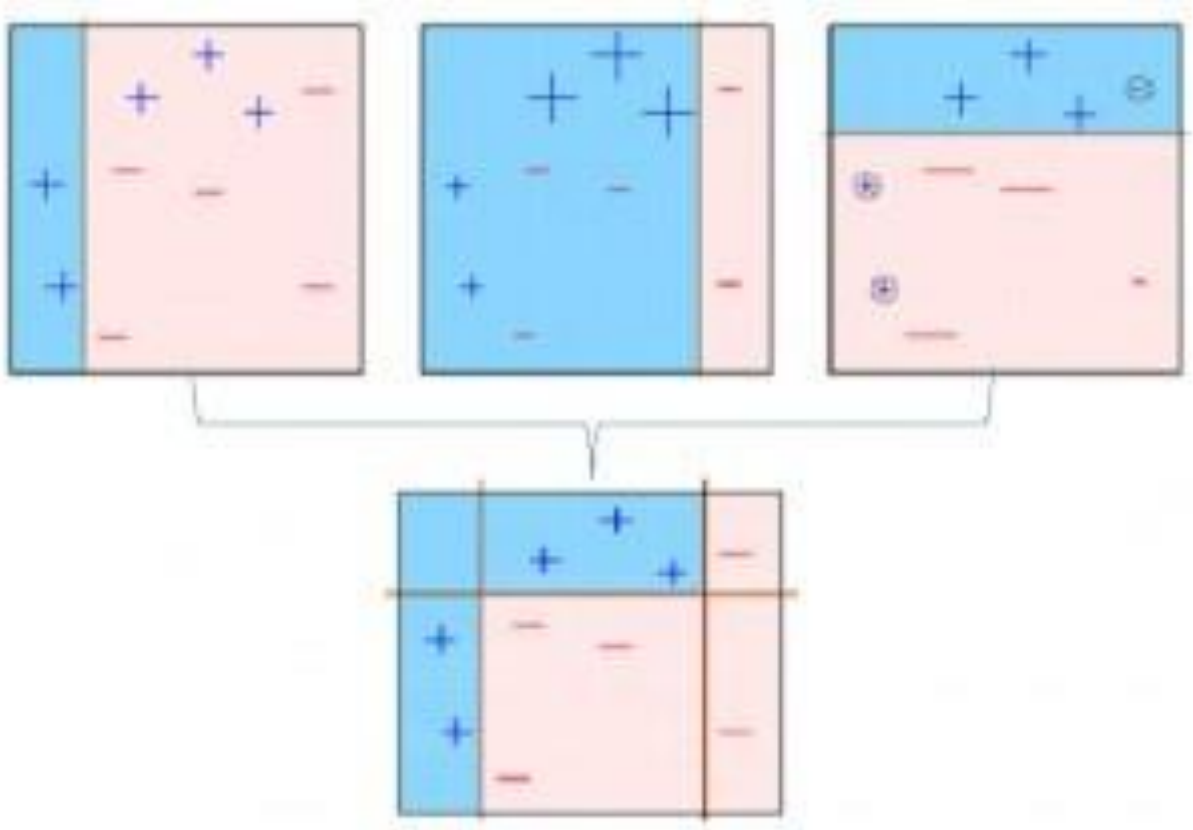
Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

Boosting - Approach :

- A subset is created from the original dataset.
- Initially, all data points are given equal weights.
- A base model is created on this subset.
- This model is used to make predictions on the whole dataset.
- Errors are calculated using the actual values and predicted values.
- The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights)
- Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)

Boosting



- Multiple models are created, each correcting the errors of the previous model.
- The final model (strong learner) is the weighted mean of all the models (weak learners).

- Individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.

AdaBoost

- Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model.
- AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

AdaBoost Algorithm

- Initially, all observations (n) in the dataset are given equal weights (1/n).
- A model is built on a subset of data.
- Using this model, predictions are made on the whole dataset.
- Errors are calculated by comparing the predictions and actual values.
- While creating the next model, higher weights are given to the data points which were predicted incorrectly.

Adaboost Algorithm

- Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
- This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

AdaBoost

Base classifiers C_i: C_1, C_2, …, C_T

Error rate:

N input samples

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

Importance of a classifier:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

https://en.wikipedia.org/wiki/AdaBoost#Choosing_αt

AdaBoost: Weight Update

Weight Update:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$ <- Eqn:5.88

where Z_j is the normalization factor

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

- Reduce weight if correctly classified else increase
- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to 1/n and the resampling procedure is repeated

AdaBoost Algorithm

innovate achieve lead

Algorithm 5.7 AdaBoost Algorithm

1: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Initialize the weights for all n instances.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all instances in the original training set, D.
7: $\epsilon_i = \frac{1}{n}\left[\sum_j w_j \, \delta\left(C_i(x_j) \neq y_j\right)\right]$ {Calculate the weighted error}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Reset the weights for all n instances.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each instance according to equation (5.88).
14: **end for**
15: $C^*(\mathbf{x}) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y))$.

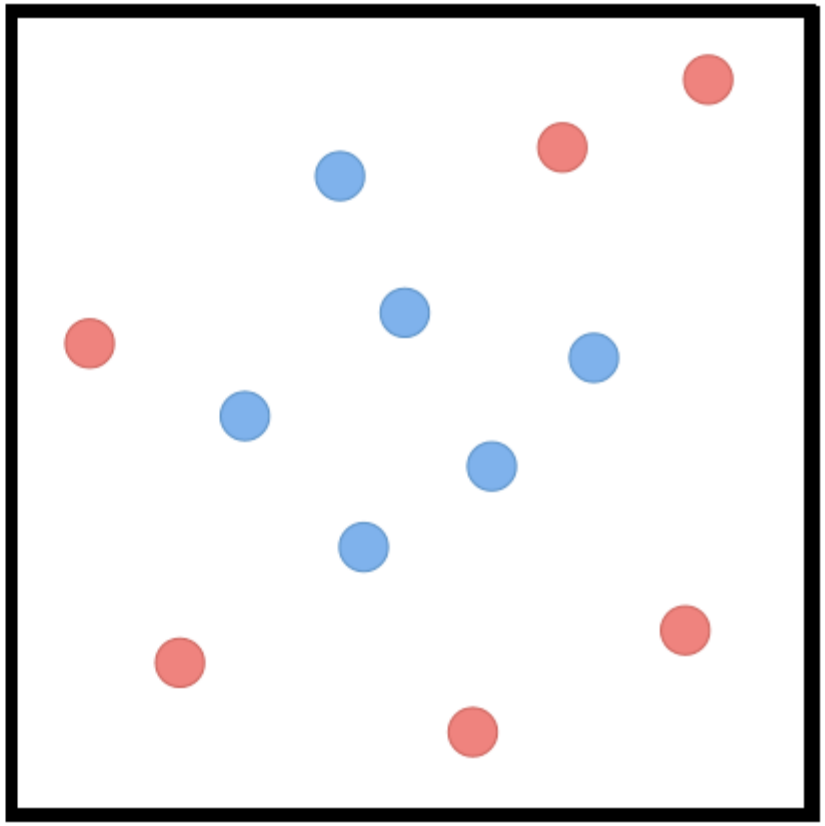
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



- Size of point represents the instance's weight

α *in earlier slide same as* $\boldsymbol{\beta}$ $=$ *weight of class*

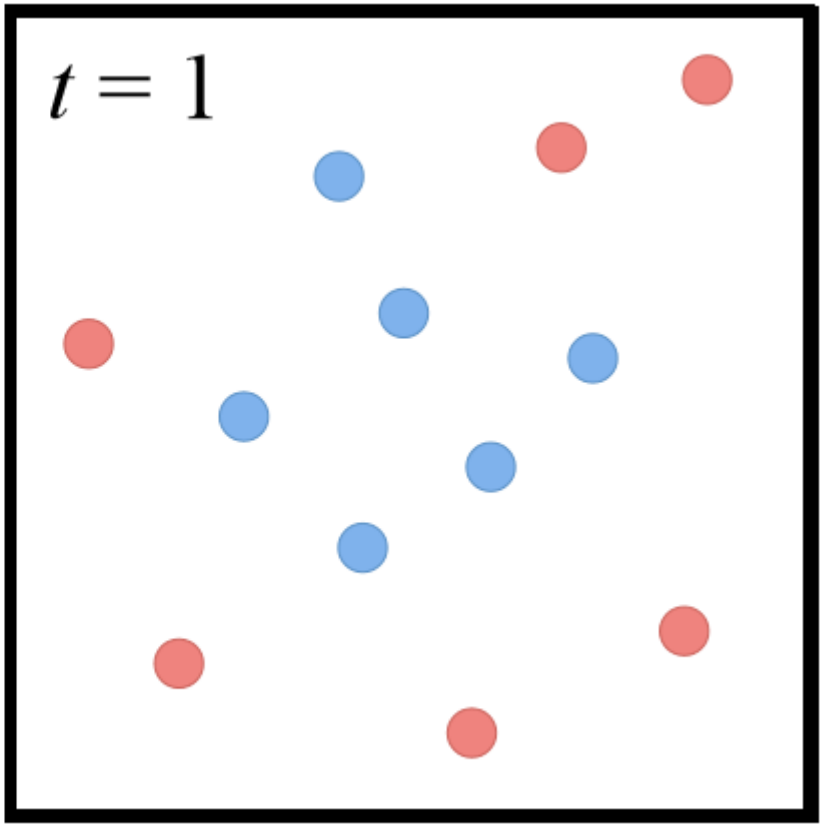
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



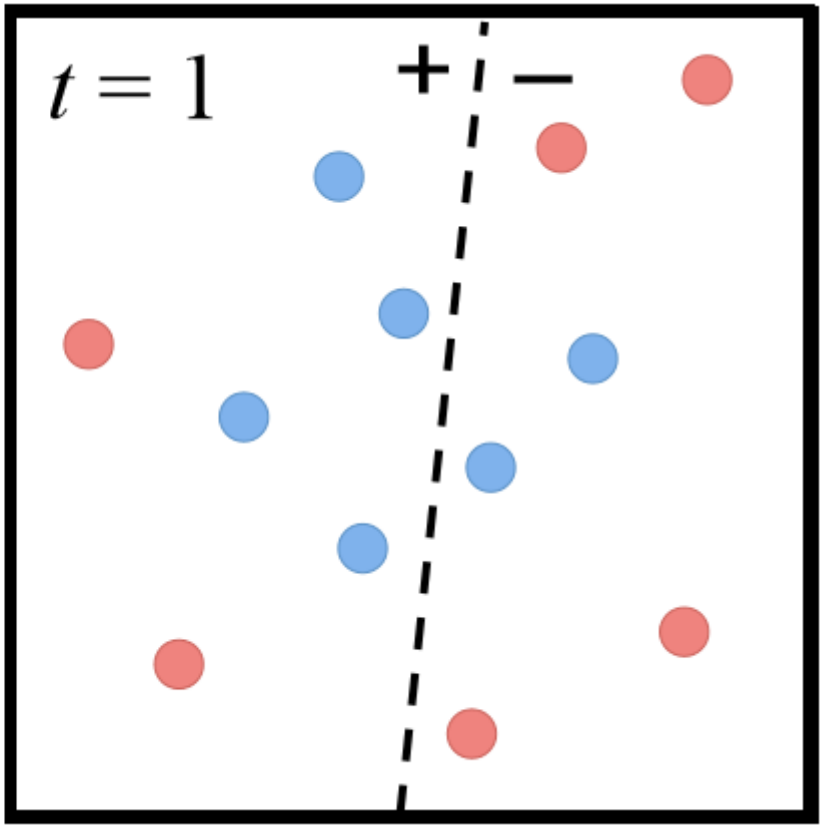
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



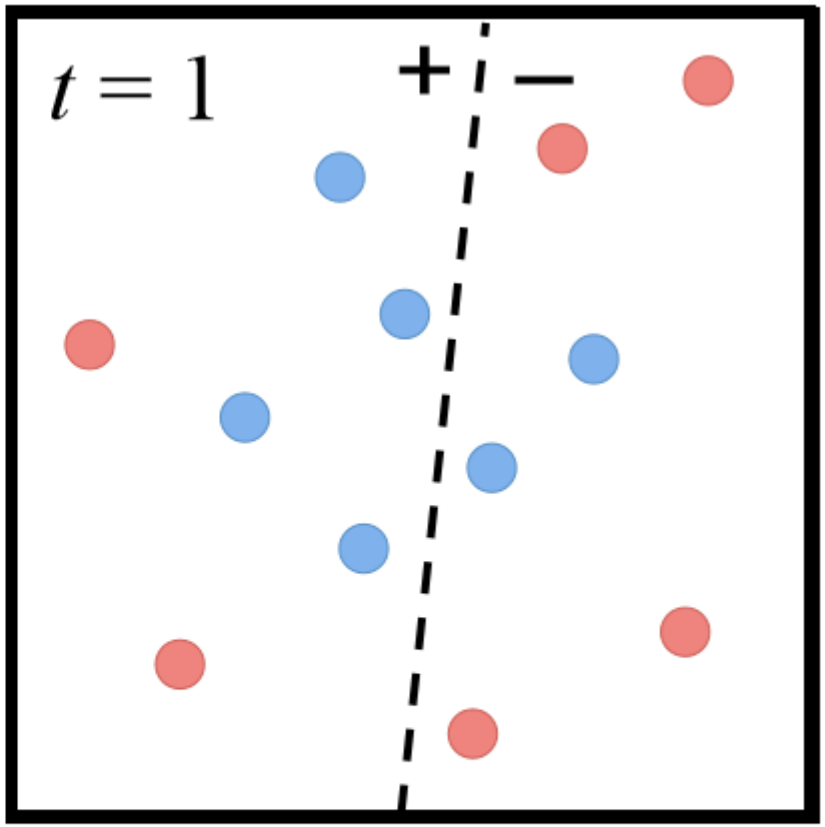
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

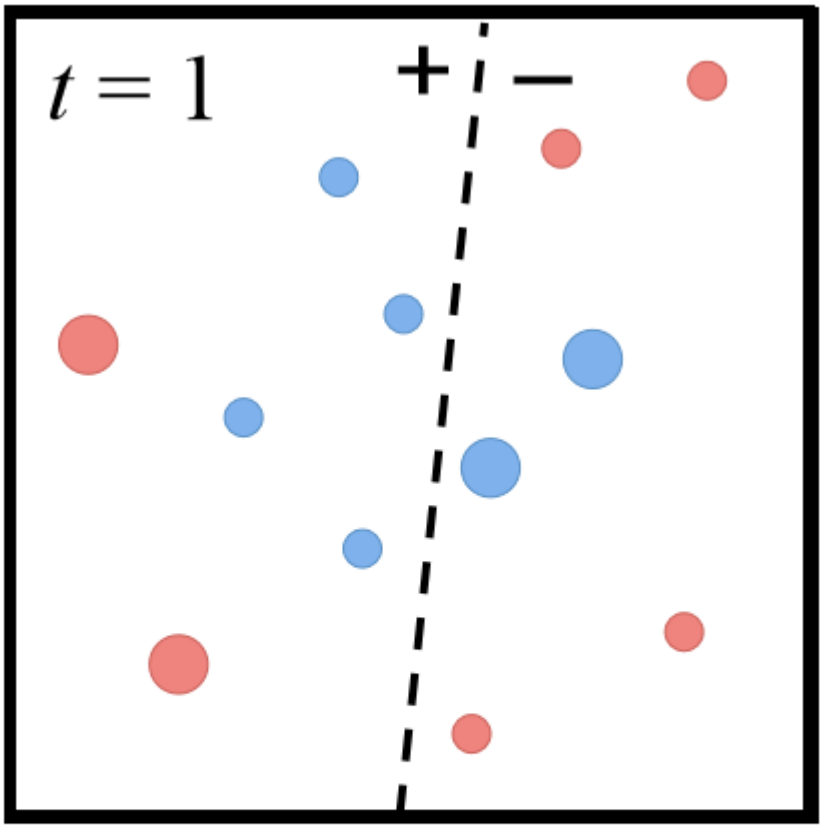
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

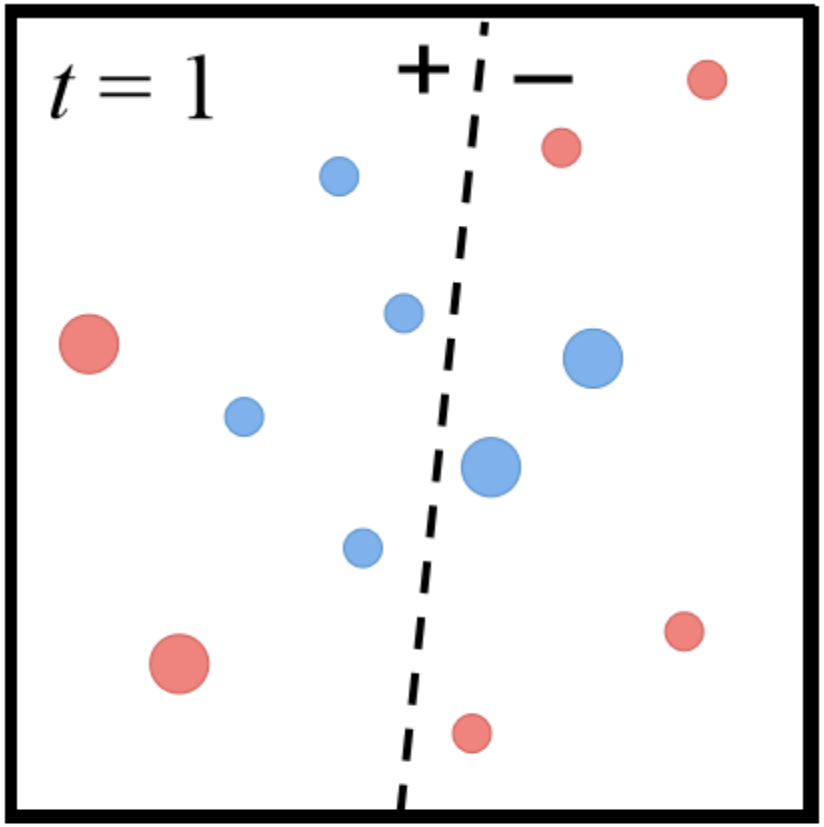
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



Disclaimer: Note that resized points in the illustration above are not necessarily to scale with β_t

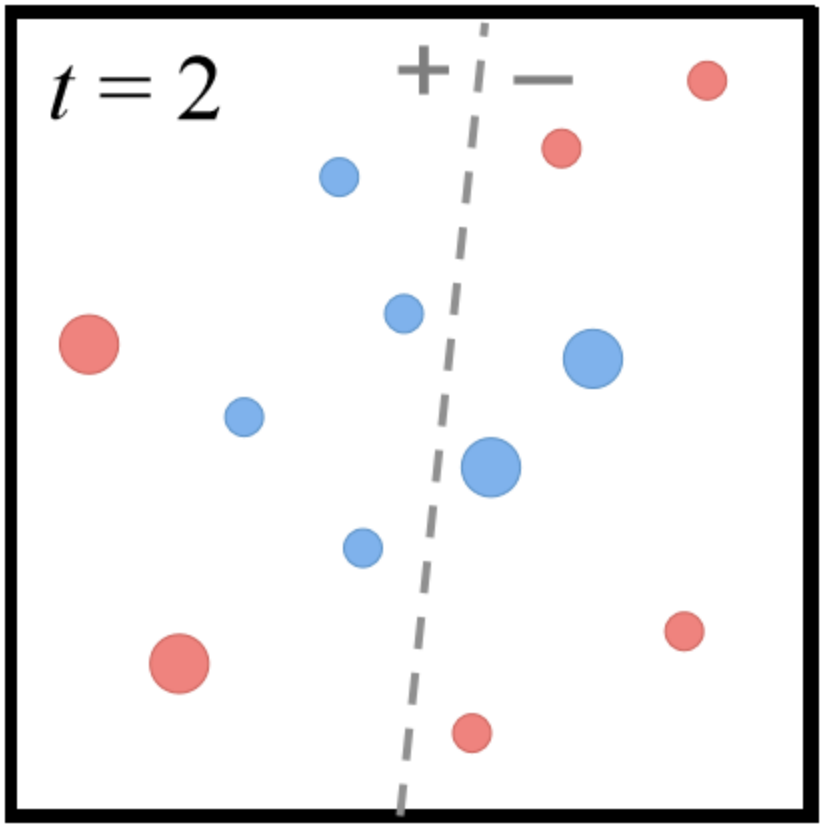
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



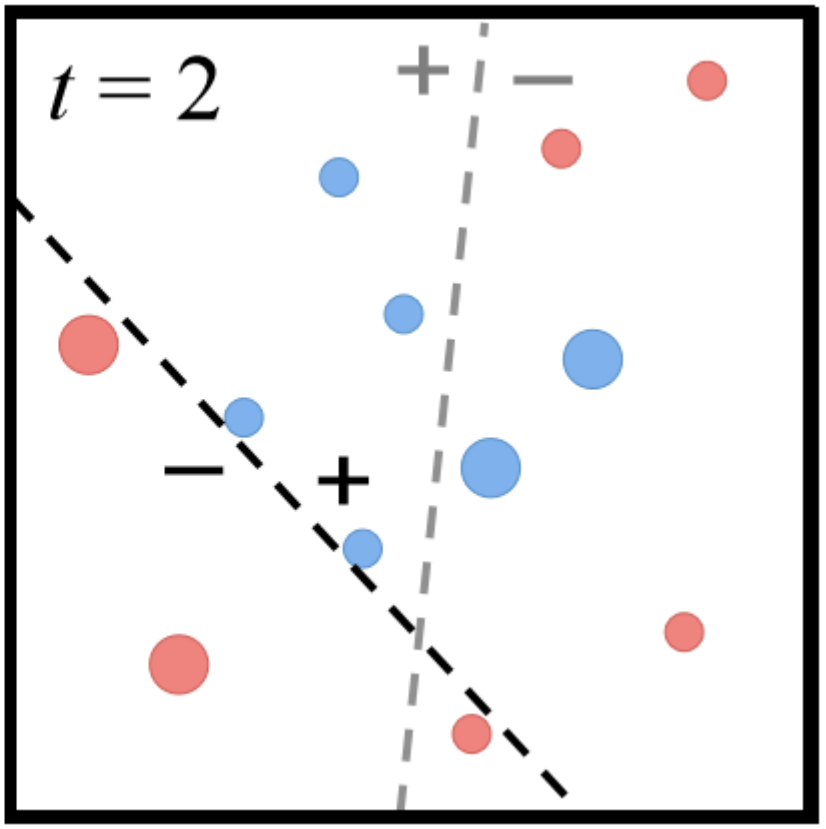
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



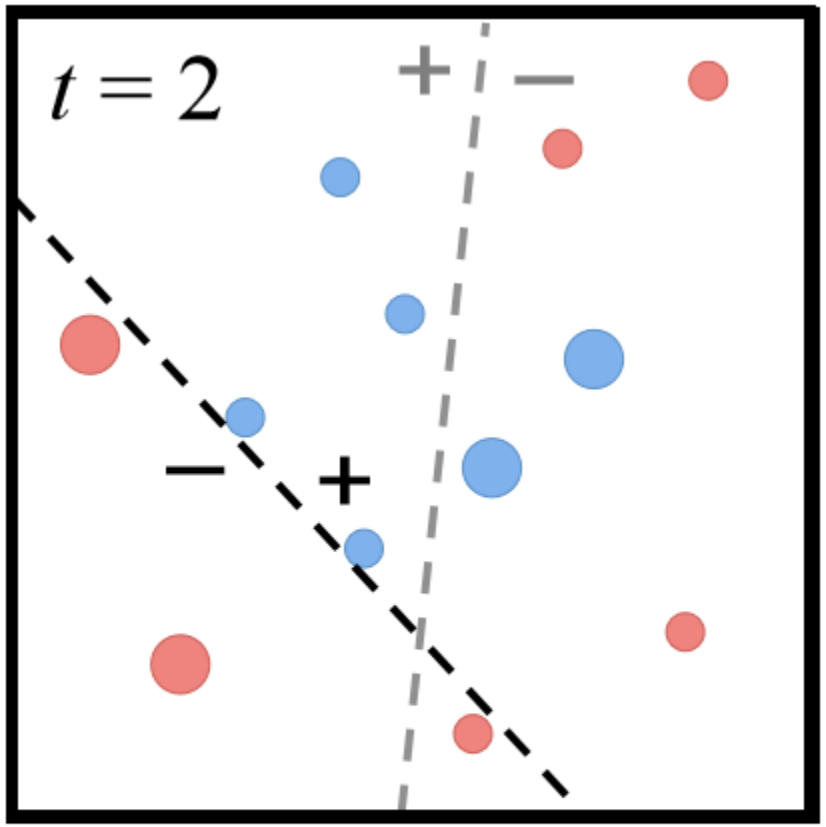
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

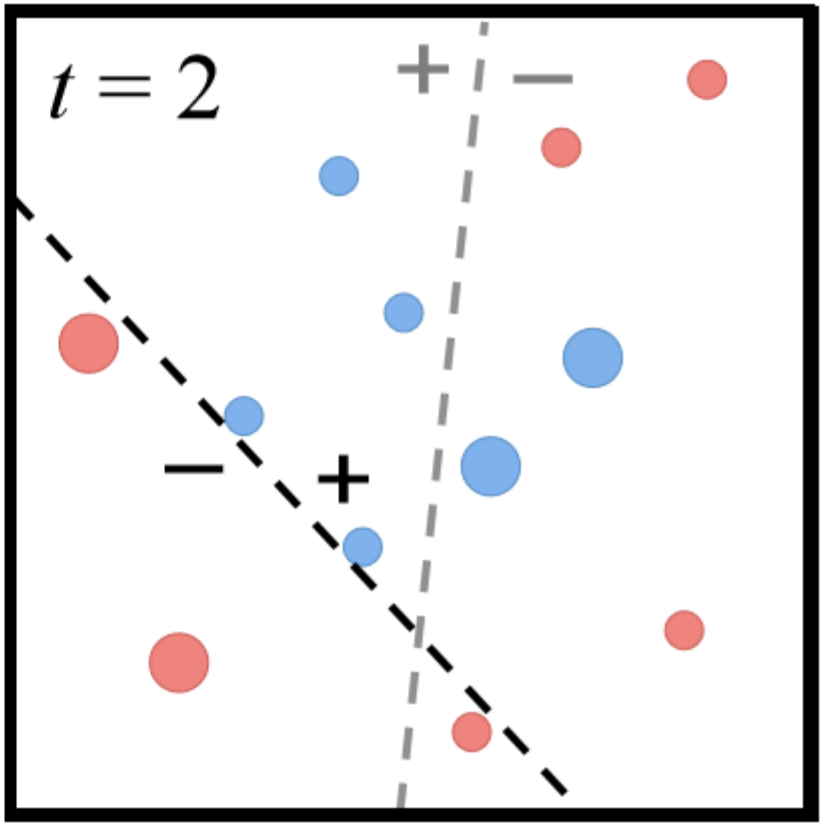
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

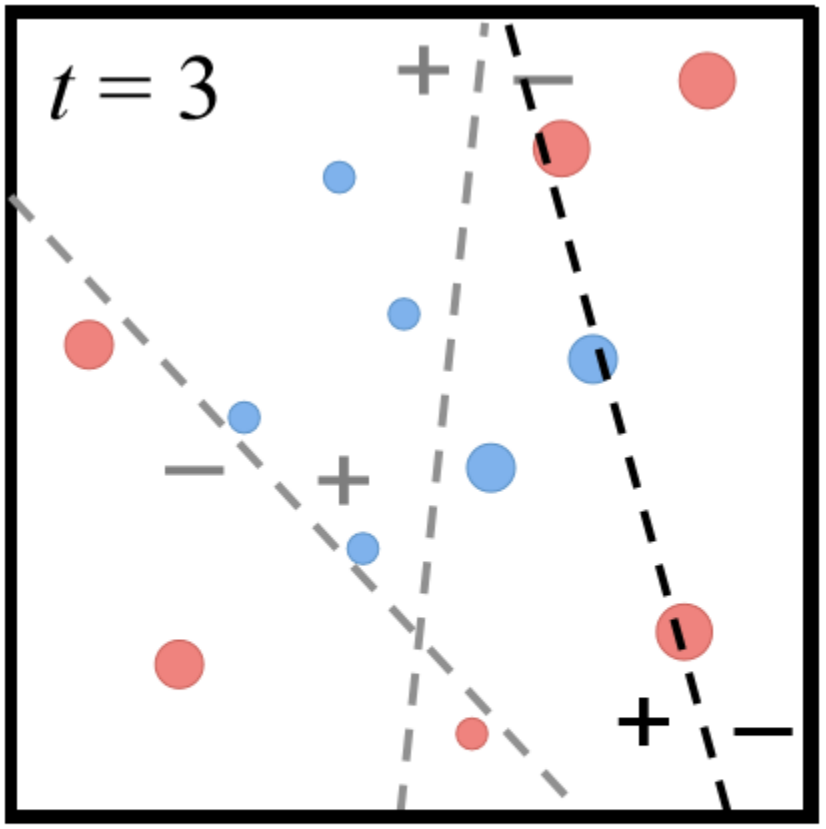
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



- β_t measures the importance of h_t
- If $\epsilon_t \leq 0.5$, then $\beta_t \geq 0$ (β_t grows as ϵ_t gets smaller)

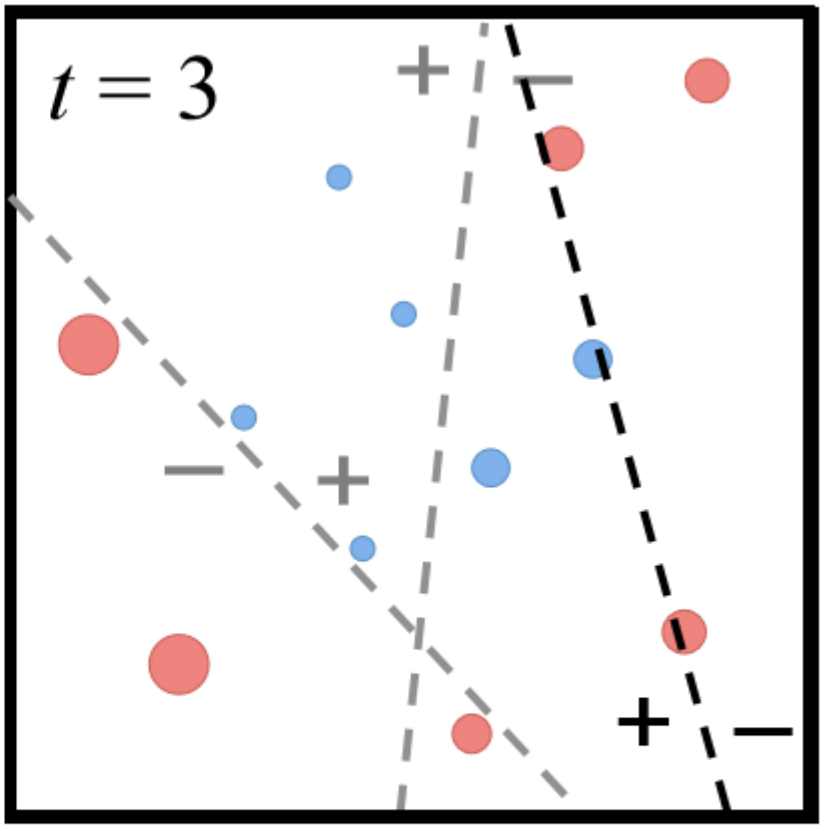
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



- Weights of correct predictions are multiplied by $e^{-\beta_t} \leq 1$
- Weights of incorrect predictions are multiplied by $e^{\beta_t} \geq 1$

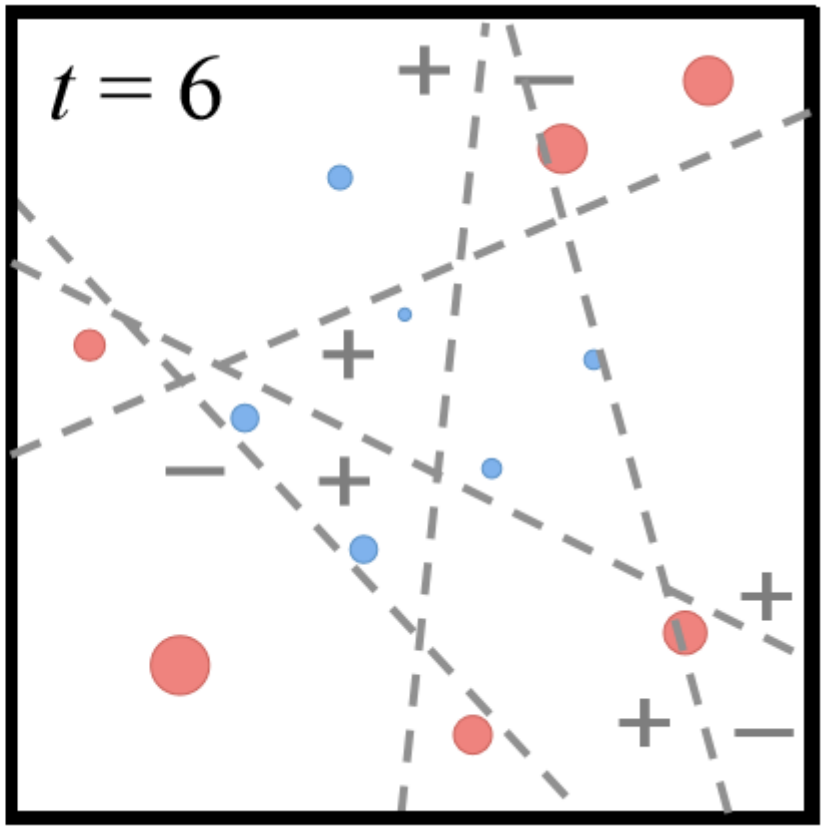
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



AdaBoost Algorithm

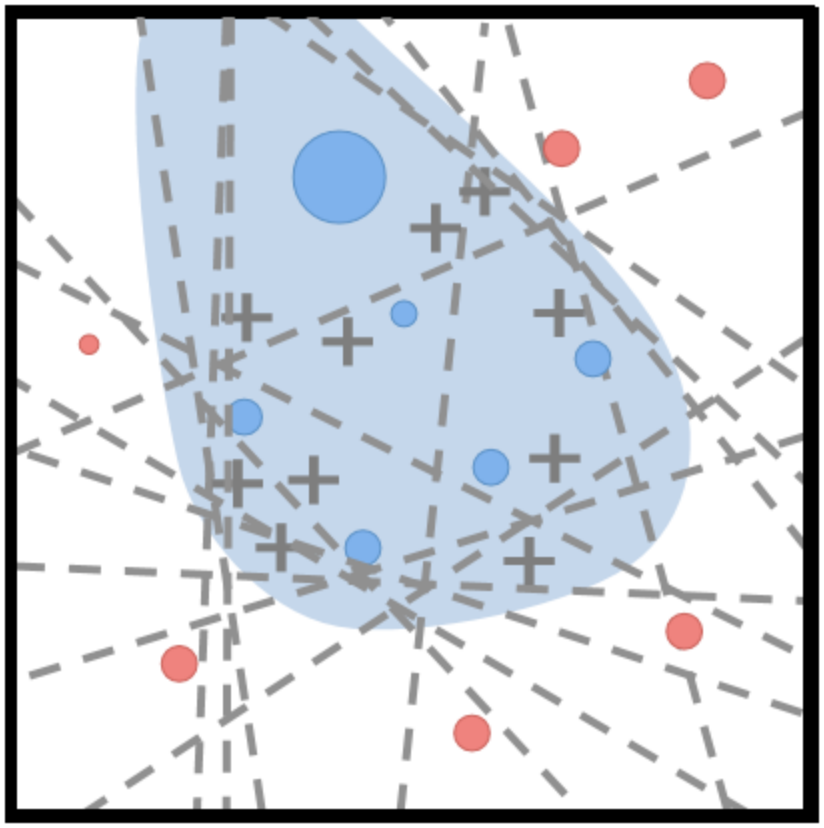
$t = \mathrm{T}$

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

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



- Final model is a weighted combination of members
  - Each member weighted by its importance

AdaBoost Algorithm

INPUT: training data $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$,
the number of iterations T

1: Initialize a vector of n uniform weights $\mathbf{w}_1 = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$

2: **for** $t = 1, \ldots, T$

3: Train model h_t on X, y with instance weights $\mathbf{w}_t$

4: Compute the weighted training error rate of h_t:

$$\epsilon_t = \sum_{i: y_i \neq h_t(\mathbf{x}_i)} w_{t,i}$$

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

6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right) \quad \forall i = 1, \ldots, n$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution:

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{n} w_{t+1,j}} \quad \forall i = 1, \ldots, n$$

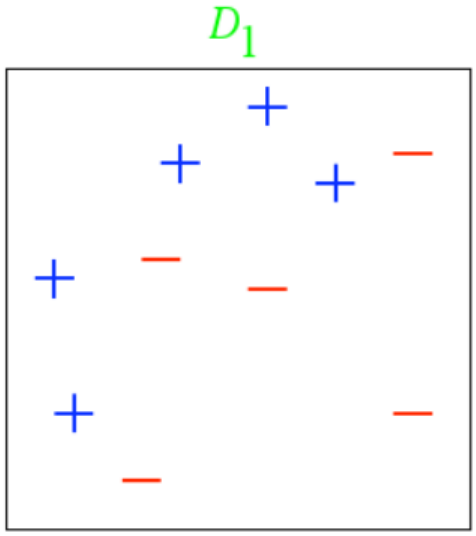
8: **end for**

9: **Return** the hypothesis

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

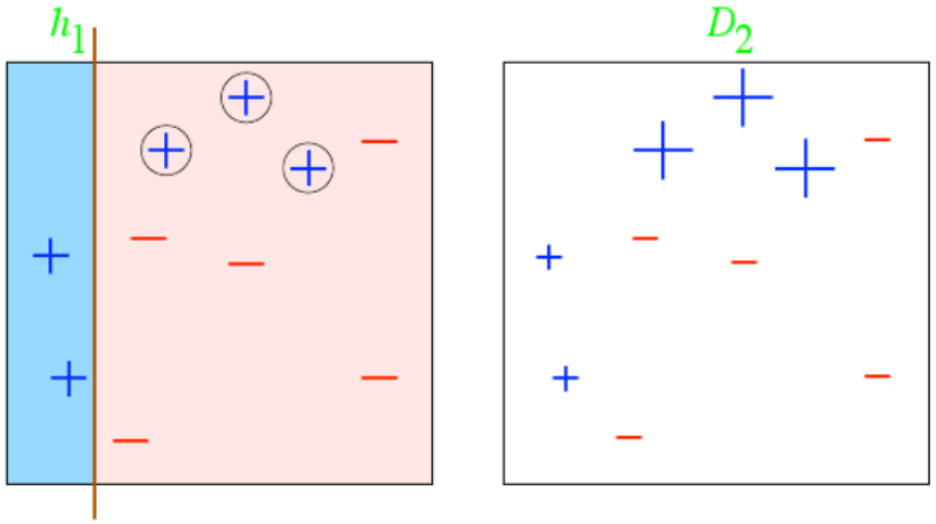
Member classifier with less error are given more weight in final ensemble hypothesis. Final prediction is a weighted combination of each members prediction

Exampl
e



From, L´eon Bottou

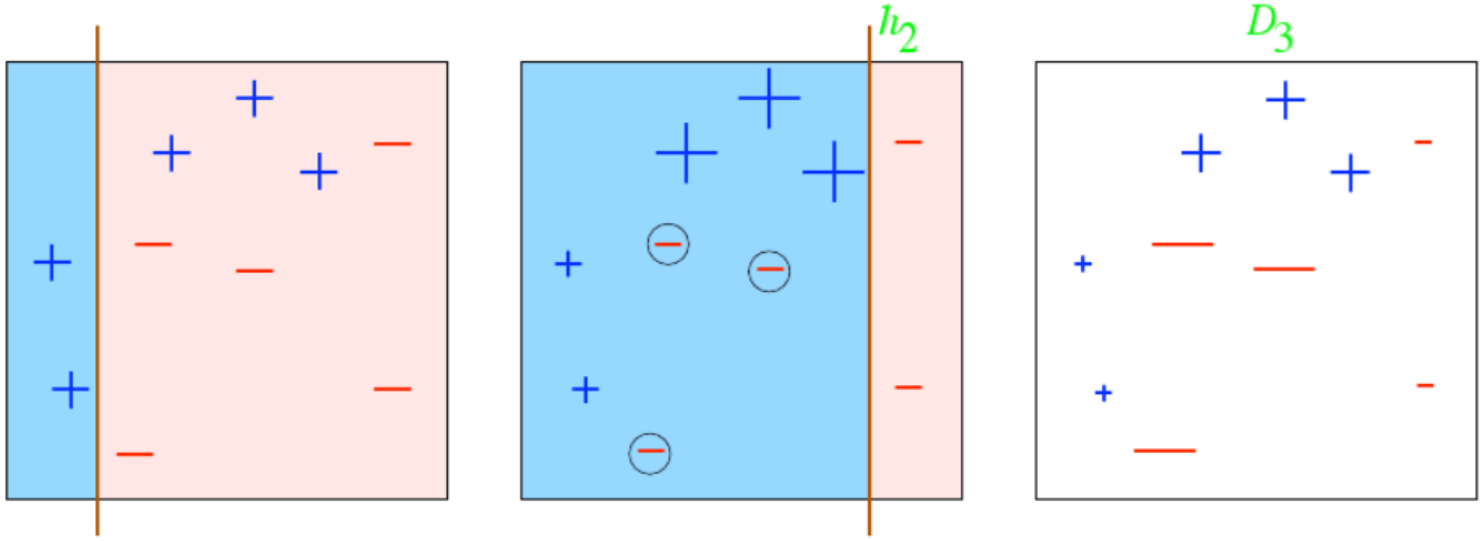
Example



From, L´eon Bottou

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right) \qquad \alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

Example



$\varepsilon_2 = 0.21$

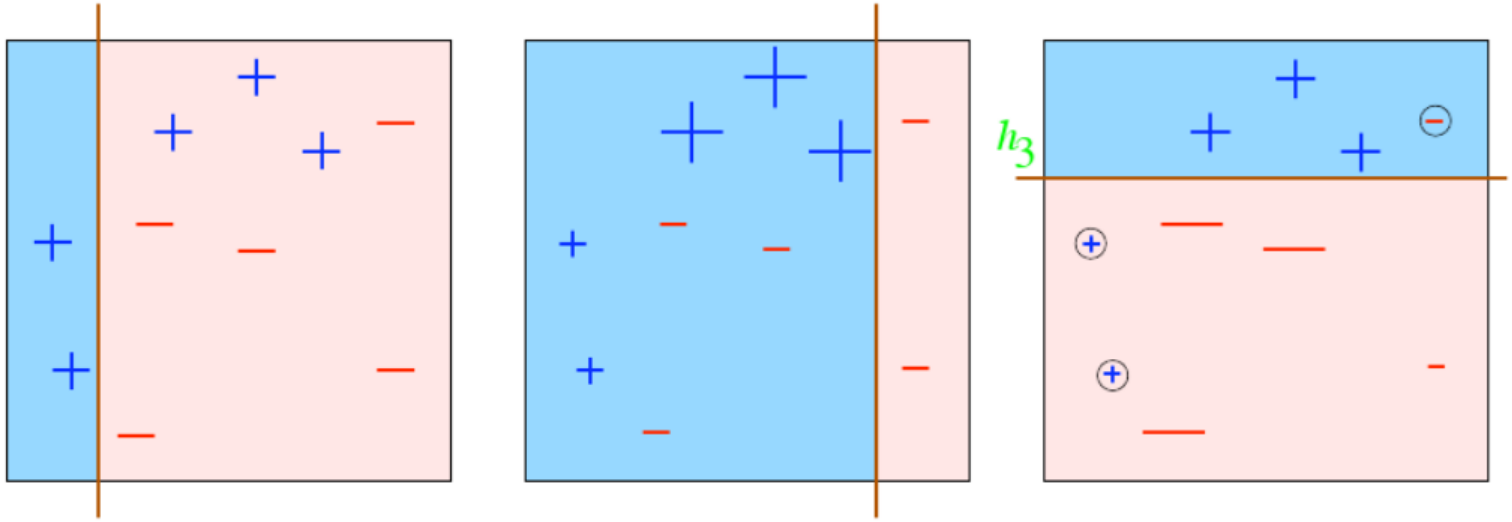
$\alpha_2 = 0.65$

From, L´eon Bottou

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

Example



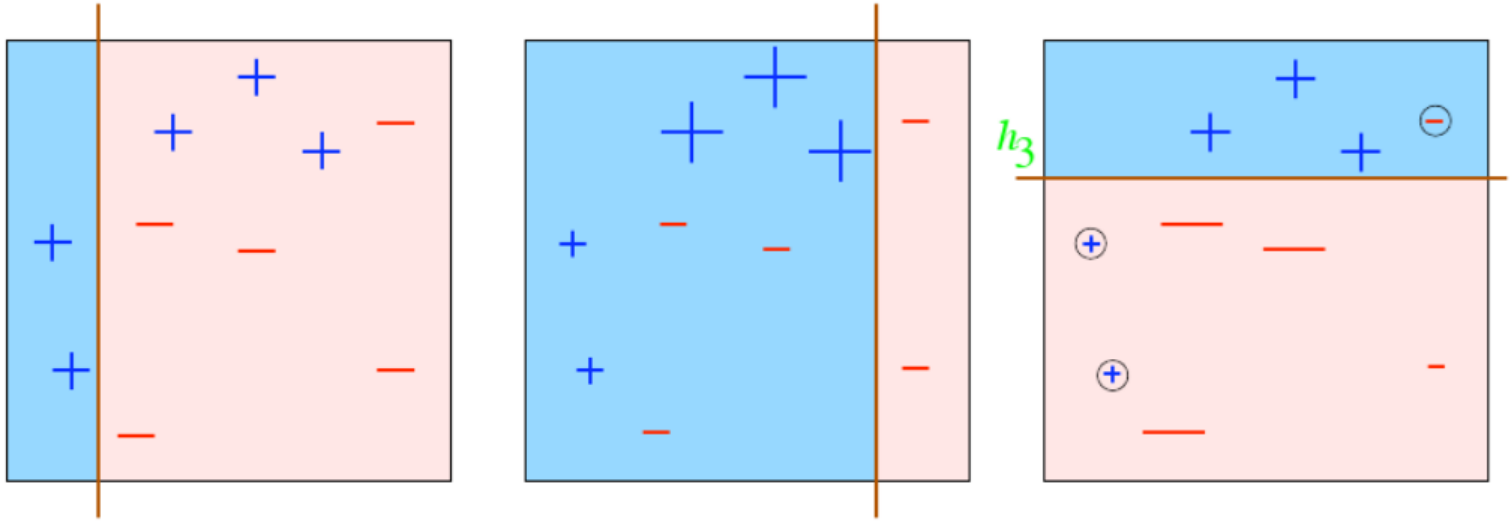
$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

From, L´eon Bottou

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

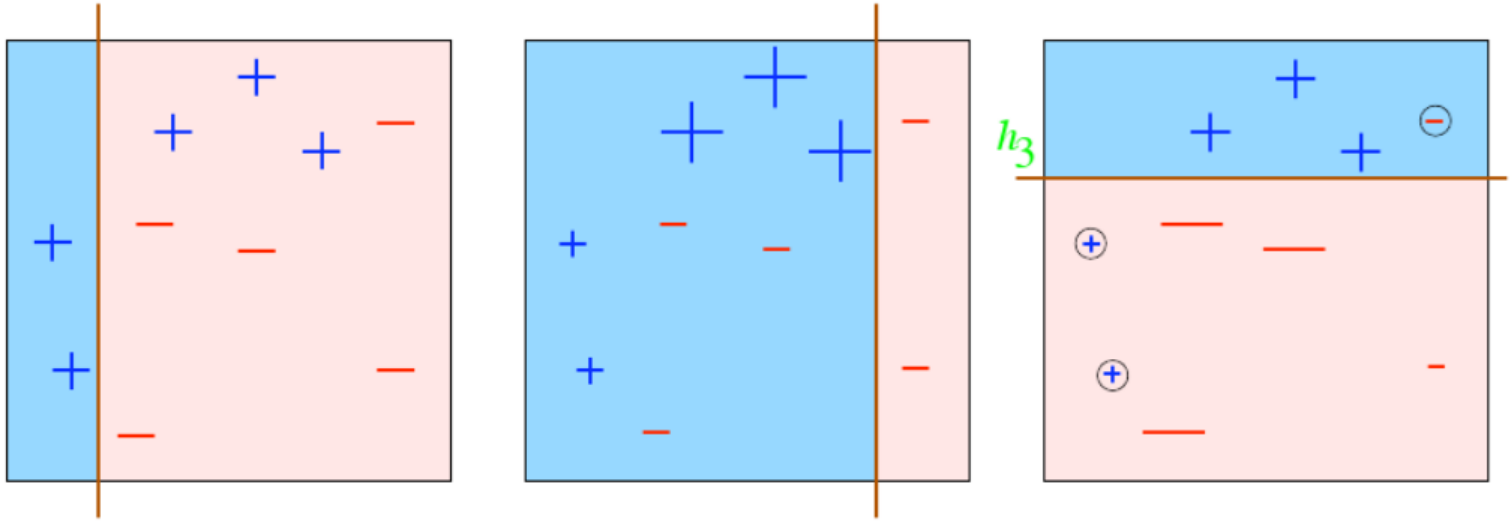
Example



From, L´eon Bottou

Example

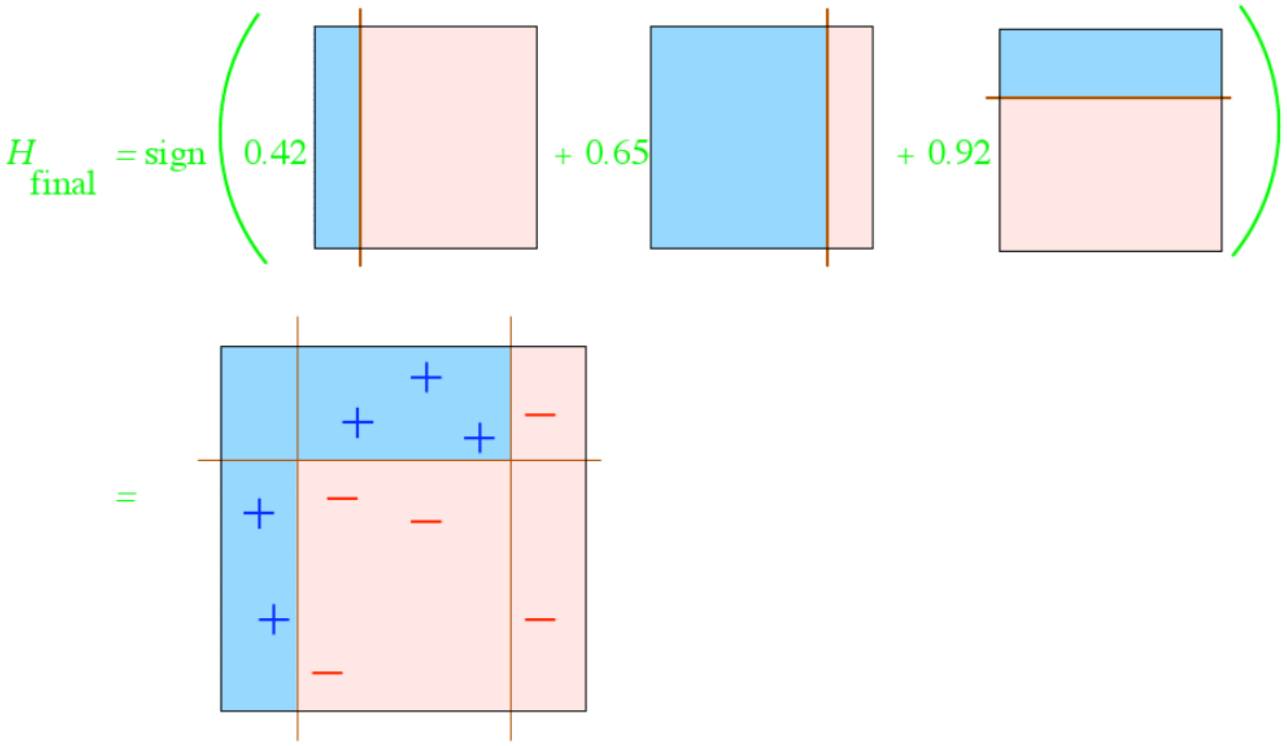
How do we combine the results now?



From, L´eon Bottou

Example

How do we combine the results now?



From, L´eon Bottou

AdaBoost

innovate achieve lead

Example

Training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

Summa

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

AdaBoost Example

Weights

Classification

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

AdaBoost error function takes into account the fact that only the sign of the final result is used, thus sum can be far larger than 1 without increasing error

AdaBoost base learners

- AdaBoost works best with "weak" learners
  - Should not be complex
  - Typically high bias classifiers
  - Works even when weak learner has an error rate just slightly under 0.5 (i.e., just slightly better than random)
    - Can prove training error goes to 0 in O(log n) iterations

- Examples:
  - Decision stumps (1 level decision trees)
  - Depth-limited decision trees
  - Linear classifiers

AdaBoost in practice

Strengths:

- Fast and simple to program
- No parameters to tune (besides T)
- No assumptions on weak learner

When boosting can fail:

- Given insufficient data
- Overly complex weak hypotheses
- Can be susceptible to noise
- When there are a large number of outliers

AdaBoost - Advantages

- Fast and Simple to Program
- No parameter tuning is required (except T)
- No assumption is made on weak learners

AdaBoost - Limitations

- Need more data.
- Affected by the presence of noise
- Doesn't work well in the presence of large number of outliers

Gradient Boosting

- The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function
- optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction.
- predictor can be any machine learning algorithm like SVM, Logistic regression, KNN , Decision tree etc. But Decision tree version of gradient boosting is much popular
- In Gradient Boosting, "shortcomings" are identified by gradients.
- Recall that, in Adaboost, "shortcomings" are identified by high-weight data points.
- Both high-weight data points and gradients tell us how to improve our model.

XGBoost

- XGBoost (Extreme Gradient Boosting) uses the gradient boosting (GBM) framework at its core.
- optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**

Gradient Boosting - Idea

- 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
- There are some mistakes:

  $F(x_1) = 0.8$, while $y_1 = 0.9$,

  $F(x_2) = 1.4$ while $y_2 = 1.3$...

  How can you improve this model?
- **Rules:**
  - You are not allowed to remove anything from F or change any parameter in F.
  - You can add an additional model (regression tree) h to F, so the new prediction will be F(x) + h(x).

Gradient Boosting

You wish to improve the model such that

- $F(x_1) + h(x_1) = y_1$
- $F(x_2) + h(x_2) = y_2$ ...
- $F(x_n) + h(x_n) = y_n$

Or, equivalently

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

$h(x_2) = y_2 - F(x_2)$ ...

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

Fit a regression tree h to data

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

- 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 tree...

Gradient boosting: Summary

- Gradient boosting involves three elements:
  - A loss function to be optimized
    - For example, regression may use a squared error and classification may use logarithmic loss.
  - A weak learner to make predictions E.g Decision tree/Decision stump
  - An additive model to add weak learners to minimize the loss function.
    - Trees are added one at a time, and existing trees in the model are not changed.
    - A gradient descent procedure is used to minimize the loss when adding trees.
    - Instead of parameters, we have weak learners
    - After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient).

Gradient boosting algorithm

let F_0 be a "dummy" constant model

for m = 1, . . . ,M

for each pair $(\mathbf{x}_i, y_i)$ in the training set

compute the pseudo-residual $R(y_i, F_{m-1}(\mathbf{x}_i))$ = negative gradient of the loss

Train a regression sub-model h_m on the pseudo-residuals

Add h_m to the ensemble: $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho . h_m(\mathbf{x})$

return the ensemble F_M

Gradient boosting: Example

Height	Age	Gender	Weight
5.4	28	Male	88
5.2	26	Female	76
5	28	Female	56
5.6	25	Male	73
6	25	Male	77
4	22	Female	57

F_0 be a "dummy" constant model

Average value is predicted.

Height	Age	Gender	Actual Weight	Predicted weight 1
5.4	28	M	88	71.2
5.2	26	F	76	71.2
5	28	F	56	71.2
5.6	25	M	73	71.2
6	25	M	77	71.2
4	22	F	57	71.2

Example credit: https://medium.com/nerd-for-tech/gradient-boost-for-regression-explained-6561eec192cb

Iteration 1

compute the pseudo-residual $R(y_i, F_{m-1}(\mathbf{x}_i))$ = negative gradient of the loss

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

Height	Age	Gender	Weight	Predicted Weight 1	Pseudo Residuals 1
5.4	28	Male	88	71.2	88-71.2 = 16.8
5.2	26	Female	76	71.2	76-71.2 = 4.8
5	28	Female	56	71.2	56-71.2 = -15.2
5.6	25	Male	73	71.2	73-71.2 = 1.8
6	25	Male	77	71.2	77-71.2 = 5.8
4	22	Female	57	71.2	57-71.2 = -14.2

Residual: $h_1(x) = y - F_0(x)$

A tree with maximum leaf nodes as 4(hyper parameter for DT) using Height, Age and Gender to predict the residuals(Error)

Example credit: https://medium.com/nerd-for-tech/gradient-boost-for-regression-explained-6561eec192cb

Iteration 1

Combining the trees to make the new prediction: $F_m(x) = F_{m-1}(x) + \rho . h_m(x)$

Learning rate : 0.1

Height	Age	Gender	Weight	Predicted Weight 1	Pseudo Residuals 1	Predicted weight 2
5.4	28	Male	88	71.2	88-71.2 = 16.8	71.2+0.1*16.8=72.9
5.2	26	Female	76	71.2	76-71.2 = 4.8	71.2+0.1*(-5.2)=70.7
5	28	Female	56	71.2	56-71.2 = -15.2	71.2+0.1*(-5.2)=70.7
5.6	25	Male	73	71.2	73-71.2 = 1.8	71.2+0.1*3.8=71.6
6	25	Male	77	71.2	77-71.2 = 5.8	71.2+0.1*3.8=71.6
4	22	Female	57	71.2	57-71.2 = -14.2	'1.2+0.1*(-14.2)=69.8

Example credit: https://medium.com/nerd-for-tech/gradient-boost-for-regression-explained-6561eec192cb

Iteration 1

Residual: $h_2(x) = y - F_1(x)$

Height	Age	Gender	Weight	Predicted Weight 1	Pseudo Residuals 1
5.4	28	Male	88	71.2	88-71.2 = 16.8
5.2	26	Female	76	71.2	76-71.2 = 4.8
5	28	Female	56	71.2	56-71.2 = -15.2
5.6	25	Male	73	71.2	73-71.2 = 1.8
6	25	Male	77	71.2	77-71.2 = 5.8
4	22	Female	57	71.2	57-71.2 = -14.2

Predicted weight 2	Pseudo Residuals2
71.2+0.1*16.8=72.9	88-72.9 = 15.1
71.2+0.1*(-5.2)=70.7	76-70.7 = 5.3
71.2+0.1*(-5.2)=70.7	56-70.7 = -14.7
71.2+0.1*3.8=71.6	73-71.6 = 1.4
71.2+0.1*3.8=71.6	77-71.6 = 5.4
71.2+0.1*(-14.2)=69.8	57-69.8 = -12.8

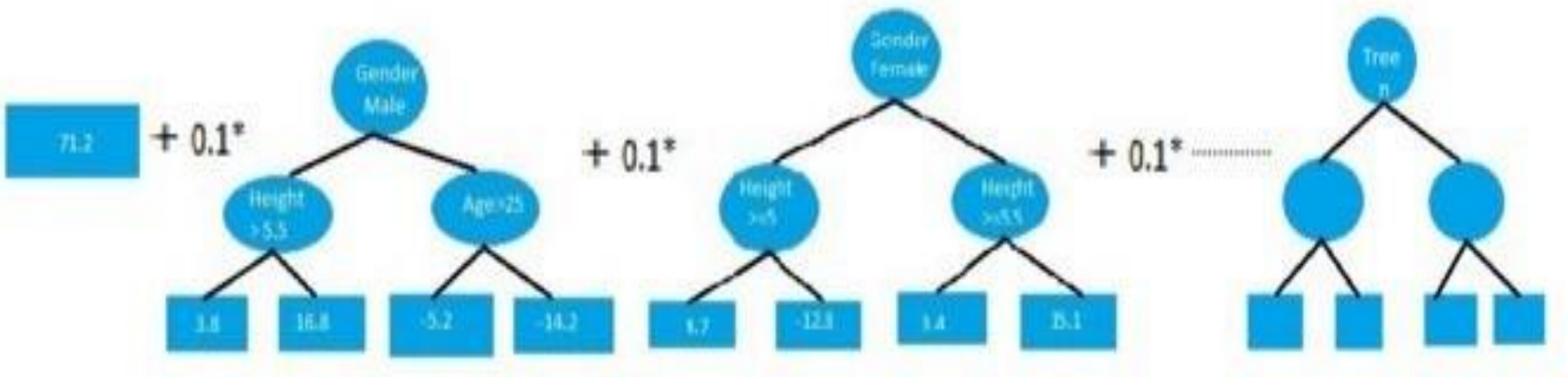
Example credit: https://medium.com/nerd-for-tech/gradient-boost-for-regression-explained-6561eec192cb

Iteration 2

Height	Age	Gender	Weight	Predicted Weight 1	Pseudo Residuals 1
5.4	28	Male	88	71.2	88-71.2 = 16.8
5.2	26	Female	76	71.2	76-71.2 = 4.8
5	28	Female	56	71.2	56-71.2 = -15.2
5.6	25	Male	73	71.2	73-71.2 = 1.8
6	25	Male	77	71.2	77-71.2 = 5.8
4	22	Female	57	71.2	57-71.2 = -14.2

Predicted weight 2	Pseudo Residuals2
71.2+0.1*16.8=72.9	88-72.9 = 15.1
71.2+0.1*(-5.2)=70.7	76-70.7 = 5.3
71.2+0.1*(-5.2)=70.7	56-70.7 = -14.7
71.2+0.1*3.8=71.6	73-71.6 = 1.4
71.2+0.1*3.8=71.6	77-71.6 = 5.4
71.2+0.1*(-14.2)=69.8	57-69.8 = -12.8

Predicted Weight 3
71.2+0.1*16.8+0.1*15.1 = 74.4
71.2+0.1*(-5.2)+0.1*(-4.7) = 70.2
71.2+0.1*(-5.2)+0.1*(-4.7) = 70.2
71.2+0.1*3.8+0.1*3.4 = 71.9
71.2+0.1*3.8+0.1*3.4 = 71.9
71.2+0.1*(-14.2)+0.1*(-12.8) = 68.5



Example credit: https://medium.com/nerd-for-tech/gradient-boost-for-regression-explained-6561eec192cb

Additional Reading Material

Source Credit : Sebastian Raschka

Gradient Boosting -- Conceptual Overview

- **Step 1:** Construct a base tree (just the root node)
- **Step 2:** Build next tree based on errors of the previous tree
- **Step 3:** Combine tree from step 1 with trees from step 2. Go back to step 2.

Gradient Boosting -- Conceptual Overview --> A Regression-based Example

In million US Dollars

x1# Rooms	x2=City	x3=Age	y=Price
5	Boston	30	1.5
10	Madison	20	0.5
6	Lansing	20	0.25
5	Waunakee	10	0.1

- **Step 1:** Construct a base tree (just the root node)

$$\hat{y_1} = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} = 0.5875$$

Gradient Boosting -- Conceptual Overview --> A Regression-based Example

- **<u>Step 2:</u>** Build next tree based on errors of the previous tree

First, compute (pseudo) residuals: $r_1 = y_1 - \hat{y}_1$

In million US Dollars

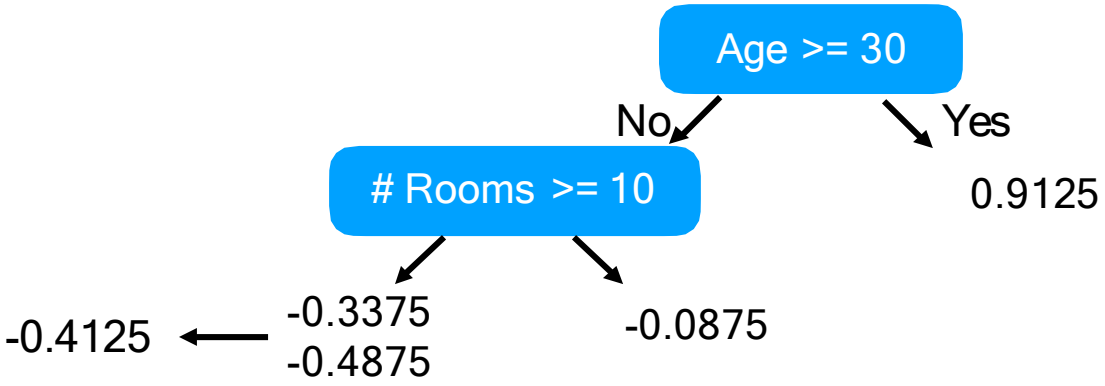
x1#	x2=City	x3=Age	y=Price	r1=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

Gradient Boosting -- Conceptual Overview --> A Regression-based Example

- **Step 2:** Build next tree based on errors of the previous tree

Then, create a tree based on x_1, ... , x_m to fit the residuals

x1#	x2=City	x3=Age	y=Price	r1=Residual
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

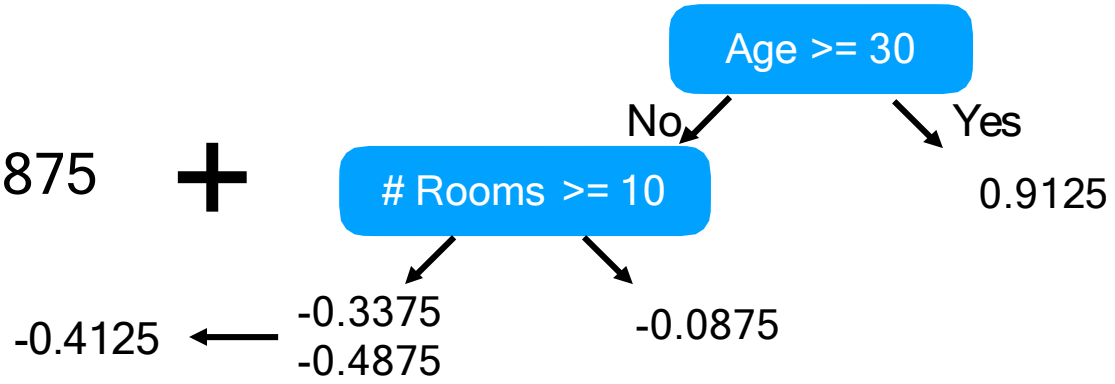


Gradient Boosting -- Conceptual Overview --> A Regression-based Example

- **Step 3:** Combine tree from step 1 with trees from step 2

x1#	x2=City	x3=Age	y=Price	r=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

$$\hat{y_1} = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0.5875$$



Gradient Boosting -- Conceptual Overview --> A Regression-based Example

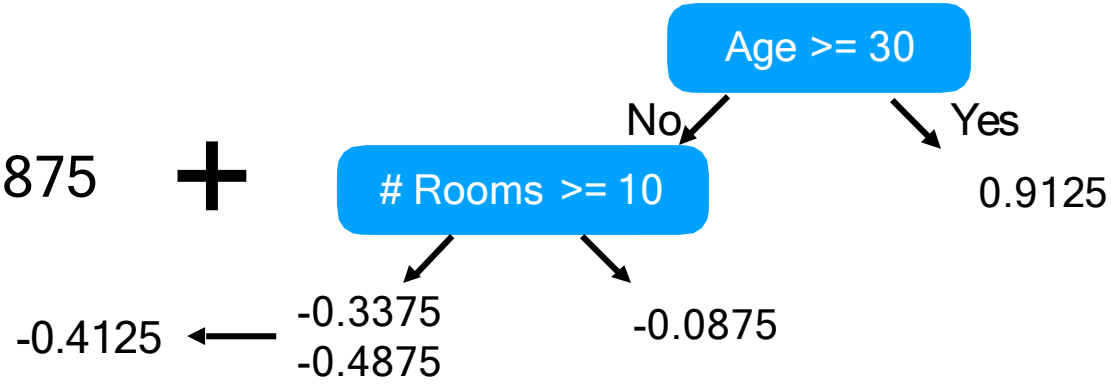
- **Step 3:** Combine tree from step 1 with trees from step 2

E.g., predict Lansing →

x1#	x2=City	x3=Age	y=Price	r=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunakee	10	0.1	0.1 - 0.5875 = -0.4875

$$\hat{y_1} = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0.5875$$

+



E.g., predict Lansing

$$0.5875 + \alpha \times (-0.4125)$$

Where learning rate between 0 and 1 (if $\alpha=1$, low bias but high variance)

Gradient Boosting -- Algorithm Overview

Step 0: Input data $\{\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\}^{n}_{i=1}$

Differentiable Loss function $L(y^{(i)}, h(\mathbf{x}^{(i)}))$

Step 1: Initialize model $h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L(y^{(i)}, \hat{y})$

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]$ $h(\mathbf{x})=h_{t-1}(\mathbf{x})$ for i=1 to n

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

▪▪▪

Gradient Boosting -- Algorithm Overview

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$ for i=1 to n

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

C. for $j = 1,\ldots,J_t$, compute

$$\hat{y}_{j,t} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y})$$

D. Update $h_t(\mathbf{x}) = h_{t-1}(\mathbf{x}) + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t} \; \mathbb{I}(\mathbf{x} \in R_{j,t})$

Step 3: Return $h_t(\mathbf{x})$

Gradient Boosting -- Algorithm Overview Discussion

Step 0: Input data $\{\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{n}$

Differentiable Loss function $L(y^{(i)}, h(\mathbf{x}^{(i)}))$

E.g., Sum-squared error in regression

$$SSE' = \frac{1}{2}(y^{(i)} - h(\mathbf{x}^{(i)}))^2$$

$$\frac{\partial}{\partial h(\mathbf{x}^{(i)})} \frac{1}{2}(y^{(i)} - h(\mathbf{x}^{(i)}))^2 \quad \text{[chain rule]}$$

$$= 2 \times \frac{1}{2}(y^{(i)} - h(\mathbf{x}^{(i)})) \times (0 - 1) = -(y^{(i)} - h(\mathbf{x}^{(i)}))$$

[neg. residual]

Gradient Boosting -- Algorithm Overview Discussion

Step 1: Initialize model $h_0(\mathbf{x}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y^{(i)}, \hat{y})$

pred. target

turns out to be the average (in regression)

$$\frac{1}{n} \sum_{i=1}^{n} y^{(i)}$$

Gradient Boosting -- Algorithm Overview Discussion

Loop to make T trees (e.g., $T=100$)

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]$

pseudo residual of the t-th tree and i-th example

$h(\mathbf{x})=h_{t-1}(\mathbf{x})$ for $i = 1$ to n

Derivative of the loss function

Gradient Boosting -- Algorithm Overview Discussion

Loop to make T trees (e.g., $T=100$)

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]$

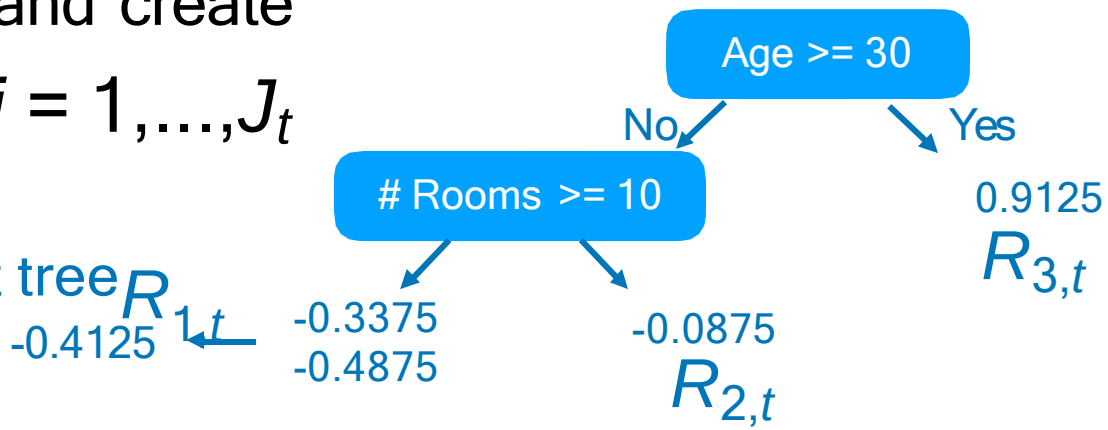
pseudo residual of the t-th tree and i-th example

$h(\mathbf{x})=h_{t-1}(\mathbf{x})$ for $i = 1$ to n

Derivative of the loss function

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

Use features in dataset to fit tree



Gradient Boosting -- Algorithm Overview Discussion

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$ for $i = 1$ to n

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

C. for $j = 1,\ldots,J_t$, compute

$$\hat{y}_{j,t} = \underset{\hat{y}}{\text{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y})$$

Compute the residual for each leaf node

Only consider examples at that leaf node

Like step 1 but add previous prediction

Gradient Boosting -- Algorithm Overview Discussion

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$ for $i = 1$ to n

B. Fit tree to $_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

C. for $j = 1,\ldots,J_t$, compute

$$\hat{y}_{j,t} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y})$$

D. Update $h_t(\mathbf{x}) = h_{t-1}(\mathbf{x}) + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t} \mathbb{I}(\mathbf{x} \in R_{j,t})$

learning rate between 0 and 1 (usually 0.1)

Summation just in case examples end up in multiple nodes

Gradient Boosting -- Algorithm Overview Discussion

For prediction, combine all T trees, e.g.,

$$h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L(y^{(i)}, \hat{y})$$

$$+\alpha\, \hat{y}_{j,t=1} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L(y^{(i)}, h_{(t=1)-1}(\mathbf{x}^{(i)}) + \hat{y})$$

...

$$+\alpha\, \hat{y}_{j,T} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L(y^{(i)}, h_{T-1}(\mathbf{x}^{(i)}) + \hat{y})$$

Gradient Boosting -- Algorithm Overview Discussion

For prediction, combine all T trees, e.g.,

$$h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L(y^{(i)}, \hat{y})$$

$$+\alpha\, \hat{y}_{j,t=1}$$

...

$$+\alpha\, \hat{y}_{j,T}$$

The idea is that we decrease the pseudo residuals by a small amount at each step

XGBoost

https://arxiv.org/abs/1603.02754

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794). ACM.

Summary and Main Points:

- scalable implementation of gradient boosting
- Improvements include: regularized loss, sparsity-aware algorithm, weighted quantile sketch for approximate tree learning, caching of access patterns, data compression, sharding
- Decision trees based on CART
- Regularization term for penalizing model (tree) complexity
- Uses second order approximation for optimizing the objective
- Options for column-based and row-based subsampling
- Single-machine version of XGBoost supports the exact greedy algorithm

References

- Introduction to Data Mining, by Pang-Ning Tan, Michael Steinbach , Vipin Kumar
- Bishop - Pattern Recognition And Machine Learning - Springer 2006
- The-Morgan-Kaufmann-Series-in-Data-Management-Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-Kaufmann-2011
- A Gentle Introduction to Gradient Boosting Cheng Li chengli@ccs.neu.edu College of Computer and Information Science Northeastern University
- https://en.wikipedia.org/wiki/Gradient_boosting#:~:text=Gradient%20boosting%20is%20a%20machine,which%20are%20typically%20decision%20trees

- https://medium.com/nerd-for-tech/gradient-boost-for-regression-explained-6561eec192cb

Next Session Plan

- Unsupervised Learning