ECE ING4
MACHINE LEARNING

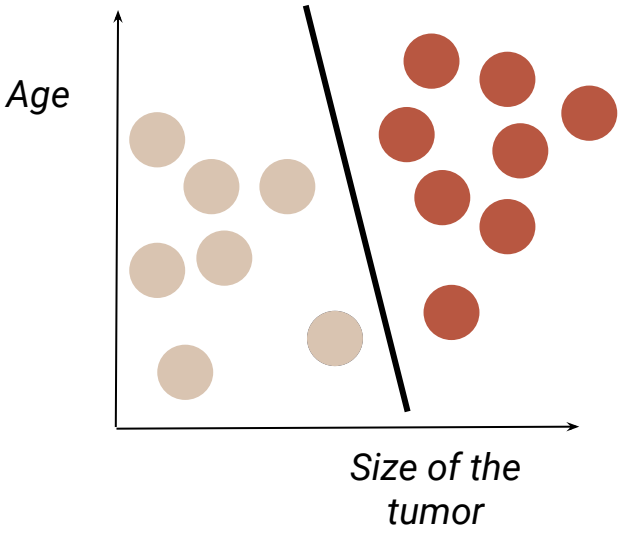
Jeremy Cohen

Decision Trees

Week 3 Review

Binary vs Multi-Class Classification

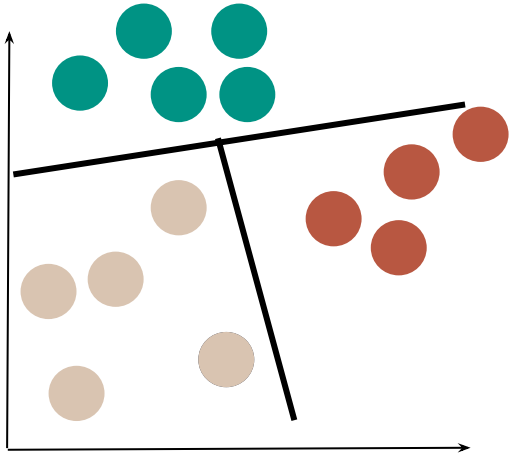
Output has 2 categories



Cancer

Not Cancer

Output has more than categories



Benign

Malignant

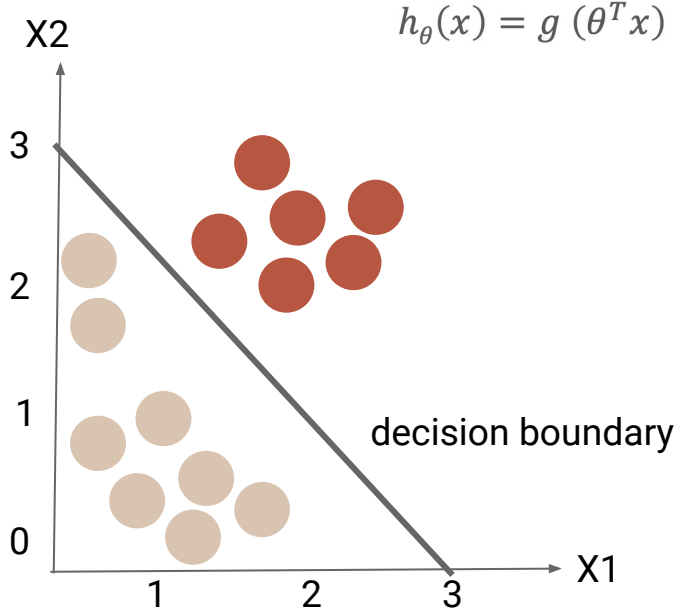
Not Cancer

Logistic Regression

$$h_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1 + e^{-\theta^T x}}$$

If $h_\theta(x) \geq 0.5$, predict "y = 1" $\quad \theta^T x \geq 0$

If $h_\theta(x) < 0.5$, predict "y = 0" $\quad \theta^T x < 0$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

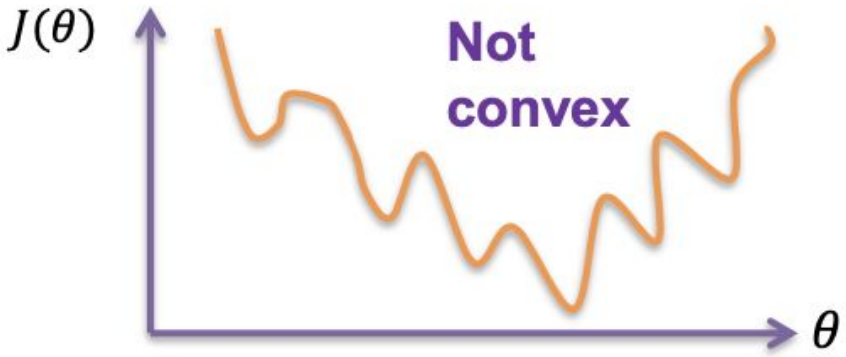
θ = -3
1
1

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

x1 + x2 ≧ 3

Cost Function for Linear Regression

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$



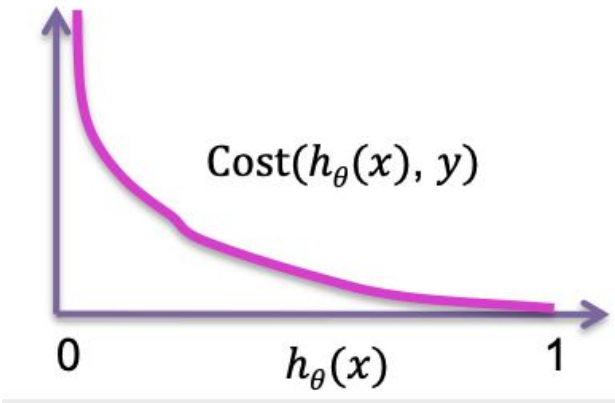
$$\frac{1}{1 + e^{-\theta^T x}}$$

Cost Function for Logistic Regression

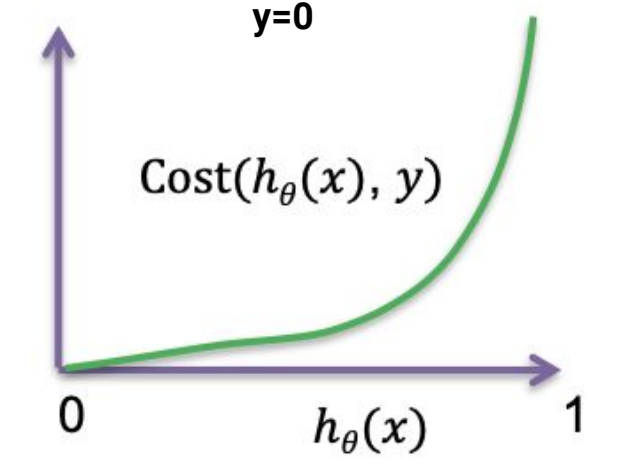
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

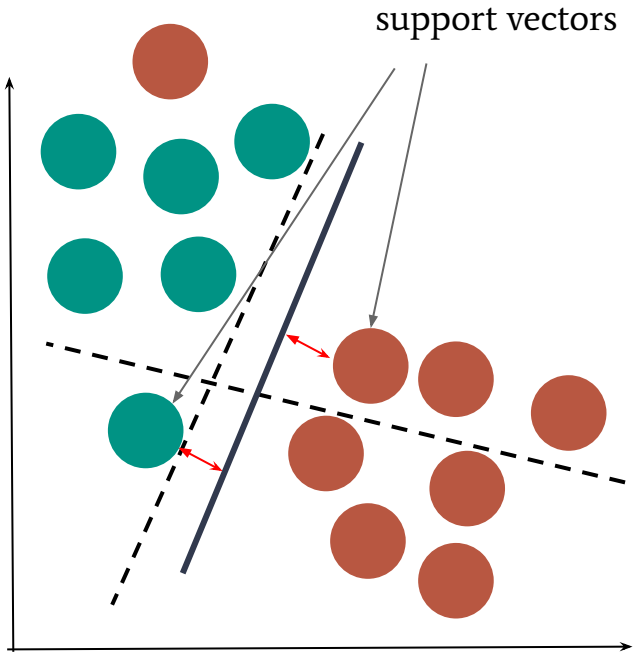
y=1



y=0



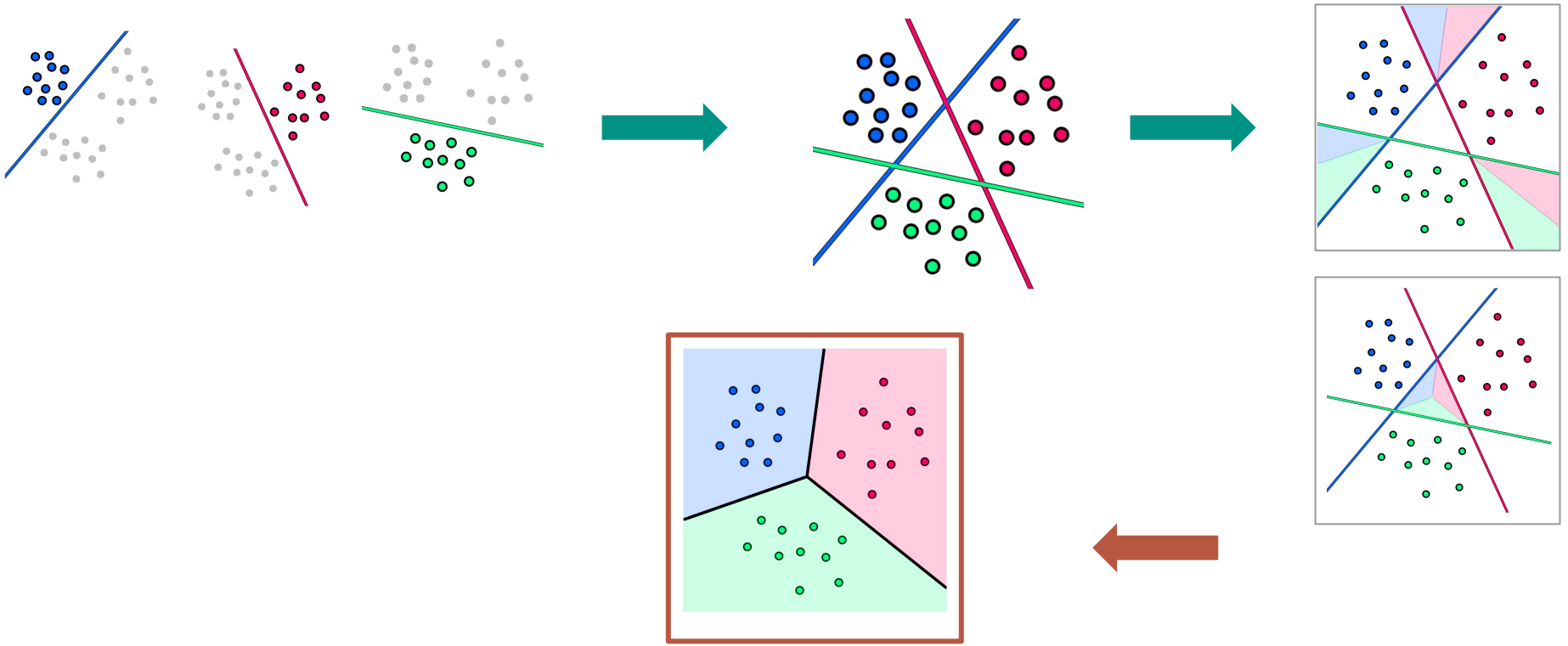
SVM



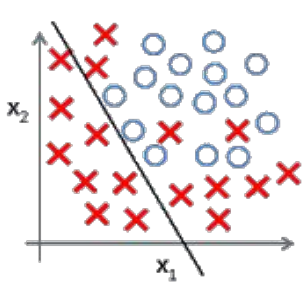
NOTABLE

- Robust to outliers/exceptions; line will not be affected

MultiClass Classification: One vs All

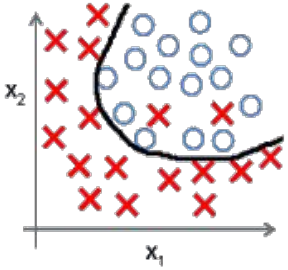


Logistic Regression

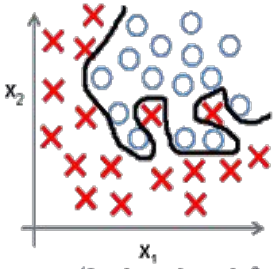


UNDERFITTING

high bias



JUST RIGHT



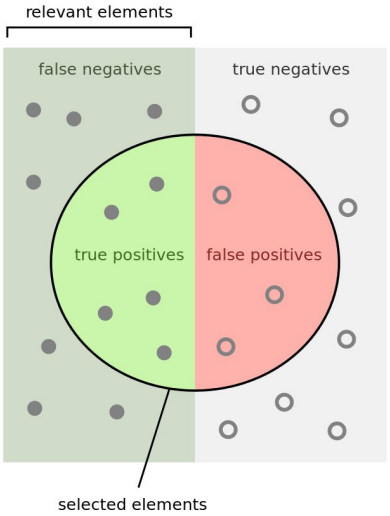
OVERFITTING

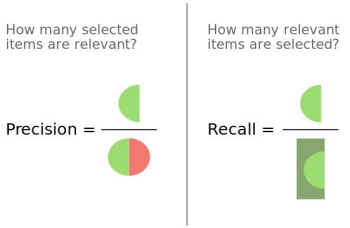
high variance

Precision & Recall

$$\text{Precision} = \frac{True\ positive}{True\ positive + False\ positive}$$

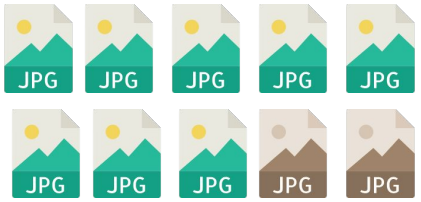
$$\text{Recall} = \frac{True\ positive}{True\ positive + False\ negative}$$





K-Fold Cross Validation

Here, the test set is 20% of the dataset.
We loose ⅕ of the dataset.

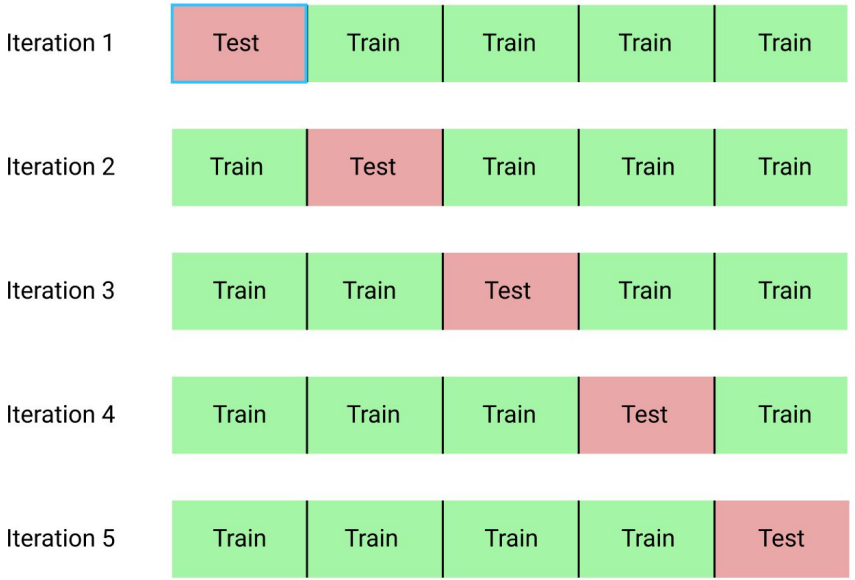


Dataset

Run K-Experiments in which we:

- Choose a test set
- Train
- Test

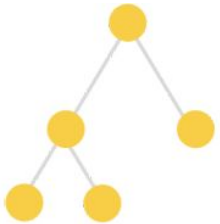
After all the experiments, we average the results



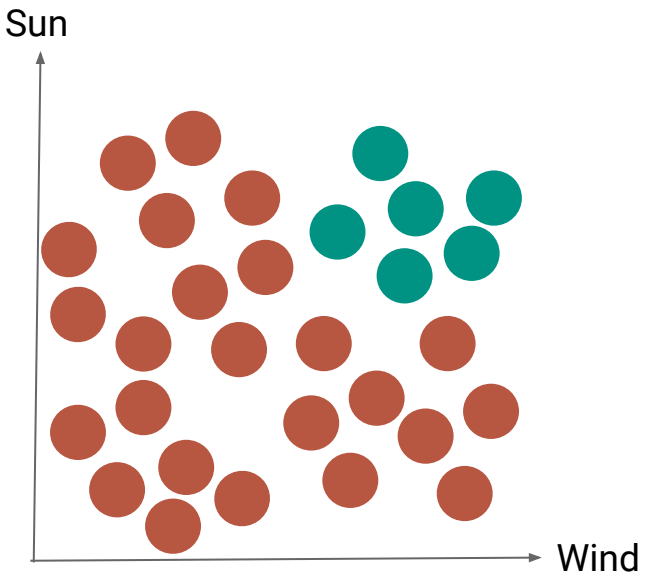
Decision Trees

Advantages

- **One of the most used algorithm in Machine Learning for classification (binary and multi-class)**
- **Extremely robust**
- **Very intuitive, the results can be visualized**



Is this data linearly separable?

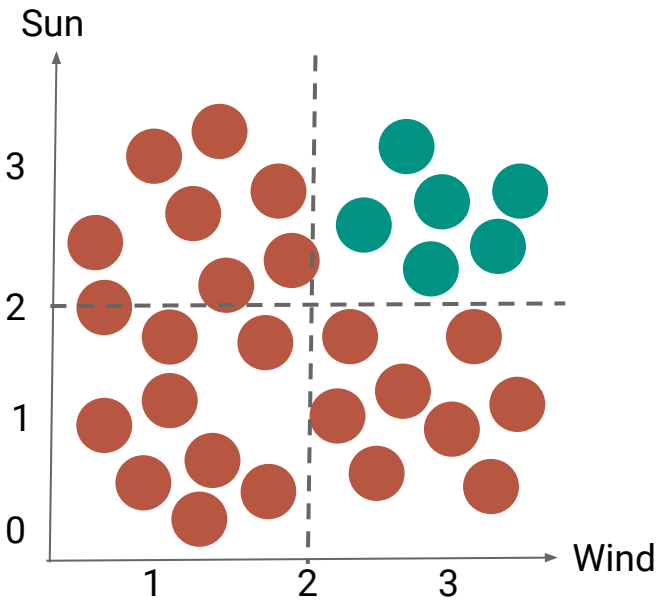


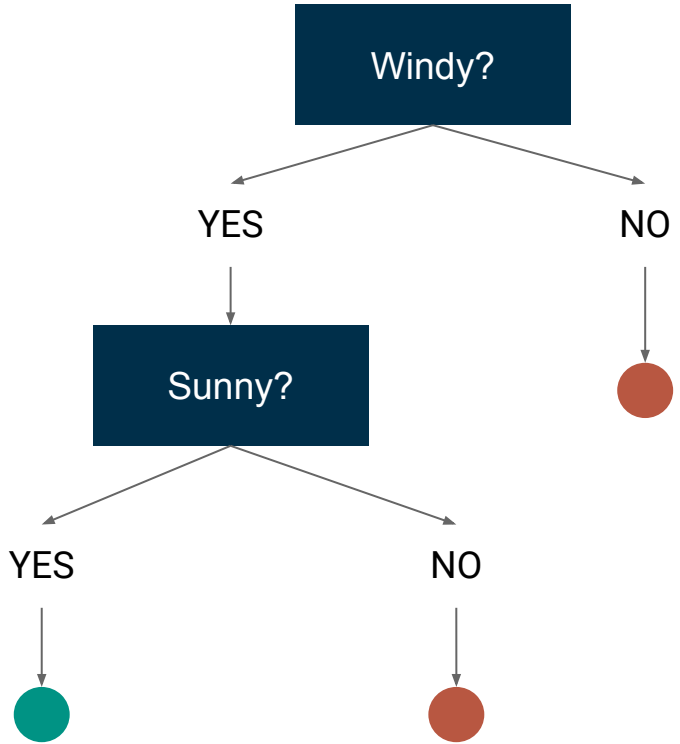
SURF

DON'T SURF

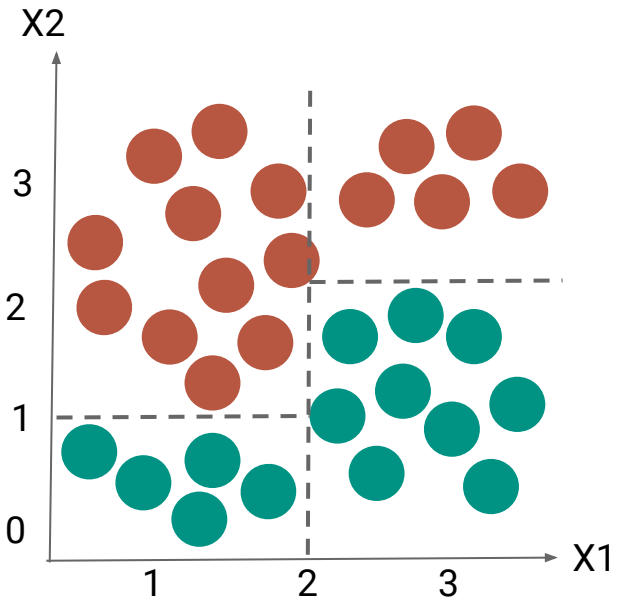
The data is not linearly separable

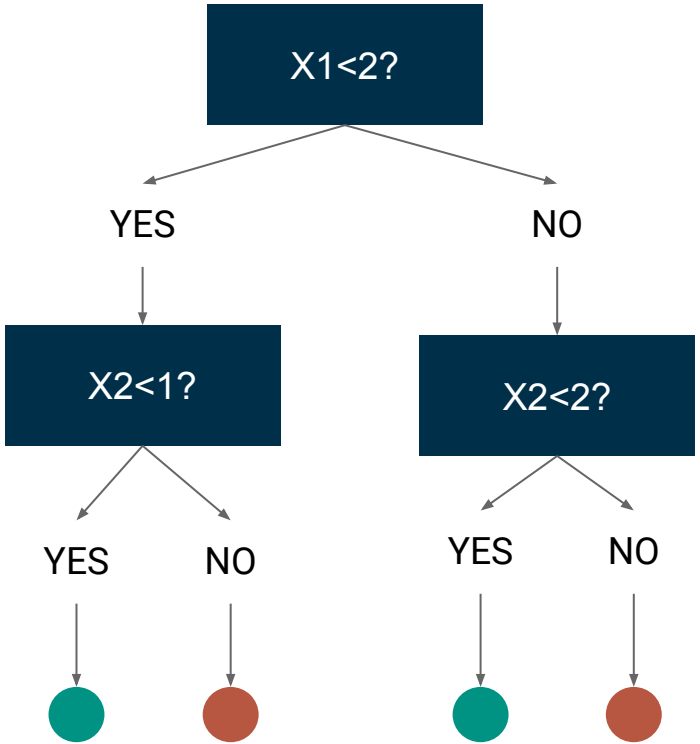
Intuition



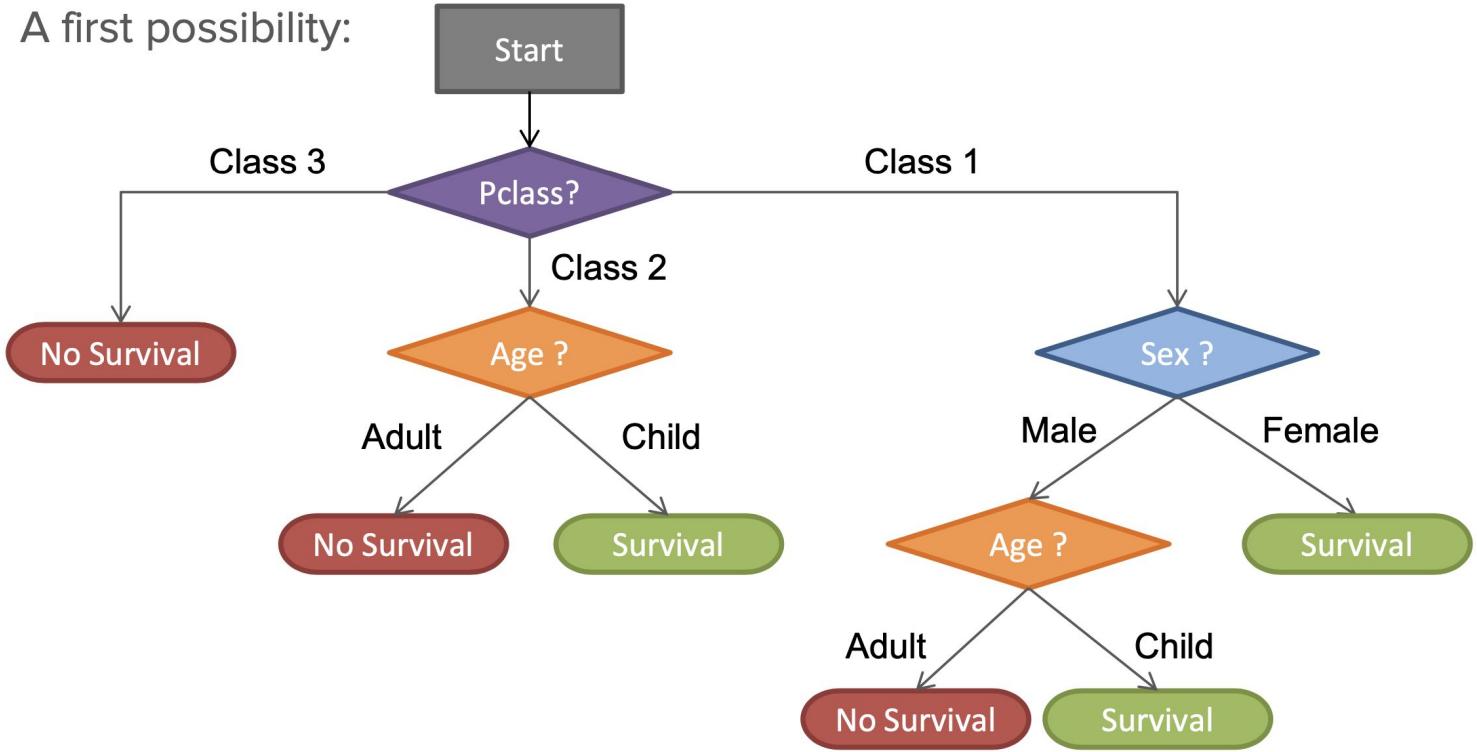


Constructing a Decision Tree

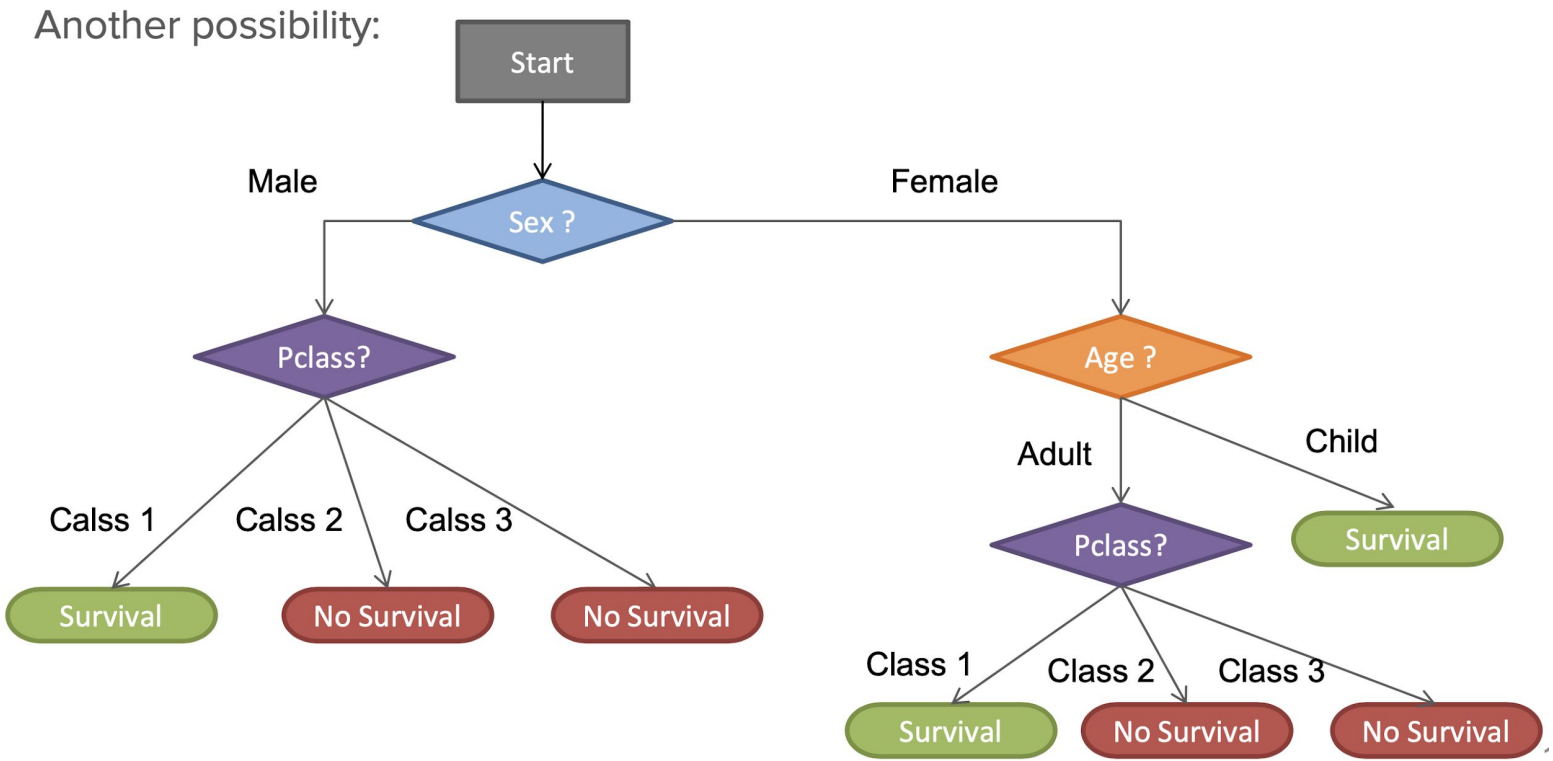




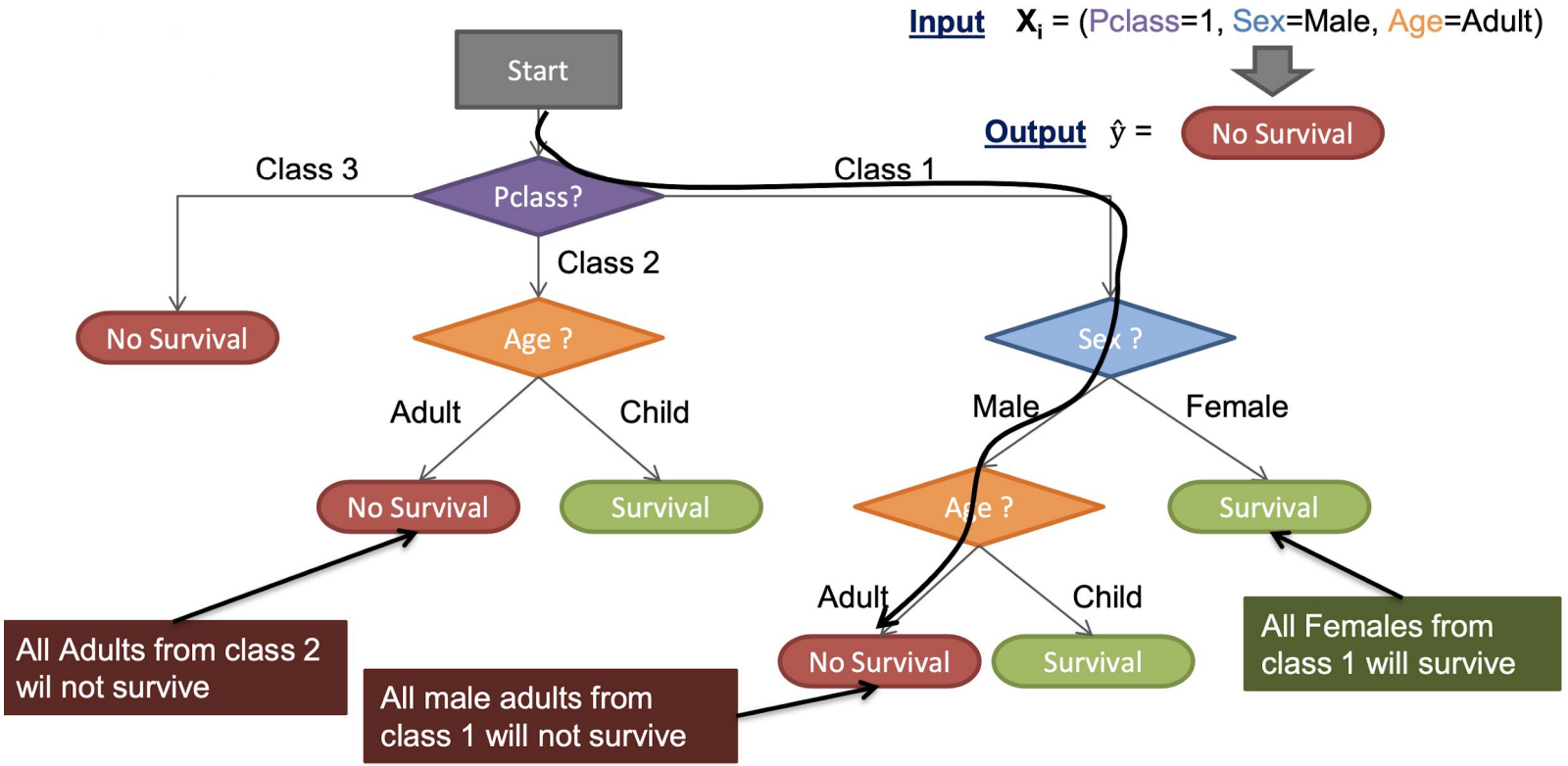
With n-features



With n-features



With n-features

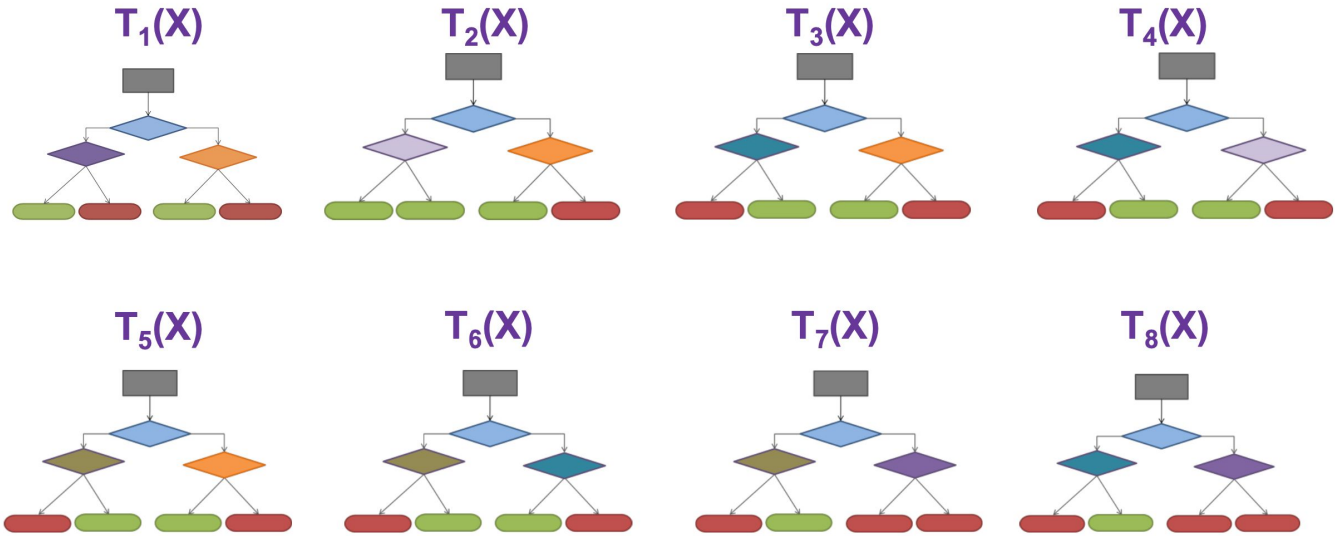


Multiple Decision Trees

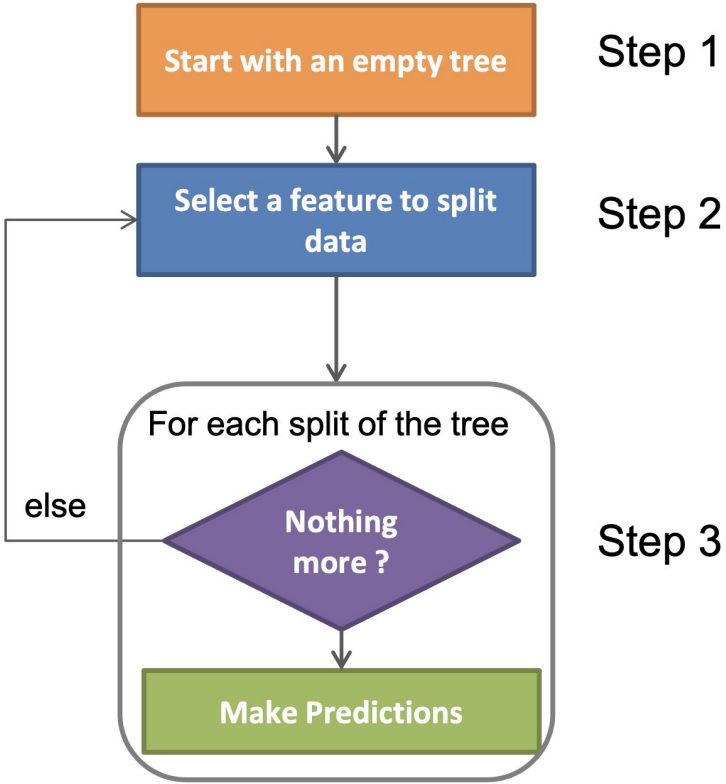
$$\text{ERROR} = \frac{\text{Nb of incorrect predictions}}{\text{Total Nb of samples}}$$

How to find the tree with lowest error ?

- Exponentially Large Number of possible trees ➔ making decision tree learning hard



Algorithm


Start with an empty tree
Step 1
Select a feature to split data
Step 2
For each split of the tree
else
Nothing more ?
Step 3
Make Predictions

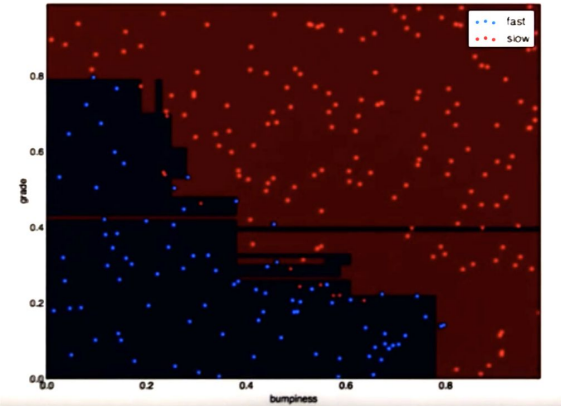
Coding a Decision Tree

class `sklearn.tree.`**`DecisionTreeClassifier`**(*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features=None, random_state=None, min_density=None, compute_importances=None, max_leaf_nodes=None*) ¶

```
from sklearn import tree
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)
```

Decision Tree Parameters

class sklearn.tree.**DecisionTreeClassifier**(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features=None, random_state=None, min_density=None, compute_importances=None, max_leaf_nodes=None) ¶

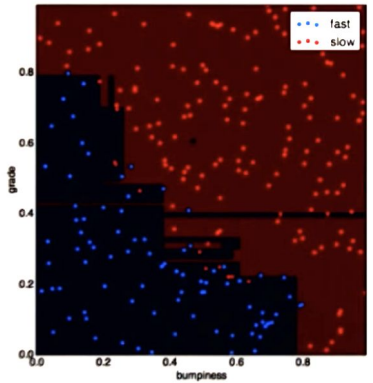


Min Samples Split

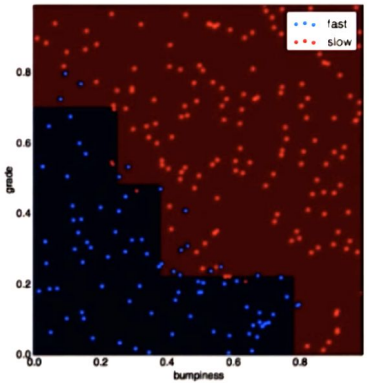
class `sklearn.tree.`**`DecisionTreeClassifier`**(*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features=None, random_state=None, min_density=None, compute_importances=None, max_leaf_nodes=None*) ¶

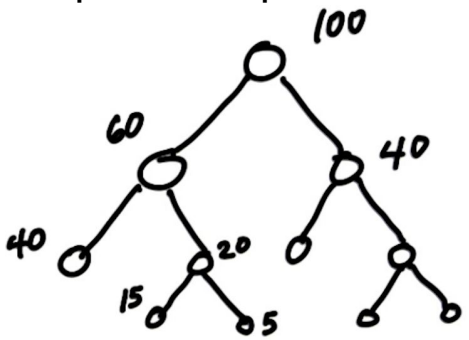
Controls whether you keep splitting or not. Number of elements required to split

2



50

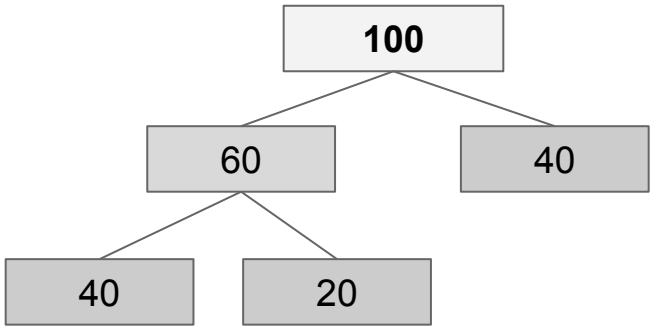




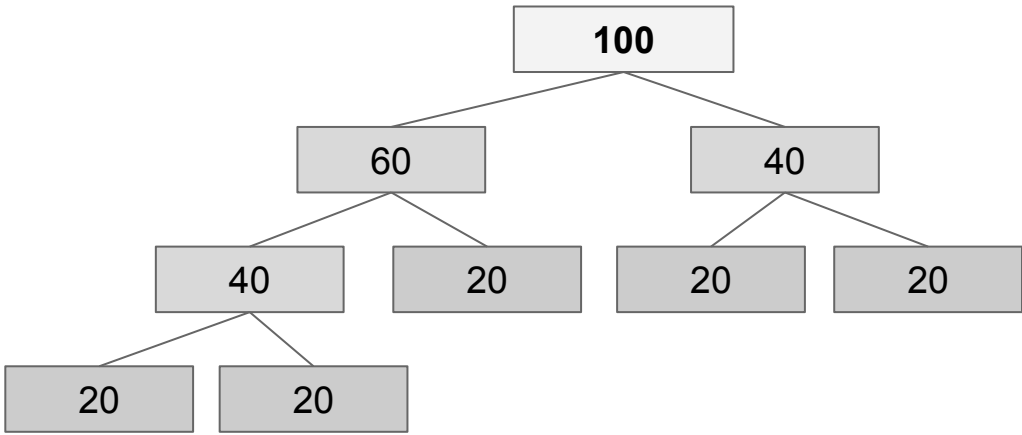
Min Samples Split

```
class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, max_features=None, random_state=None, min_density=None,
compute_importances=None, max_leaf_nodes=None) ¶
```

Min Samples Split = 50

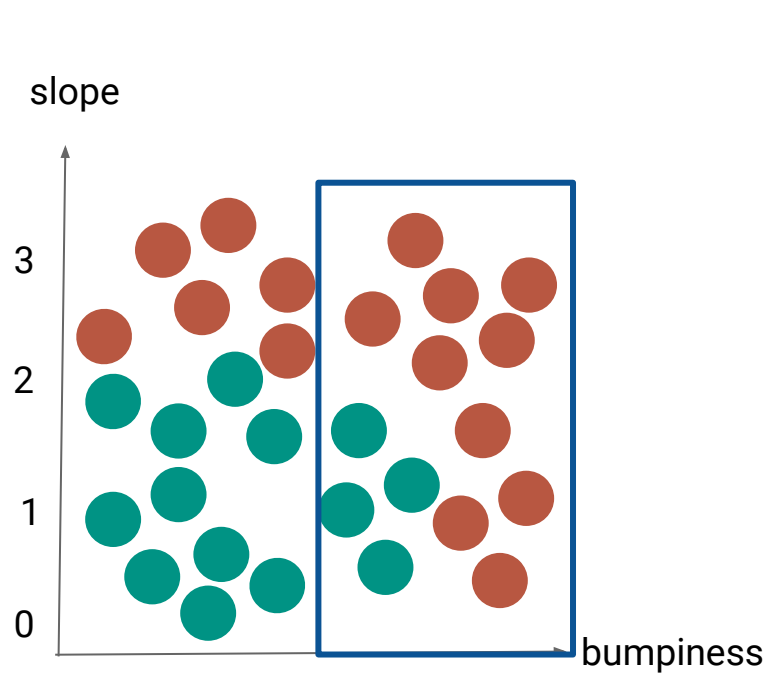


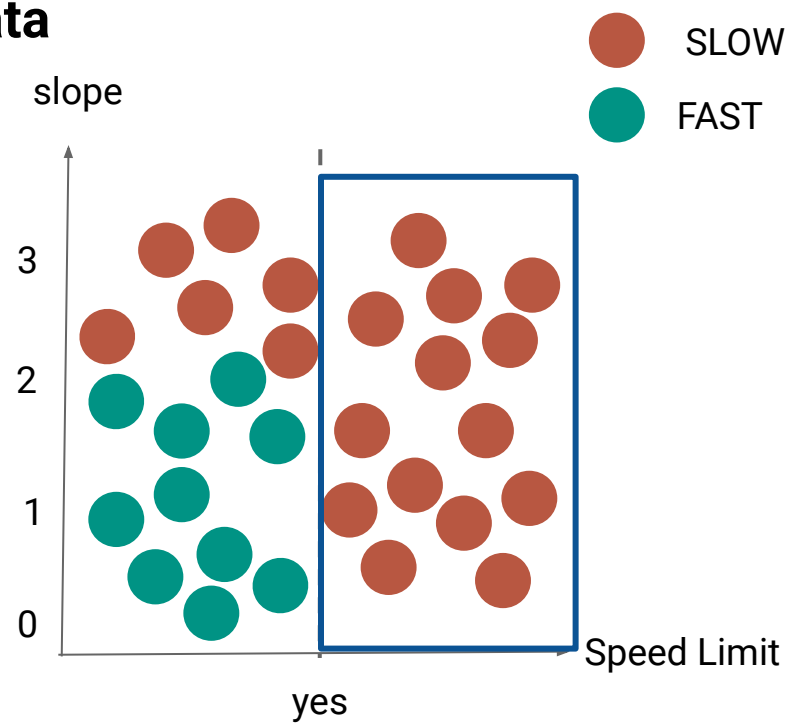
Min Samples Split = 21



Entropy

Measure of impurity in examples that controls where to split the data





Entropy

When splitting the data, we want the sample to be as pure as possible

$$\text{Entropy} = -\sum_i (p_i) \log_2 (p_i)$$

Pi is a fraction of examples for a given class

Entropy

$$\text{Entropy} = -\sum_i (p_i) \log_2 (p_i)$$

slope	bumpiness	speed limit	Speed target
steep	**high**	**yes**	**slow**
steep	**low**	**yes**	**slow**
flat	**high**	**no**	**fast**
steep	**low**	**no**	**fast**

We have 2 slow examples out of 4 total examples.

pi = fraction of **slow** examples

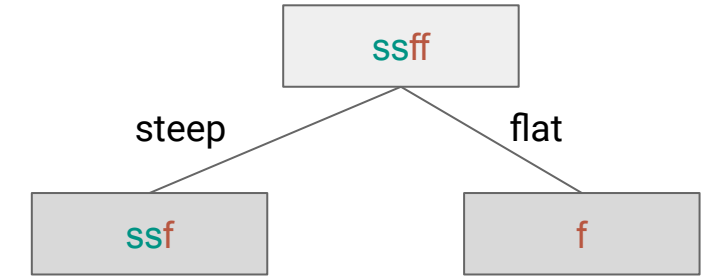
Pslow = 0.5
Pfast = 0.5

Entropy = 1.0
→ Entropy is maximum

Information Gain on Slope

information gain = entropy(parent) - [weighted_average]*entropy(children)

slope	bumpiness	speed limit	Speed target
steep	**high**	**yes**	**slow**
steep	**low**	**yes**	**slow**
flat	**high**	**no**	**fast**
steep	**low**	**no**	**fast**

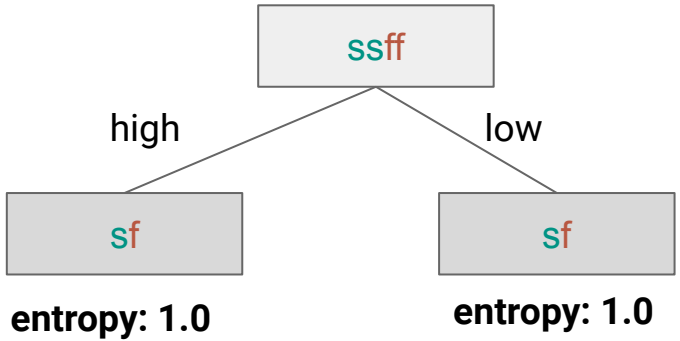


Information Gain = 0.3112

Information Gain on Bumpiness

information gain = entropy(parent) - [weighted_average]*entropy(children)

slope	bumpiness	speed limit	Speed target
steep	high	yes	slow
steep	low	yes	slow
flat	high	no	fast
steep	low	no	fast

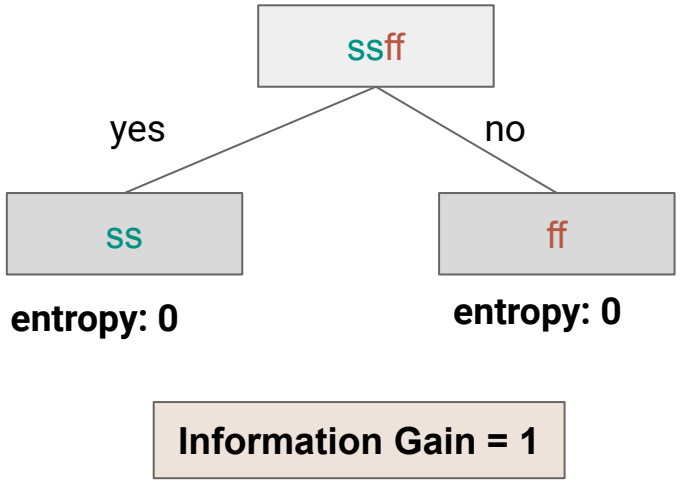


Information Gain = 0

Information Gain on Speed Limit

information gain = entropy(parent) - [weighted_average]*entropy(children)

slope	bumpiness	speed limit	Speed target
steep	high	yes	slow
steep	low	yes	slow
flat	high	no	fast
steep	low	no	fast



Where to Split?

slope	bumpiness	speed limit	Speed target
steep	**high**	**yes**	**slow**
steep	**low**	**yes**	**slow**
flat	**high**	**no**	**fast**
steep	**low**	**no**	**fast**

Information Gain (slope) = 0.3112

Information Gain (bumpiness) = 0

Information Gain (speed limit) = 1

What about continuous values?

It works too.

The decision Tree calculates values where to split (Age >21?) based on variance of the data and other criteria.

Coding the split

class `sklearn.tree.`**`DecisionTreeClassifier`**(*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features=None, random_state=None, min_density=None, compute_importances=None, max_leaf_nodes=None*) ¶

We can set criterion = Entropy, Gini or classification error

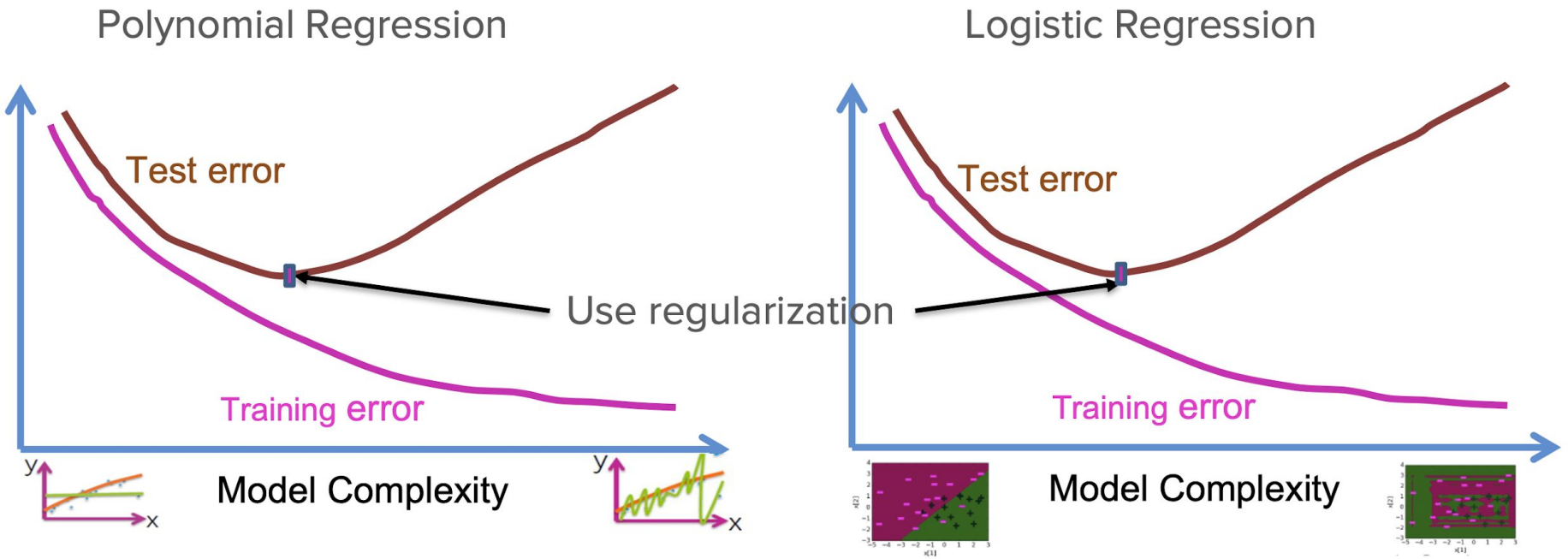
$$Classification\ error = 1 - \max(p, 1-p)$$

$$Gini\ Index = 2p(1-p)$$

$$Entropy = -plog(p) - (1-p)\log(1-p)$$

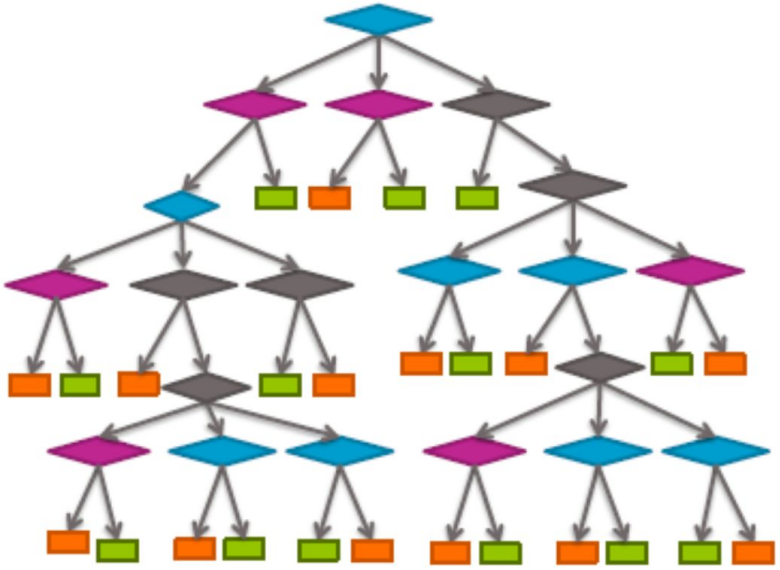
Overfitting

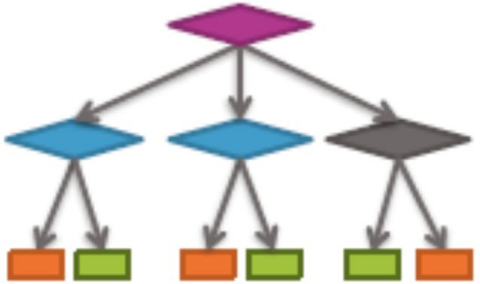
Overfitting



Overfitting

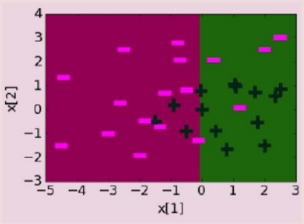
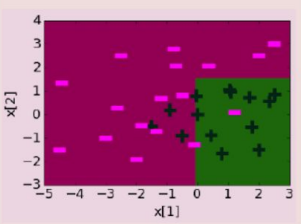
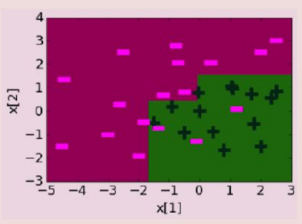
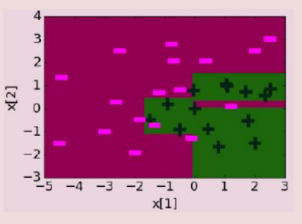
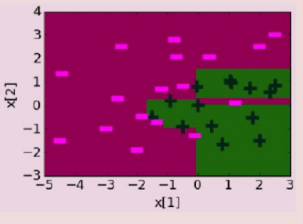
Tree depth





Simpler

Overfitting

Tree depth	depth = 1	depth = 2	depth = 3	depth = 5	depth = 10
Training error	0.23	0.13	0.1	0.033	**0.00**
Decision boundary					

- More depth = More complexity = Risk of overfitting

➔ Implement **Early Stopping** before the tree becomes too complex

Tuning Parameters

sklearn.tree.DecisionTreeClassifier

class sklearn.tree. **DecisionTreeClassifier** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_split=1e-07, class_weight=None, presort=False*) [source]

- Max_depth: The maximum depth of the tree
- min_samples_split: minimum number of samples required to split an internal node
- min_samples_leaf: Minimum number of samples required to be at a leaf node

Drawbacks

- **Prone to overfitting**
- **Find the balance between complex and simple tree**

Ensemble Methods: Bagging & Boosting

General Idea

Goal: Combine the predictions of several base estimators (ex. Decision trees) in order to improve generalizability / robustness over a single estimator

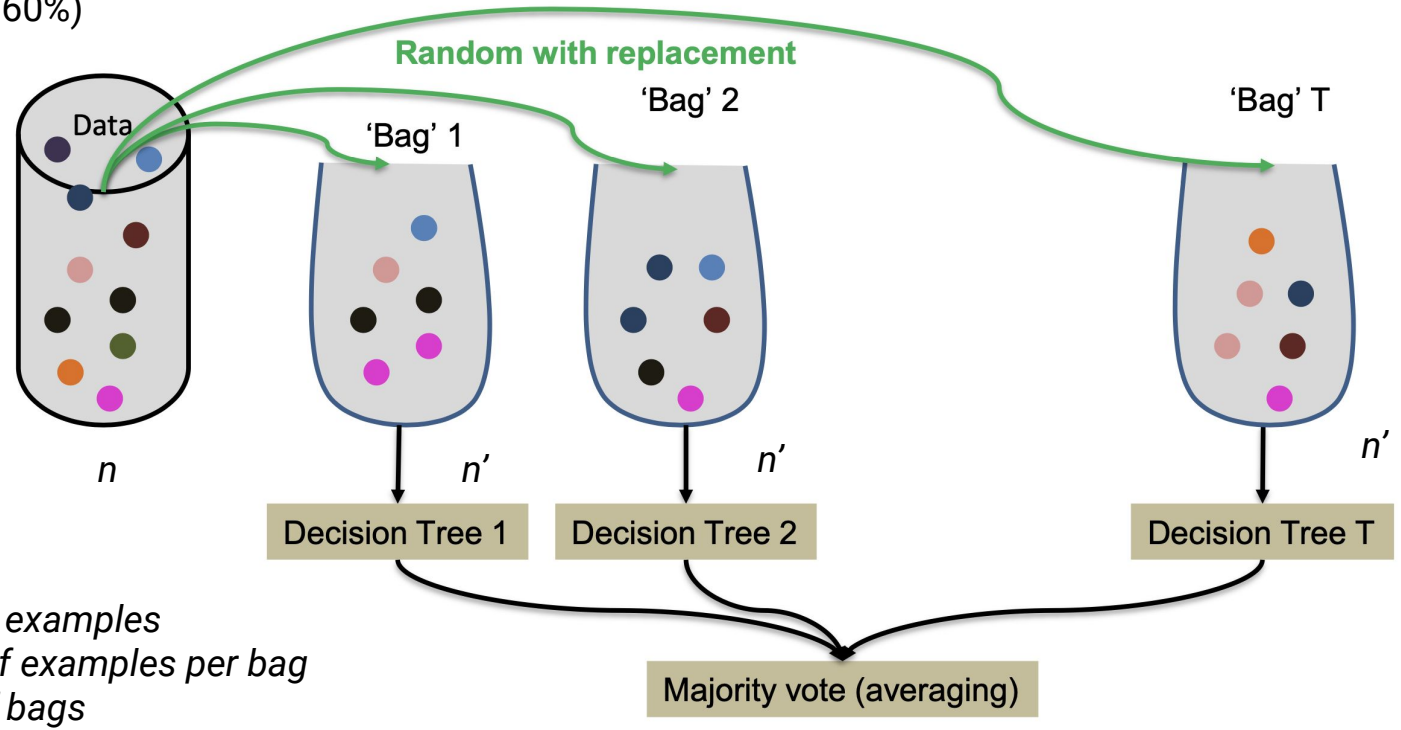
Bagging (Averaging methods): the driving principle is to build several estimators on different **subsets** of the data. Prediction proceeds with **majority vote** (averaging)

Boosting methods: base estimators are **built sequentially** and one tries to reduce the error of the previous one. Prediction proceeds with **weighted vote**.

Bagging: Combine multiple decision trees

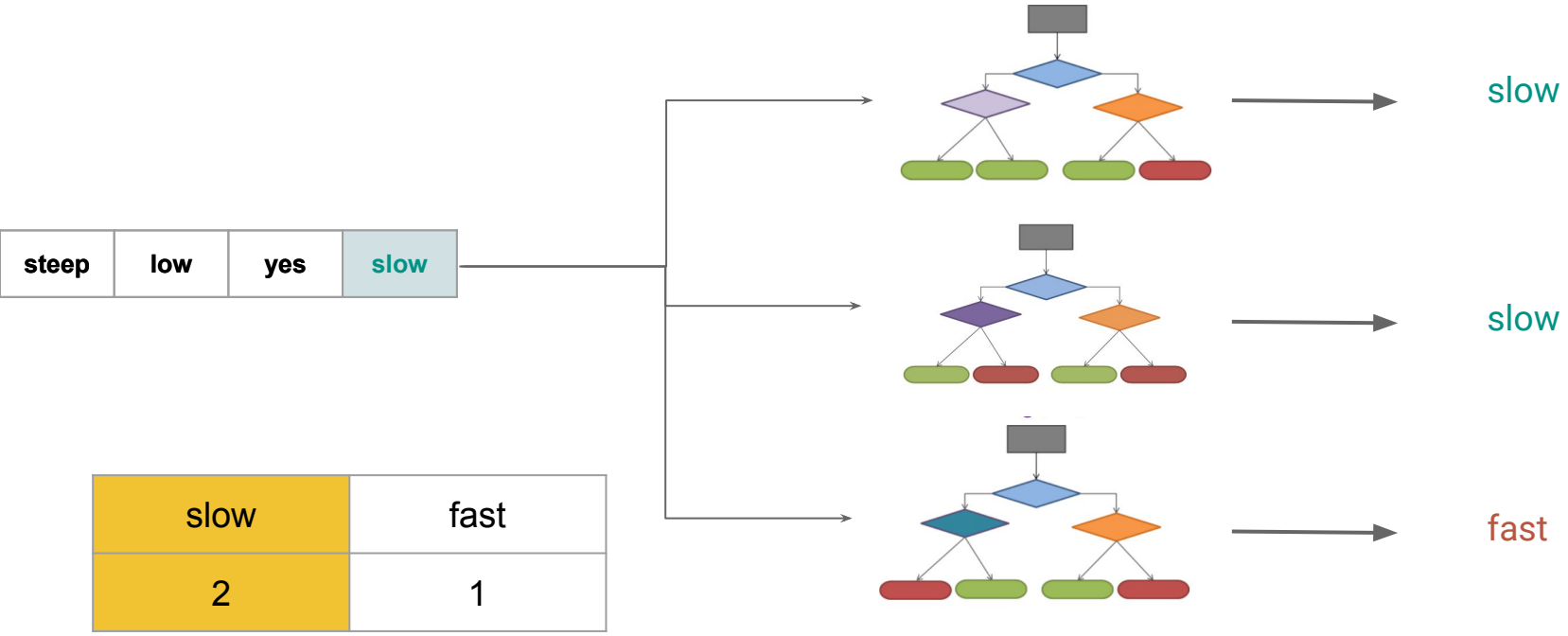
Bagging

n'<n (about 60%)



n: Number of examples
n' : Number of examples per bag
T: Number of bags

Bagging



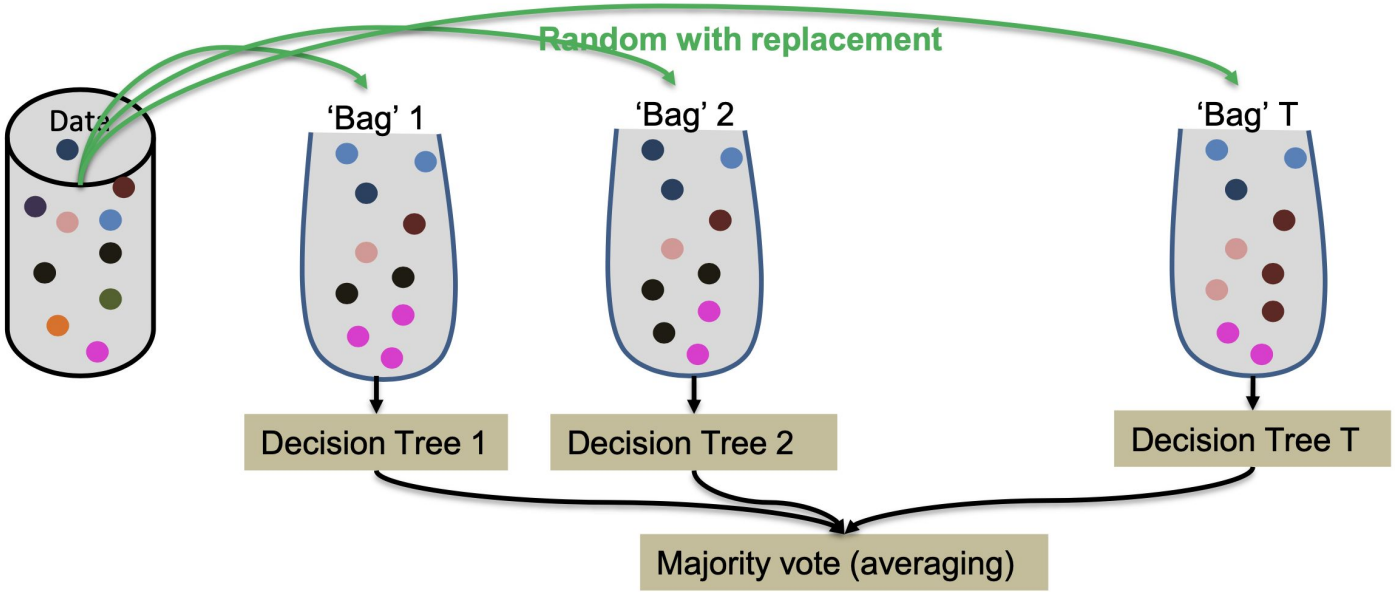
slow	fast
2	1

Random Forests

n'=n

Random forest is a special case of bagging where:

- The sub-sample size is always the same as the original input sample size
- When splitting, pick the best split among a random subset of the features.



Random Forests

1. Create a bootstrapped dataset
2. Build a Decision Tree with only a few variables to split selected at random
3. Aggregate the votes and take the final decision

Bootstrapped Aggregating = Bagging

Random Forests

slope	bumpiness	speed limit	Speed target
steep	high	yes	slow
steep	low	yes	slow
flat	high	no	fast
steep	low	no	fast

1. Create a bootstrapped dataset (n'=n)

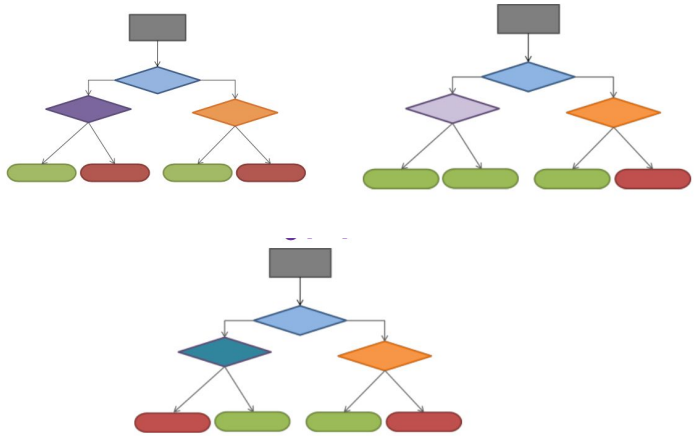
flat	high	no	fast
steep	high	yes	slow
steep	low	yes	slow
steep	low	yes	slow

Random Forests

slope	bumpiness	speed limit	Speed target
flat	**high**	**no**	**fast**
steep	**high**	**yes**	**slow**
steep	**low**	**yes**	**slow**
steep	**low**	**yes**	**slow**

2. Build hundreds of decision trees with only a few (e.g. 2) variable to split selected at random

Take the best information gain variable, split, ...

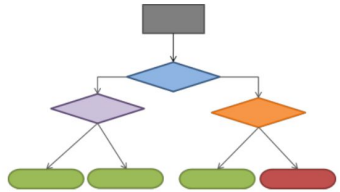


Random Forests

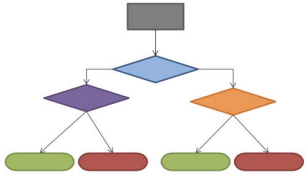
3. Aggregate the votes and take the final decision on a new data

steep	low	yes	slow

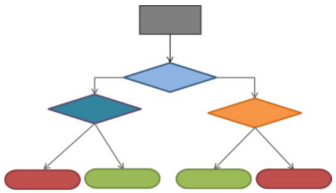
slow	fast
2	1



slow



slow

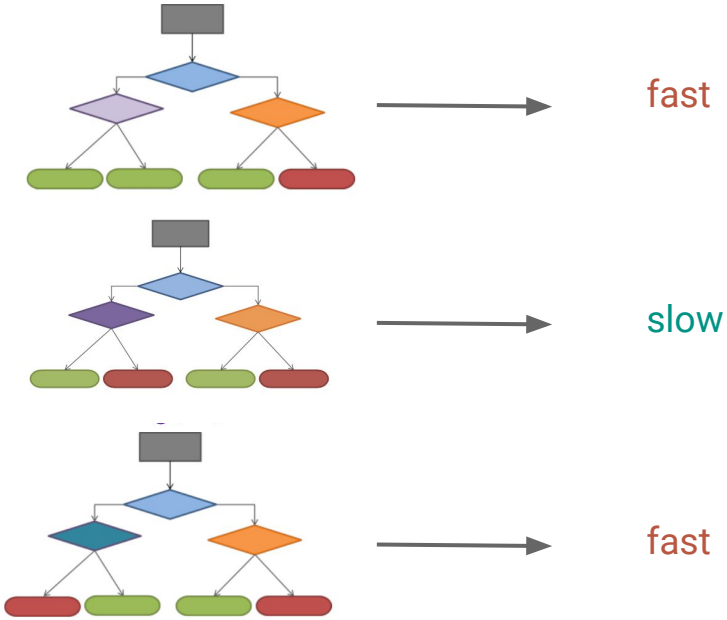


fast

Random Forests Accuracy

To calculate the accuracy, we use our “Out of Bag” data (About ⅓ of the full dataset) and use it as a test set by running it through the random forests

slope	bumpiness	speed limit	Speed target
steep	**high**	**yes**	**slow**
steep	**low**	**yes**	**slow**
flat	**high**	**no**	**fast**
steep	**low**	**no**	**fast**



Random Forests Accuracy

Calculate the accuracy and then rebuild the whole thing by changing the number of random variables where we split

Boosting: Turn a “weak” learning algorithm into a strong one

Adaboost

Adaboost is a boosting algorithm developed in 1999 by Freund & Schapire

Start same weight for all points: $\alpha^i = 1/m$

For t = 1,...,T

- Learn $h_t(x)$ with data weights α^i
- Compute $h_t(x)$'s coefficient w_t
- Update data weights α^i
- Normalize data weights α^i

$$w_t = \frac{1}{2}\ln\left(\frac{1 - weigted\ error(ht(x))}{weigted\ error(ht(x))}\right)$$

$$\alpha^i \leftarrow \begin{cases} \alpha^i * e^{-wt}, if ht(x^i) = yi \\ \alpha^i * e^{wt}, if ht(x^i) \neq y^i \end{cases}$$

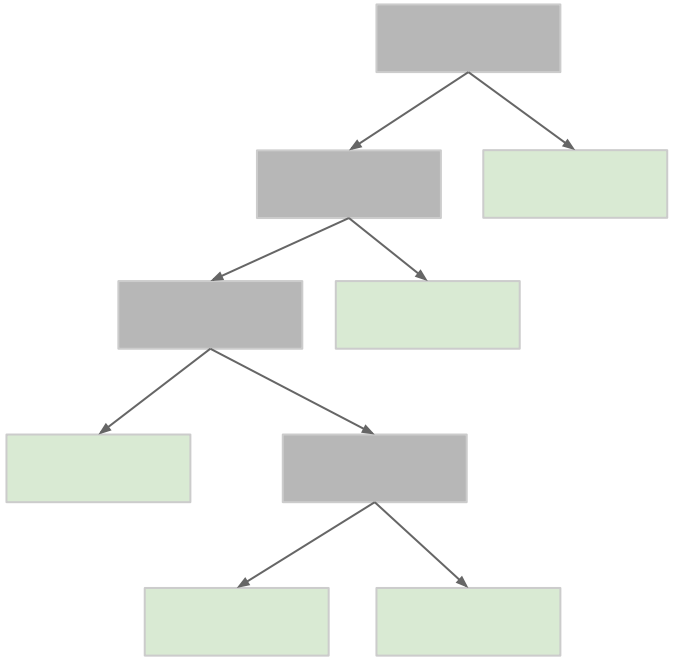
$$\alpha^i \leftarrow \frac{\alpha^i}{\sum_{j=1}^{m} \alpha^j}$$

Final model predicts by:

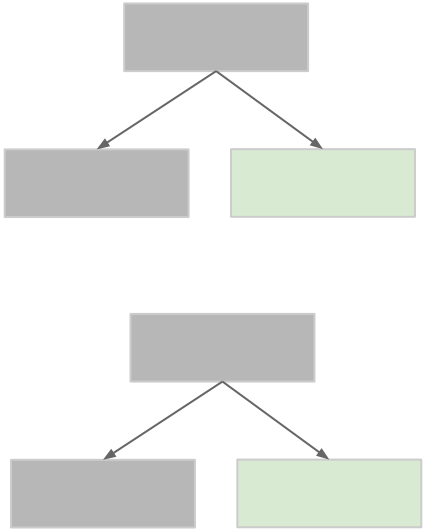
- $\hat{y} = \text{sign}\left(\sum_{t=1}^{T} w_t * ht(x)\right)$

Weak Learners / Stumps

Decision Tree

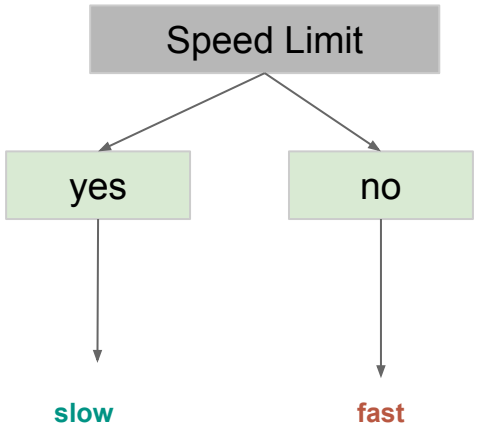


Stumps



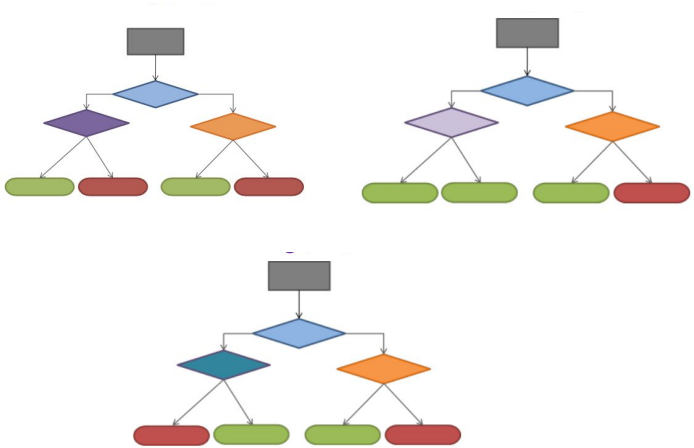
Weak Learners / Stumps

slope	bumpiness	speed limit	Speed target
steep	**high**	**yes**	**slow**
steep	**low**	**yes**	**slow**
flat	**high**	**no**	**fast**
steep	**low**	**no**	**fast**

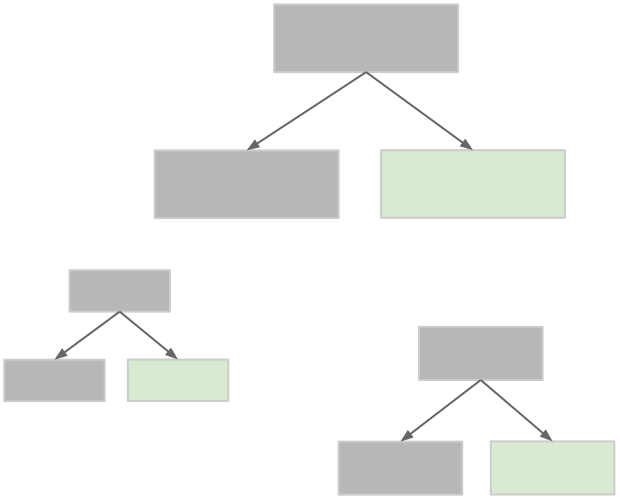


Random Forest vs. Adaboost

Each tree gets an equal vote

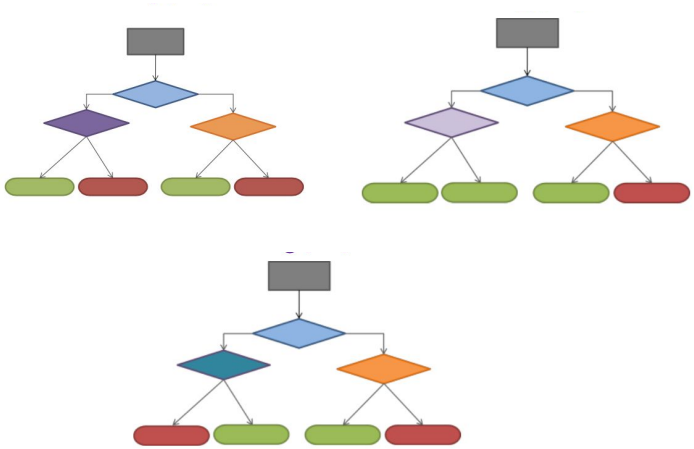


Each tree has a weight

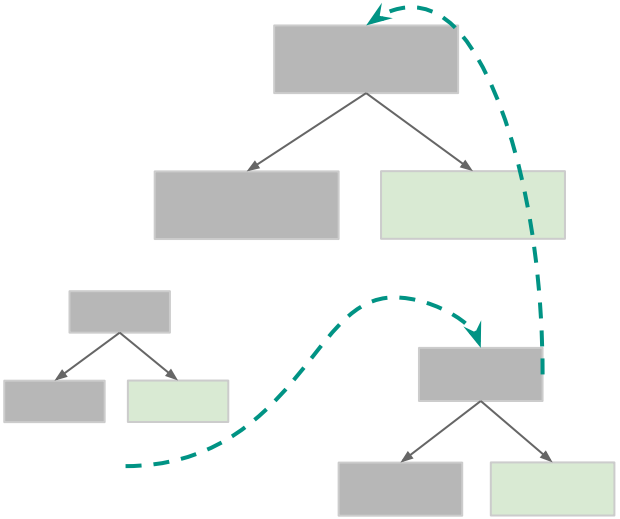


Random Forest vs. Adaboost

Each tree is independant

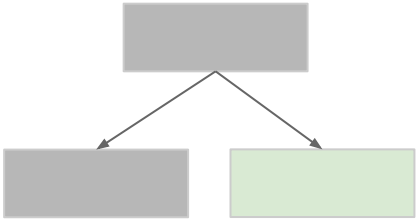


Trees are built based on the others

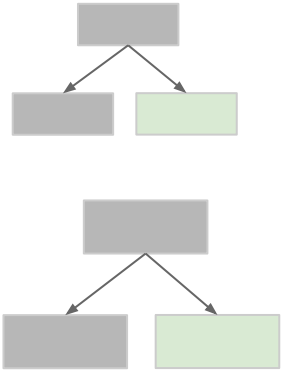


Adaboost Ideas

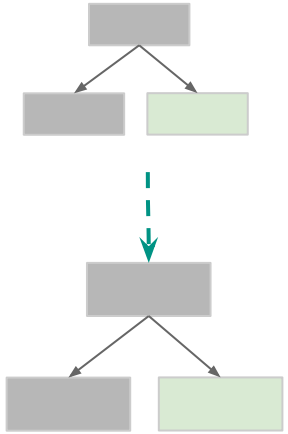
Adaboost combines a lot of weak learners. Almost all of them are stumps.



Some get more to say than others

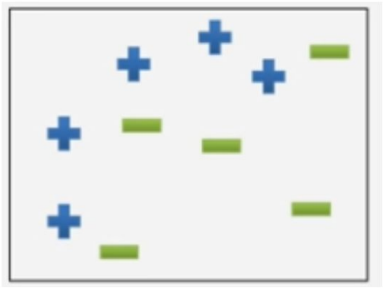


Each stump is made by taking the previous' mistakes into account

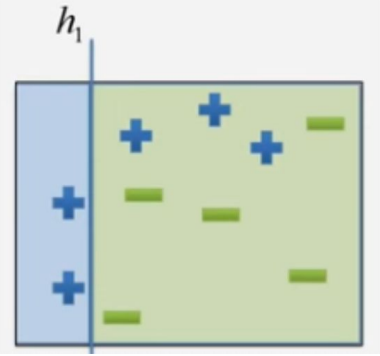


Adaboost

Our weak classifiers are only allowed to be lines that are either horizontal or vertical.



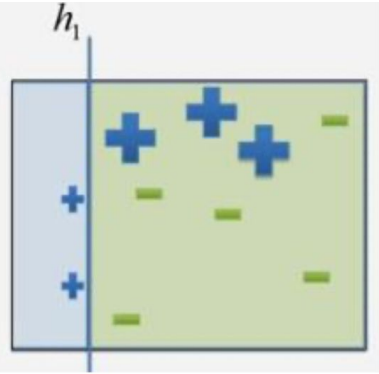
All data points start with equal weights



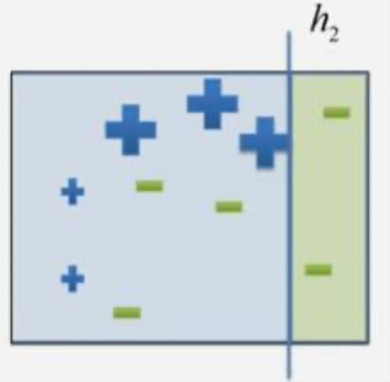
Run the weak learning algorithm, to get a weak classifier

Choose coefficient $w_1 = 0.41$

Adaboost



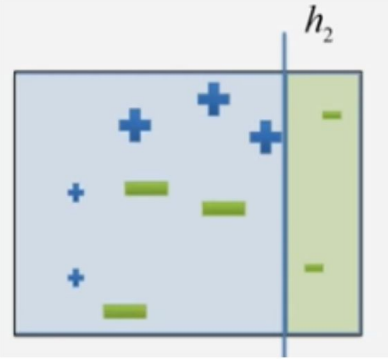
Increase the weights on the misclassified points.
Decrease the weights on the correctly classified points.



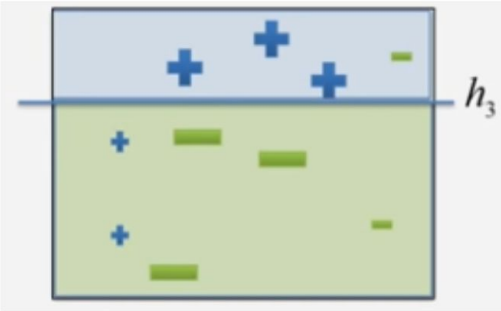
Run the weak learning algorithm, to get a weak classifier for the weighted data

Choose coefficient $w_2 = 0.66$

Adaboost



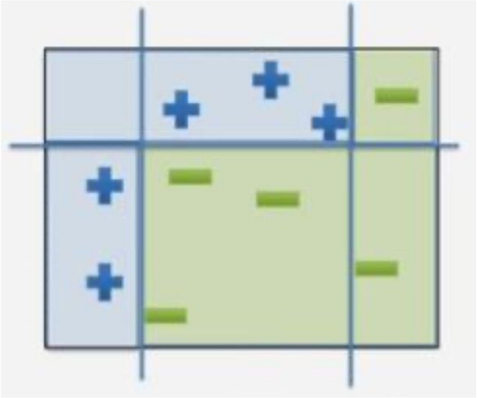
Increase the weights on the misclassified points.
Decrease the weights on the correctly classified points.



Run the weak learning algorithm, to get a weak classifier for the weighted data

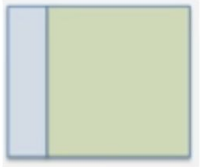
Choose coefficient $w_3 = 0.93$

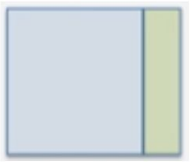
Boosting

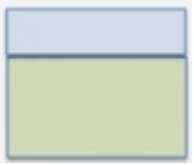


Combined classifier

$$\hat{y} = \text{sign}\,(0.41 * \quad +0.66 * \quad +0.93 * \quad)$$



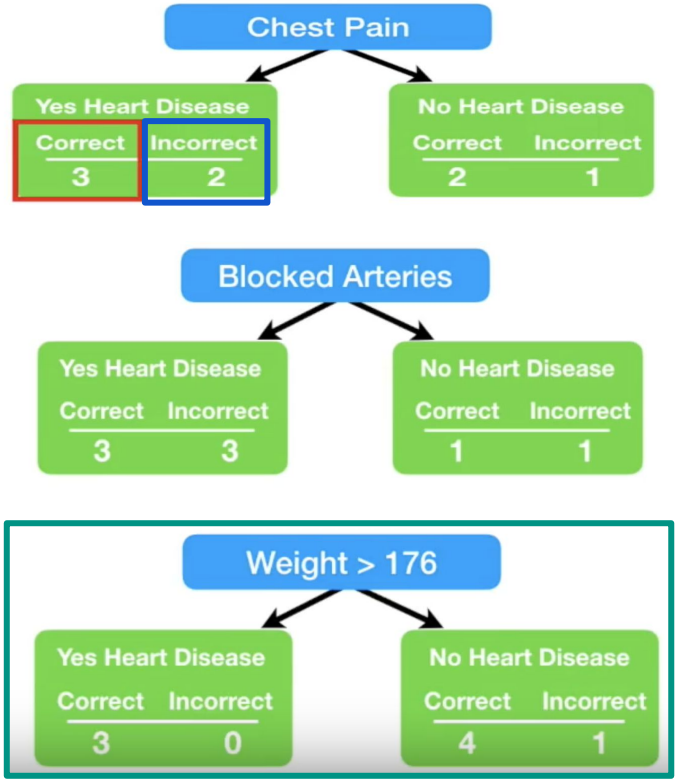




credits : StatQuest with Josh Starmer

Adaboost - Amount of Say

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8



purest feature

credits : StatQuest with Josh Starmer

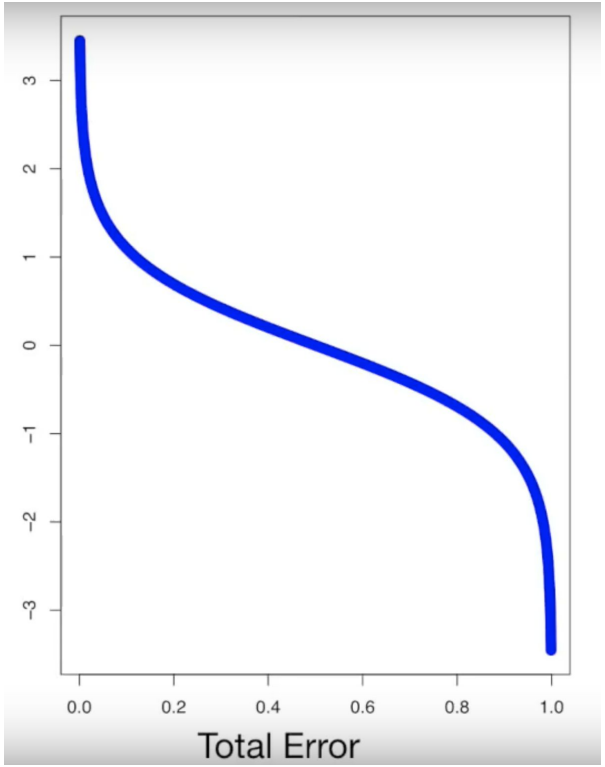
Adaboost - Amount of Say

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

The **Total Error** for a stump is the sum of the weights associated with the *incorrectly* classified samples.

credits : StatQuest with Josh Starmer

Adaboost - Amount of Say



$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$

Adaboost - Amount of Say

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	0.05
No	Yes	180	Yes	1/8	0.05
Yes	No	210	Yes	1/8	0.05
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	0.05
No	Yes	125	No	1/8	0.05
Yes	No	168	No	1/8	0.05
Yes	Yes	172	No	1/8	0.05

For misclassified points

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

For correctly classified points

$$\text{New Sample Weight} = \text{sample weight} \times e^{-\text{amount of say}}$$

credits : StatQuest with Josh Starmer

Adaboost - Amount of Say

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight	Norm. Weight
Yes	Yes	205	Yes	1/8	0.05	0.07
No	Yes	180	Yes	1/8	0.05	0.07
Yes	No	210	Yes	1/8	0.05	0.07
Yes	Yes	167	Yes	1/8	0.33	0.49
No	Yes	156	No	1/8	0.05	0.07
No	Yes	125	No	1/8	0.05	0.07
Yes	No	168	No	1/8	0.05	0.07
Yes	Yes	172	No	1/8	0.05	0.07

credits : StatQuest with Josh Starmer

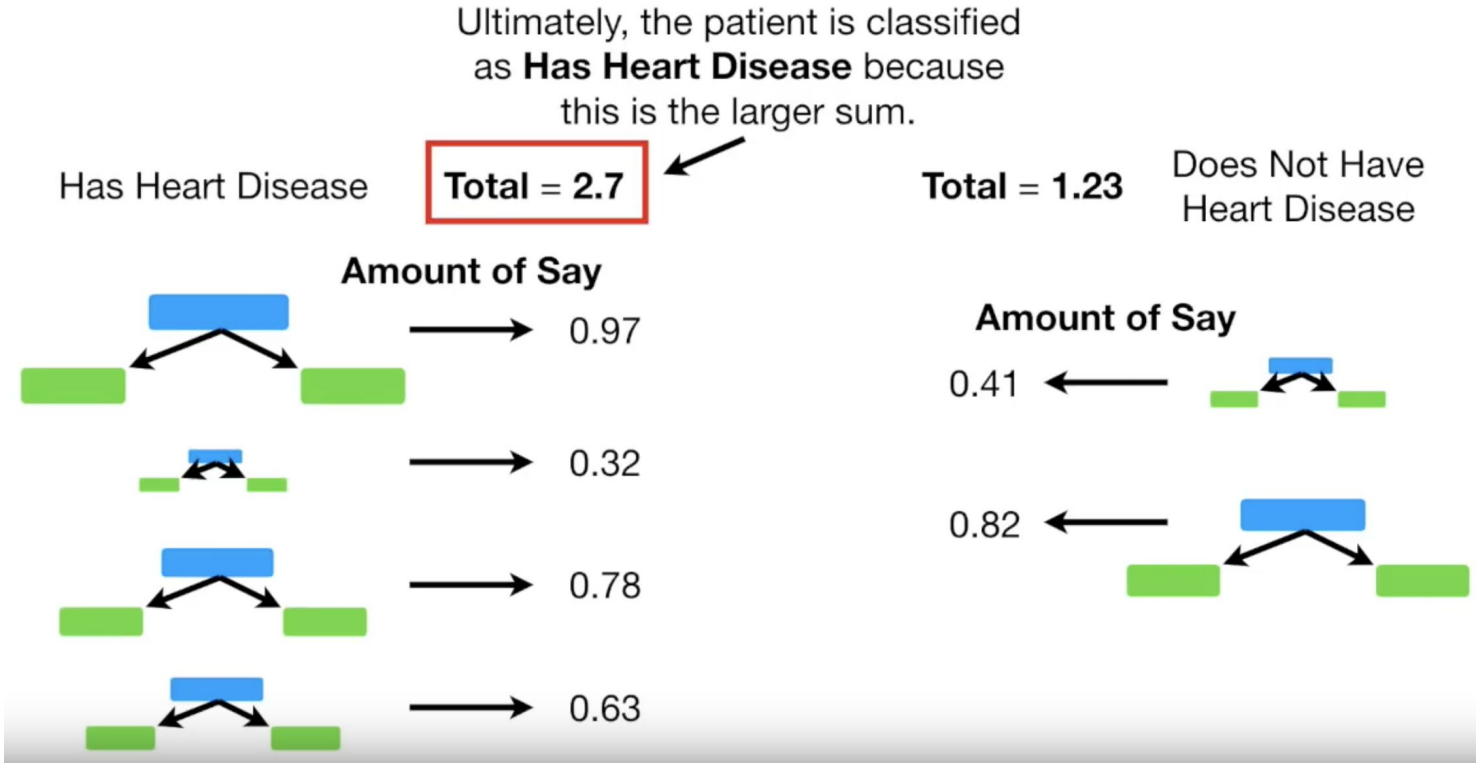
Adaboost - Amount of Say

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

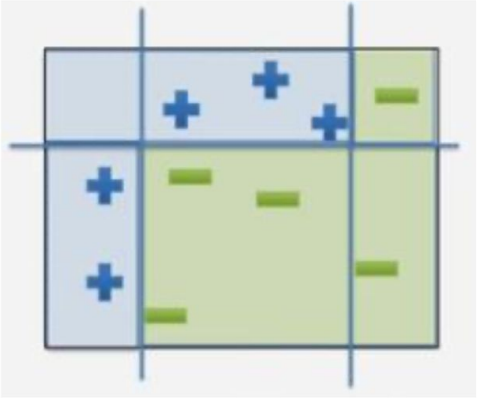
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	205	Yes	1/8
Yes	Yes	167	Yes	1/8

credits : StatQuest with Josh Starmer

Adaboost - Amount of Say

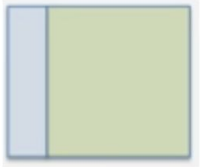


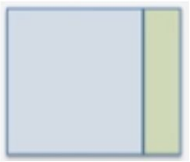
Boosting

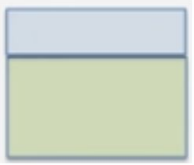


Combined classifier

$$\hat{y} = \text{sign}\,(0.41 * \quad +0.66 * \quad +0.93 * \quad)$$







Thank You

jeremycohen.podia.com

https://www.linkedin.com/in/jeremycohen2626/