Data Science

Yogesh Kulkarni

Decision Tree

Lets play a game

Game

Remember '20 questions' game?

- Some one holds name of one famous person in mind
- Others need to find out that by asking, at max, 20 questions.
- Questions should be such that the answers to which are only Yes or No.
- No queries, no descriptive answers, nothing else.

Game

What are we doing here, essentially?

- Eliminating subgroups.
- So, for question 'Male or Female', any answer will remove half of the population from our search space.
- Each of these questions are actually splitting the search space one by one.
- Making it narrower and narrower.
- Till we get to one person.

Game

Its a tree.

- Starting at root
- Each question at node
- Answer at leaves
- Whats the max depth of the tree allowed in the game we just discussed?

Game

Whats the success?

- Idea is to have minimum depth ie minimum number of questions
- So, your questions should be such that you get to the answer quicker.

This is the decision tree, and the questions or the splits decides its success.

Tree? - Comp Science way

- What is a Tree?
- Comp Science definition?
- What are other Data Structures?
- Diff between Tree and Graph?

Decision Tree Example - Admissions

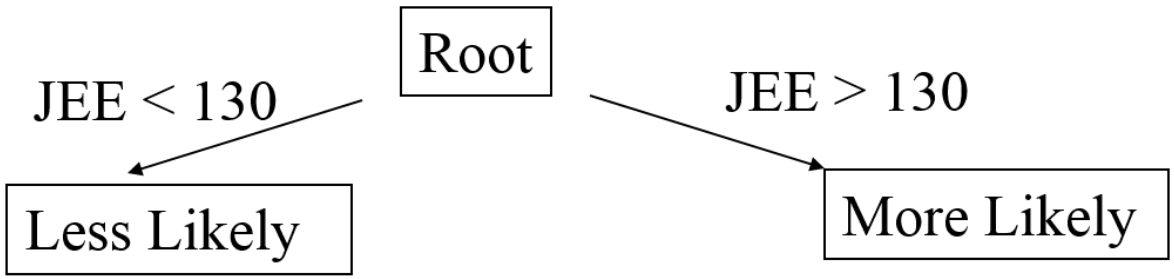
Admissions

- Admission to a reputed college depends, say, on JEE score as well as 12th std marks.
- Previous data of 1000 applicants shows that average JEE score for admitted students is 150 and average 12th std marks are 80%.
- We are asked to create a model that will allow us to predict students most likely to be admitted.
- How do we go about doing this?

Approach

- Divide applicants into subgroups: those with more than, say, like 130 JEE, are in 'more likely' group
- Draw the tree

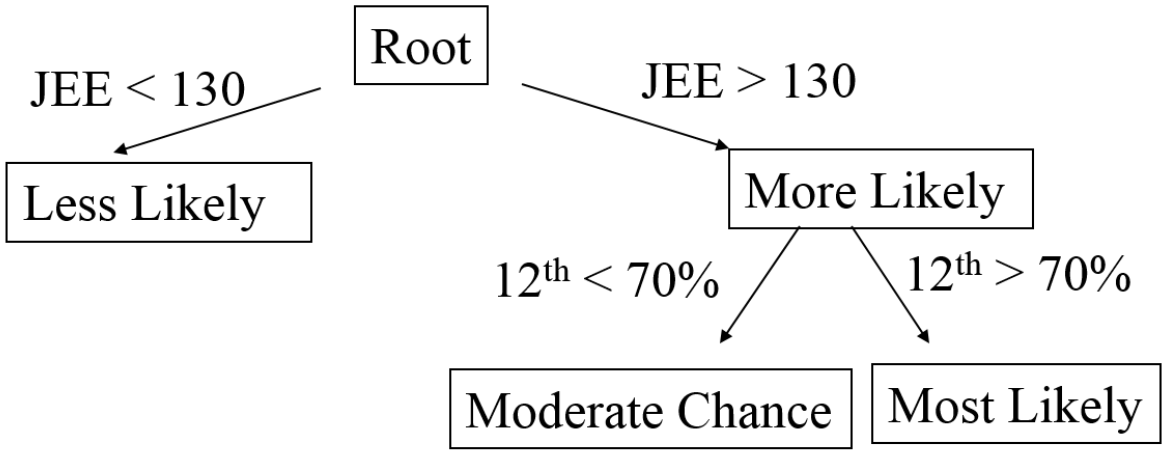
Approach



Approach

- Split this group itself based 12th percentage, above 70% (called "most likely") or below (called "high maybe").
- Add to the tree

Approach


Root
JEE < 130
JEE > 130
Less Likely
More Likely
12th < 70%
12th > 70%
Moderate Chance
Most Likely

Approach

- Then we do the same thing to the group with JEE score below 130, calling these subgroups as "Less Likely".
- Add to the tree and finish till the leaves.

Approach

- We thus formed a Decision Tree.
- We can now predict outcome of any unknown case.

Questions

- What if we split on 12th marks first and then JEE scores - would we get the same groupings?
- What is the best way to split?

Questions

- What if we used more criteria such as essay evaluation scores, extra curricular, awards and distinctions in sports etc. i.e. how many attributes should we use to create splits and what are the most significant attributes.

All these criteria can be modeled using Decision Tree technique.

Decision Tree in Machine Learning

Decision Trees

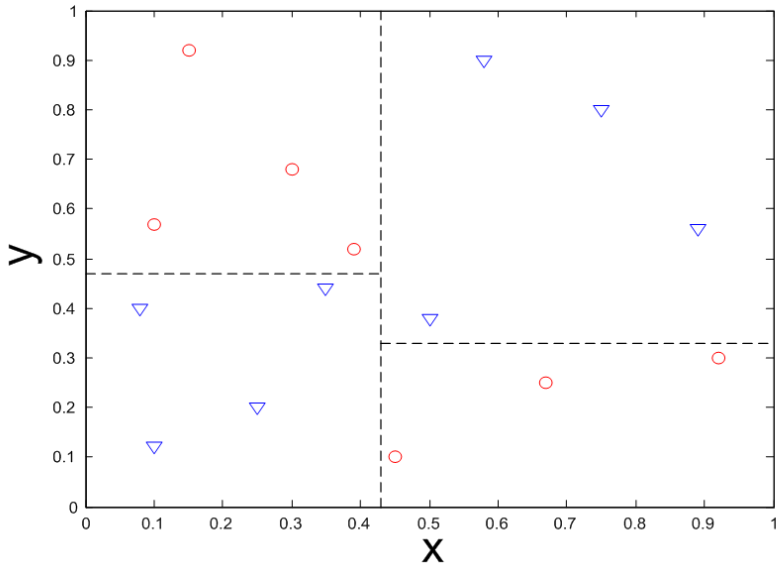
- Decision Trees (DTs) are a non-linear supervised learning method used for classification and regression.
- The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

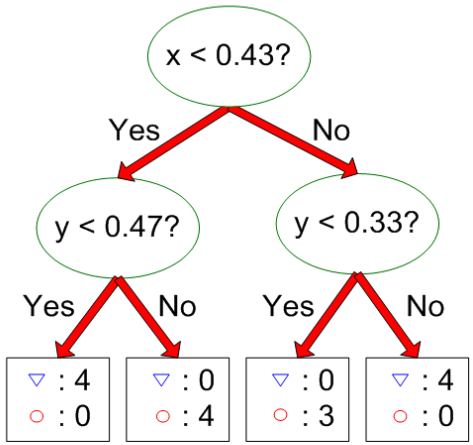
Decision Trees

Can you rewrite the previous decision trees as a If-then-else statement?

Graphically

Decision Boundaries





- Border line between two neighboring regions of different classes is known as decision boundary
- Decision boundary is parallel to axes because test condition involves a single attribute at-a-time

Decision Trees

Simple, yet widely used classification technique

- For categorical target variables. There also are Regression trees, for continuous target variables
- Predictor Features: binary, nominal, ordinal, discrete, continuous.
- Evaluating the model: One metric: error rate in predictions

Decision Trees Structure

- **A root node** has no incoming edges and zero or more outgoing edges
- **Internal nodes**, each of which has exactly one incoming edge and two or more outgoing edges. An internal node is a new question.
- **Leaf/terminal nodes** , each of which has exactly one incoming edge and no outgoing edges. Is a class label. If a leaf node is reached, you got the conclusion.

Decision Trees

- Were really popular 2 decades ago, but fail out of fashion due to over fitting
- Then came newer versions like C4.5, C5 to address some of the issues
- Rather than a single tree, it was found that collection of trees addresses this variance/over-fitting problem
- Thus, now we have Random Forest and Boosting, to make Trees as sought after solution.

Decision Tree for Loan Default

Decision Trees

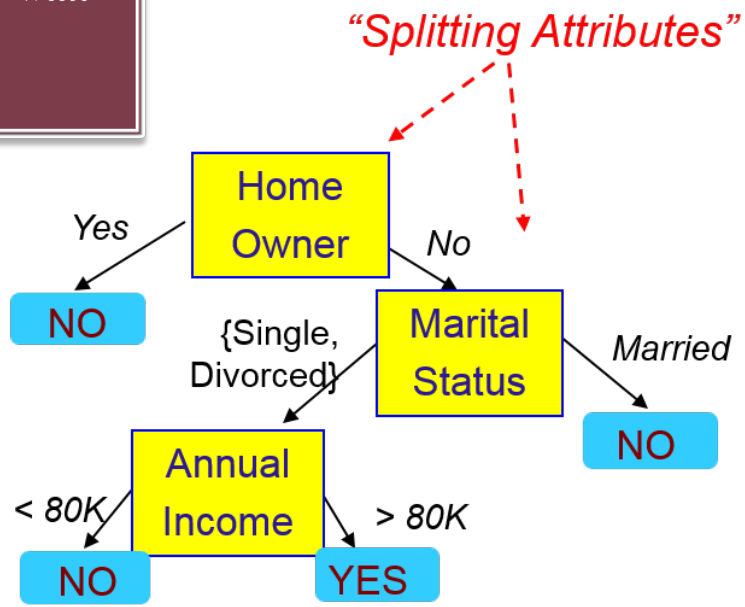
binary categorical continuous class

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data

Tree is consistent with training dataset.

1. Root node
2. Internal nodes
3. Leaf / terminal nodes



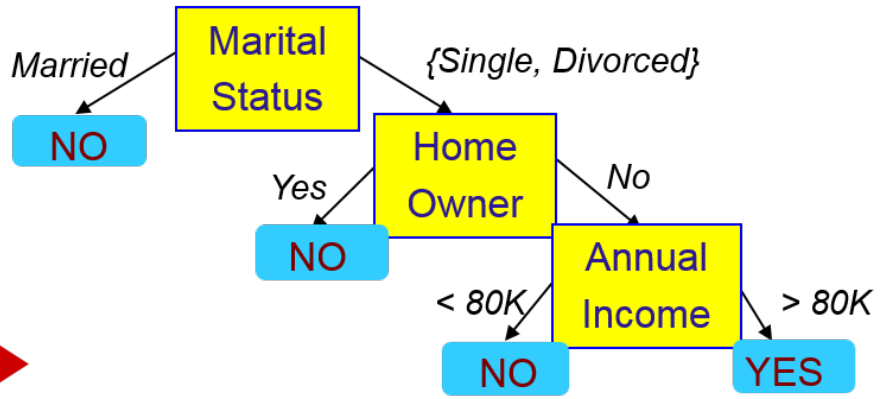
Decision Tree Model #1

Decision Trees

	binary	categorical	continuous	class
Tid	**Home Owner**	**Marital Status**	**Annual Income**	**Defaulted Borrower**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data

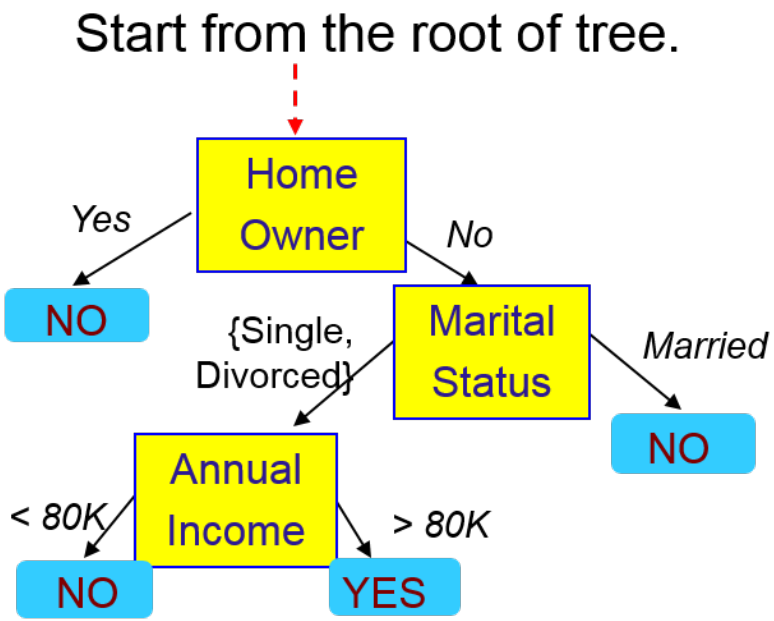
Tree is consistent with training dataset.



Decision Tree Model #2

There could be more than one tree that fits the same data!

Apply Model to Test Data



Test Data

Home Owner	Marital Status	Annual Income	Defaulted Barrower
No	Married	80K	?

Decision Tree for 'Will John Play?'

Predict if John will play tennis

- We have records of outdoors condition and whether John played on those days or not.
- That's the training data.
- We wish to predict for certain UNSEEN condition, whether John will play or not.

Training examples: **9 yes / 5 no**

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

(Ref: Decision Trees - Victor Lavrenko)

Predict if John will play tennis

Steps:

- Split at columns
- Are they pure? (all yes or all no)
- If yes: stop; if not: repeat

Lets start with first column" "Outlook"

Predict if John will play tennis

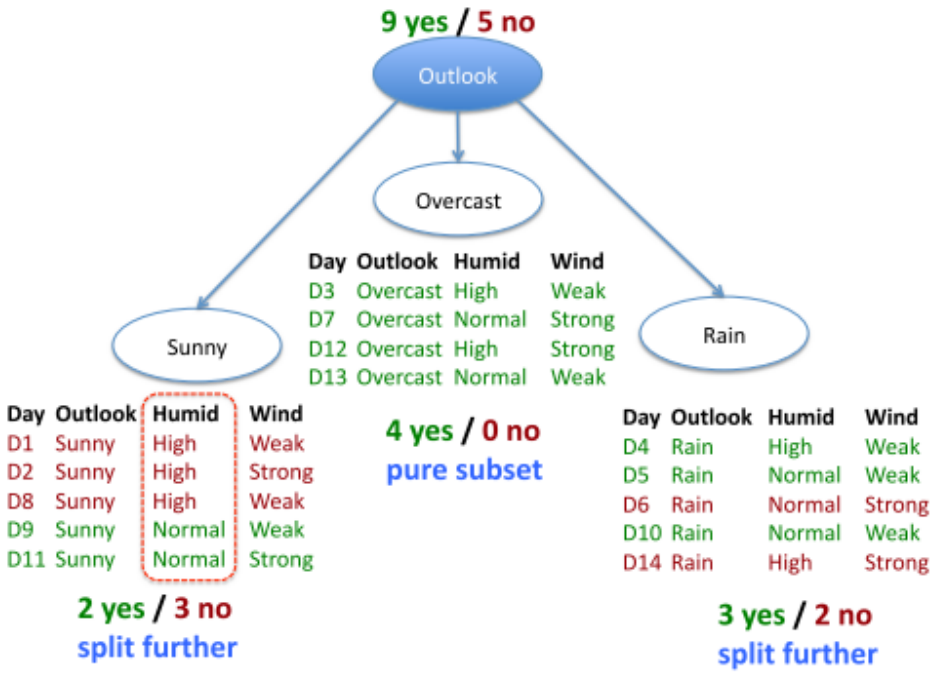
Training examples: **9 yes / 5 no**

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

New data:

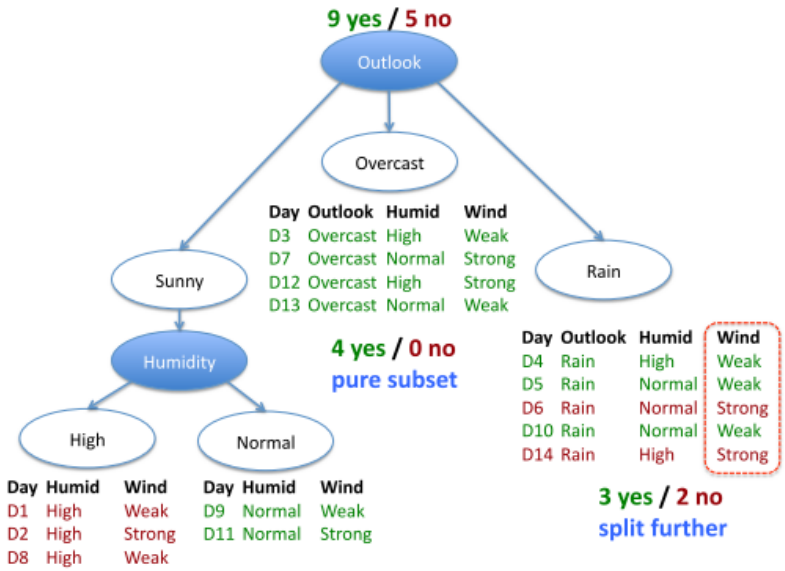
D15	Rain	High	Weak	?

Predict if John will play tennis



- Outlook-Overcast turned Pure , in the first split itself!!
- Humidity and Wind are irrelevant for this middle node now.
- Cannot expand this middle node further.
- Go for the other nodes: Outlook-Sunny and Outlook-Rain

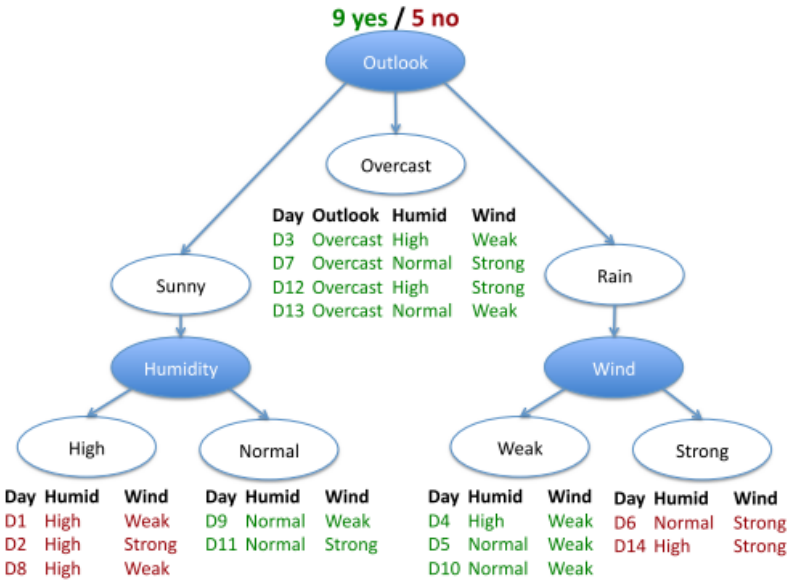
Predict if John will play tennis



- Outlook-Sunny after splitting on Humidity turned Pure
- Need Outlook-Rain to split now.

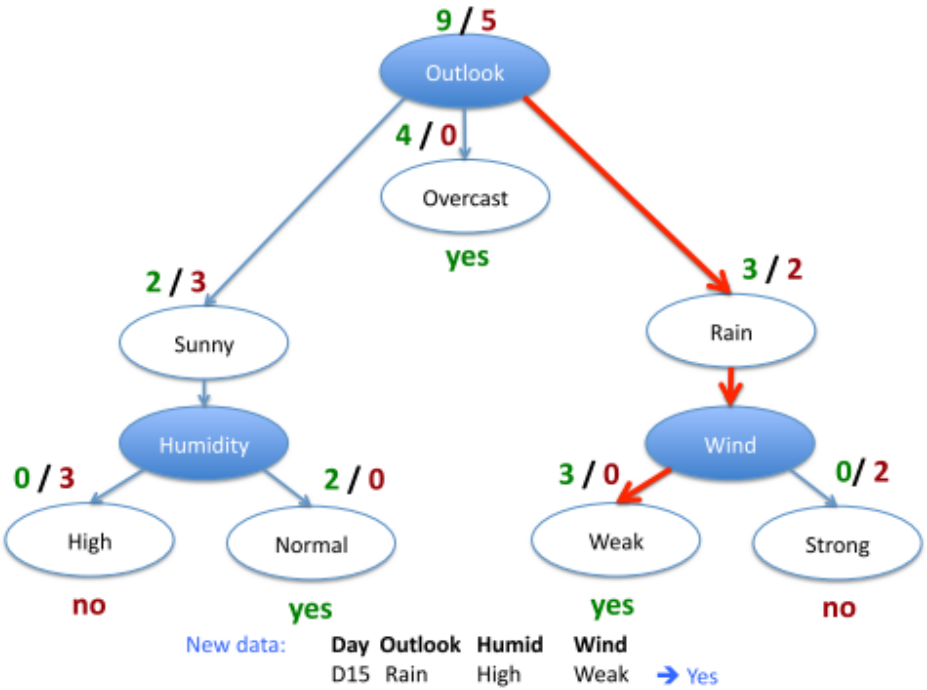
Predict if John will play tennis

- Rain after splitting on Wind turned Pure.
- All leaves are Pure now.
- That's the end of splitting



Predict if John will play tennis

- Decision tree result can be seen below.
- To test the unseen condition, traverse the tree
- for Day 15, with Outlook-Rain, Humid-High and Wind-Weak, the outcome is "Yes". John will play.



Decision Tree Classifier

- Decision tree models are relatively more descriptive than other types of classifier models
  - Easier interpretation
  - Easier to explain predicted values
- Exponentially many decision trees can be built
- Which is best? Some trees will be more accurate than others
- How to construct the tree? Computationally infeasible to try every possible tree.

Decision Tree Inventor

- The late Leo Breiman was instrumental in developing several tree-based methods.
- Worked on the problem of classifying ships from radar signals.
- He covered the walls of his office with data from the radars prior to his "aha" moment on Decision Trees.

(Something like this!!!)

See: "Leo Breiman (CART)" video on Youtube

Building the Decision Tree

Formally

- A decision tree has three types of nodes:
  - A root node that has no incoming edges and zero or mode outgoing edges
  - Internal nodes, each of which has exactly one incoming edge and two or more outgoing edges
  - Leaf nodes (or terminal nodes), each of which has exactly one incoming edge and no outgoing edges
- Each leaf node is assigned a class label
- Non-terminal nodes contain attribute test conditions to separate records that have different characteristics

What is the "Best Attribute" to split

There are different criteria that can be used to determine what is the best attribute to split.

- Information Gain
- Gini Index
- ...

Which attribute to split on

- Want to measure PURITY of the split
- Are we more certain about Yes/No after the split?
  - pure set (4 yes/ 0 no) then completely certain (100%)
  - impure set (3 yes/ 3 no) then completely uncertain (50%)
- Must be symmetric: 4 yes / 0 no as pure s 0 yes/ 4 no.

Entropy

- High 'mixed'-ness, high uncertainty thus higher "entropy"
- Say if all are identical members, then no uncertainty and thus zero "entropy".

Entropy

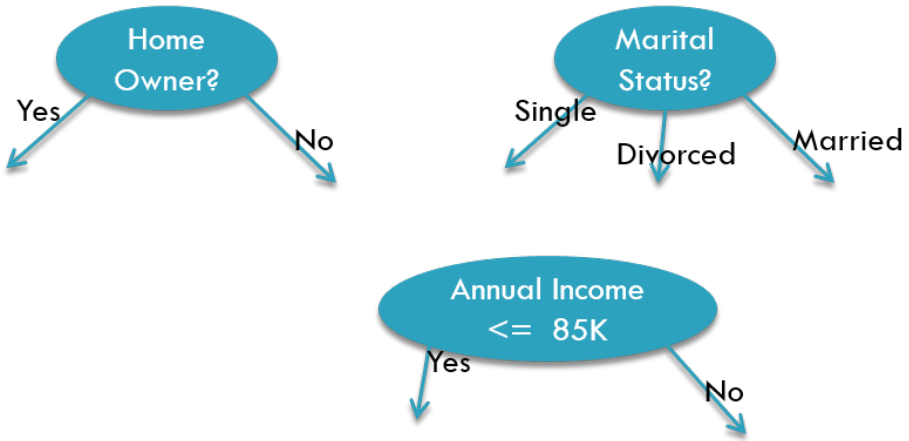
- So, when we split a set we want the halves to be more distinct from each other and the members in the group to be more like each other
- We want entropy to go down as we keep splitting.
- So if we use an approach/split that doesn't reduce the entropy by much, then it is probably not a good attribute or a good value to split on.
- Shannon's entropy is defined for a system with N possible states as $S = -\sum_{i=1}^{N} p_i \log_2 p_i$ where p_i is the probability of finding the system in the i-th state.
- N designates number of classes. For binary classification $N = 2$.
- Shannon's formula is part of information theory, given minimum number of letters to encode a message.

Entropy

- Originally from Second Law of Thermodynamics, Entropy is a measure of disorder (uncertainty, chaos).
- In the current context, its measure of disorder in the target-label.
- Entropy can be described as the degree of chaos in the system.
- The higher the entropy, the less ordered the system and vice versa.
- Information Gain is a measure of decrease in Entropy by partitioning the data-set.

How to determine the Best Split?

How does entropy help us?



Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

How to calculate Information Gain using Entropy?

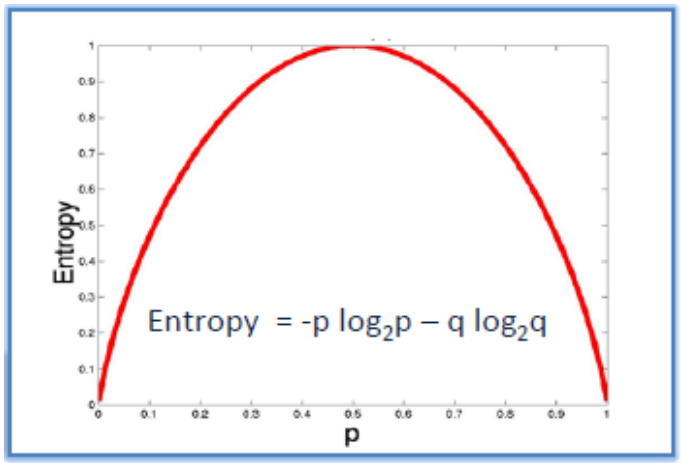
Information Gain

- Want many items in pure sets.
- Try to split on each possible feature in a dataset. See which split works "best".
- Measure the reduction in the overall entropy of a set of instances.

$$InformationGain = Entropy(s) - \sum \frac{s_i}{s} E(s_i)$$

Entropy

- ID3 algorithm uses entropy to calculate the homogeneity of a sample.
- If the sample is completely homogeneous the entropy is zero
- If the sample is equally divided then it has entropy of one.
- By Binary outcome, $Entropy(n) = -p_{(+)}log_2 p_{(+)} - p_{(-)}log_2 p_{(-)}$
- $Entropy(n) = -0.5log_2 0.5 - 0.5log_2 0.5 = 1$

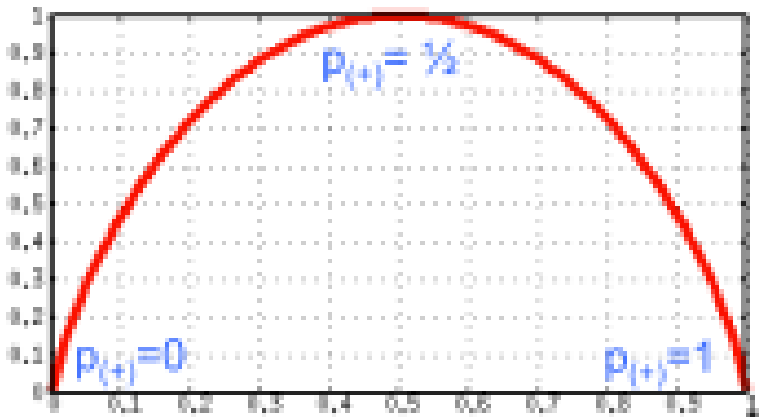


Entropy = -0.5 $\log_2$0.5 – 0.5 $\log_2$0.5 = 1

Entropy

In our original example:

- Impure (3 yes/ 3 no): $Entropy(n) = -3/6log_2 3/6 - 3/6log_2 3/6 = 1$
- Pure (4 yes/ 0 no): $Entropy(n) = -4/4log_2 4/4 - 0/4log_2 0/4 = 0$
- So, Entropy also ranges from 0 to 1, in following fashion

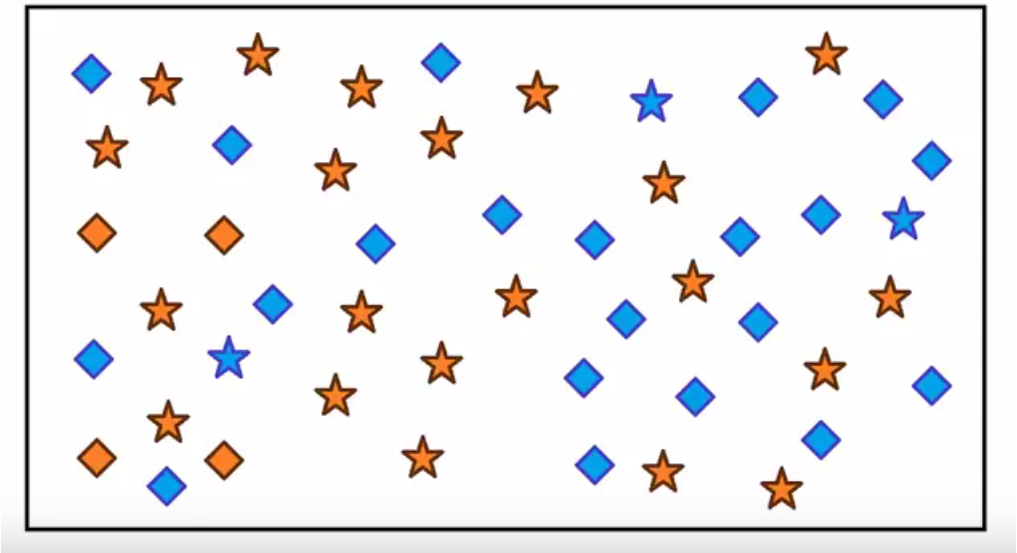


Entropy

- $Entropy(n) = -\sum_i^c p_i log_2 p_i$
- Weighted sum of the logs of the probabilities of each of the possible outcomes
- Entropy of the set of 52 playing cards: Randomly selecting any specific card i is $1/52$. $Entropy(n) = -\sum_1^{52} 0.019 log_2 0.019 = 5.7$
- Entropy of the set of 4 suits: Randomly selecting any specific card i is $1/4$. $Entropy(n) = -\sum_1^4 0.25 log_2 0.25 = 2$
- The higher the impurity, the higher the entropy.
- That's the Reason for Using the log function

Information Gain Exercise of Stars/Diamonds, Blue/Orange

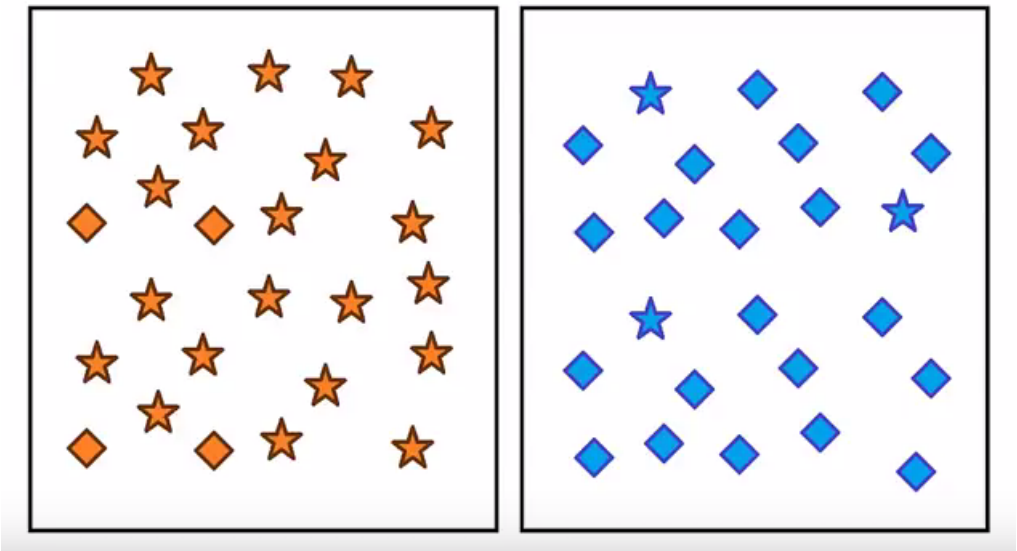
Entropy Example



- There are stars and diamonds, 24 stars, 25 diamonds
- Input: Color
- Output: Shape

Entropy Example

- Predicting: Star vs Diamond
- Split on: Color
- Orange box and blue box.
- With this, HAS it becomes easier to predict Star or Diamond?
- Lesser chance of mis-classification.
- Less disorder, more information gain.



Entropy Example

- Full group : 24 stars, 25 diamonds
- 25 orange object: 21 stars, 4 diamonds
- 24 blue object: 3 stars, 21 diamonds
- $p_{stars} = 24/49$ and $p_{diamonds} = 25/49$

Full Group:

$$
\begin{aligned}
E(\vec{d}) &= -\sum_i p_i \log_2(p_i) \\
&= -(p_1 \log_2(p_1) + p_2 \log_2(p_2)) \\
&= -\left(\frac{24}{49}\log_2\left(\frac{24}{49}\right) + \frac{25}{49}\log_2\left(\frac{25}{49}\right)\right) \\
&\cong 0.9997
\end{aligned}
$$

This is Full Group, Original Entropy. Need to reduce it by partitioning.

Entropy Example

- $p_{diamonds,orange} = 4/25$ and $p_{stars,orange} = 21/25$
- $p_{diamonds,blue} = 21/24$ and $p_{stars,blue} = 3/24$

Orange Group:

$$E(\vec{d}, orange) = -\sum_i p_i \log_2(p_i)$$
$$= -(p_1 \log_2(p_1) + p_2 \log_2(p_2))$$
$$= -\left(\frac{4}{25}\log_2\left(\frac{4}{25}\right) + \frac{21}{25}\log_2\left(\frac{21}{25}\right)\right)$$
$$\cong 0.6343$$

Blue Group:

$$E(\vec{d}, blue) = -\sum_i p_i \log_2(p_i)$$
$$= -(p_1 \log_2(p_1) + p_2 \log_2(p_2))$$
$$= -\left(\frac{21}{24}\log_2\left(\frac{21}{24}\right) + \frac{3}{24}\log_2\left(\frac{3}{24}\right)\right)$$
$$\cong 0.5436$$

Entropy Example

- Original Full Group Entropy (without partitioning) was : 0.9997
- Weighted sum after partitioning on Color

Combined Entropy:

$$E(\vec{d}) = 0.9997$$

$$E(\vec{d}, \vec{a}) = \frac{25}{49} E(orange) + \frac{24}{49} E(blue)$$

$$= \frac{25}{49}(0.6343) + \frac{24}{49}(0.5436) = 0.5899$$

Information Gain:

$$I(\vec{d}, \vec{a}) = E(\vec{d}) - E(\vec{d}, \vec{a}) = 0.9997 - 0.5899 = 0.4097$$

Substantial gain.

GINI Index

(Ref: Decision Tree. It begins here - Rishabh Jain)

Gini Index

Gini index says, if we select two items from a population at random then they must be of same class and probability for this is 1 if population is pure.

- Works with categorical target variable
- Performs only Binary splits
- Higher the value of Gini higher the homogeneity
- CART (Classification and Regression Tree) uses Gini method to create binary splits

Gini Index Steps

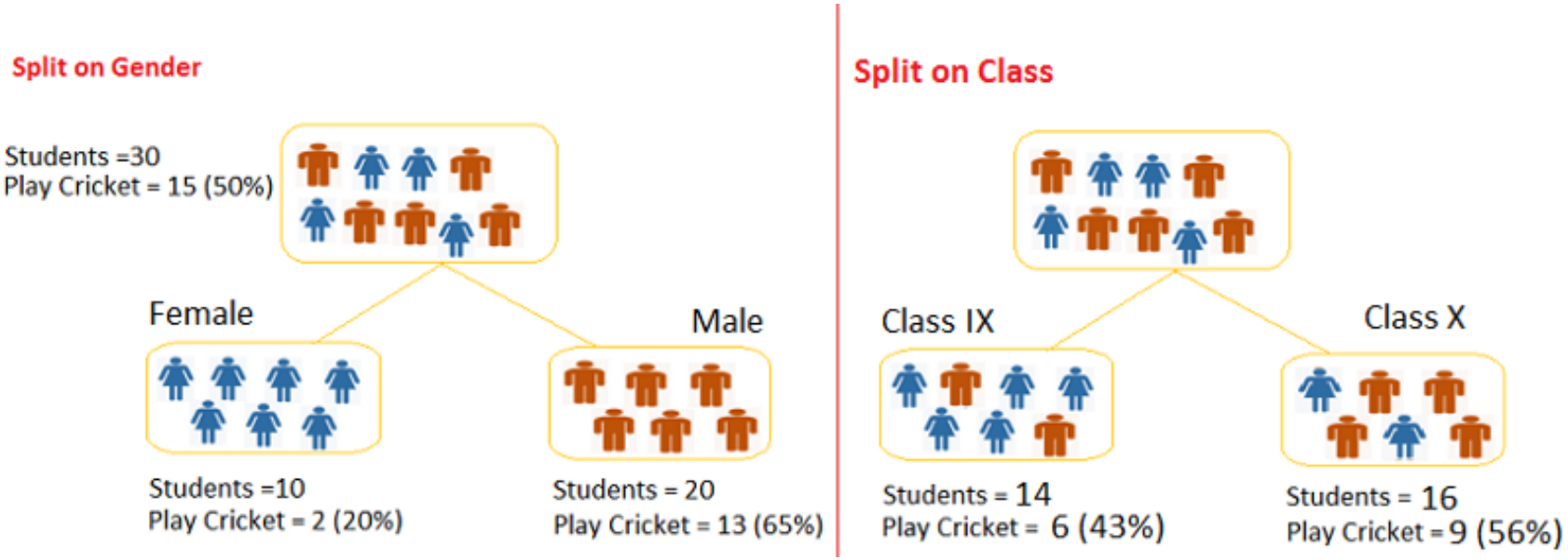
Steps to Calculate Gini for a split

- Calculate Gini for sub-nodes, using formula sum of square of probability for success and failure $(p^2 + q^2)$
- Calculate Gini for split using weighted Gini score of each node of that split

Example

- We want to segregate the students based on target variable (playing cricket or not)
- We split the population using two input variables Gender and Class.
- Now, I want to identify which split is producing more homogeneous sub-nodes using Gini index.

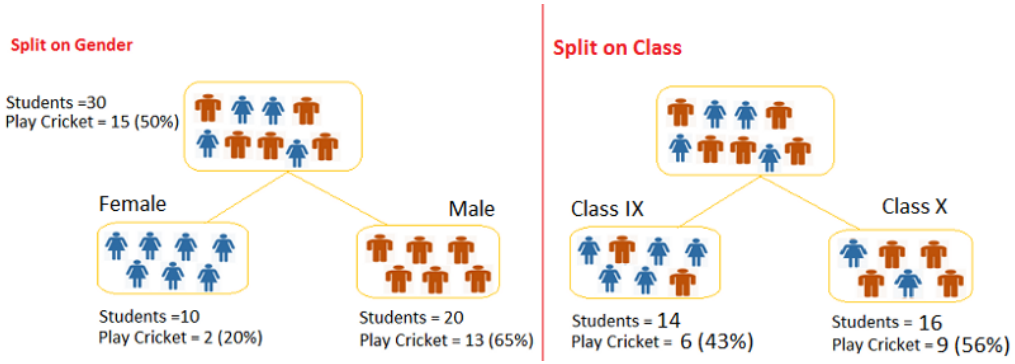
Entropy



(Note:characters shown in the box are not proportional to numbers shown outside!!!)

Split on Gender

- Gini for sub-node Female $= (0.2) * (0.2) + (0.8) * (0.8) = 0.68$
- Gini for sub-node Male $= (0.65) * (0.65) + (0.35) * (0.35) = 0.55$
- Weighted Gini for Split Gender $= (10/30) * 0.68 + (20/30) * 0.55 = 0.59$



Split on Class

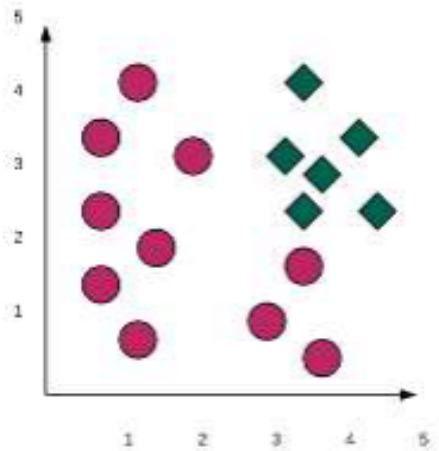
- Gini for sub-node Class IX $= (0.43) * (0.43) + (0.57) * (0.57) = 0.51$
- Gini for sub-node Class X $= (0.56) * (0.56) + (0.44) * (0.44) = 0.51$
- Weighted Gini for Split Class $= (14/30) * 0.51 + (16/30) * 0.51 = 0.51$

Above, you can see that Gini score for Split on Gender is higher than Split on Class, hence, the node split will take place on Gender.

GINI Index (Recap)

A measure of inequality developed by an Italian named Gini.

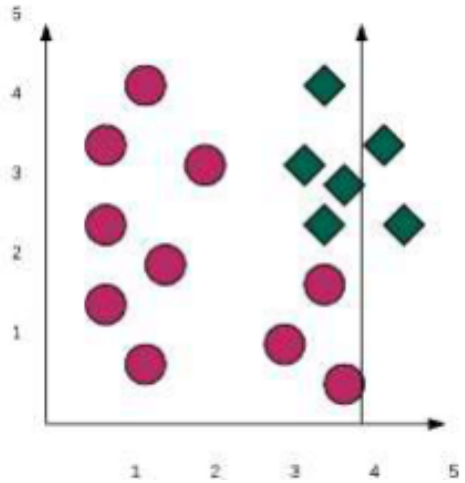
- Definition : Expected error rate $G(S) = 1 - [\sum p(j|t)^2]$
- GINI =0 (All elements are same class)
- GINI =0.5 (All elements evenly split between classes)



$G(t) = 1 - [(\frac{10}{16})^2 + (\frac{6}{16})^2] = 0.46875$

GINI Gain (Recap)

- Definition : $G(A, S) = G(S) - \sum(s_j/s)G(S)$
- If we Split $X < 4$
- $Gini < 4 = 0.4081$
- $Gini > 4 = 0$
- Gini Gain = 0.4687 - 10/16 (0.4081) - (0/16) (0)
- Gini Gain = 0.2136



When to use which?

- Gini for continuous attributes
- Entropy for categorical.
- Entropy is slower to calculate than GINI
- Gini may fail with very small probability
- Difference between the two is theoretically around 2%

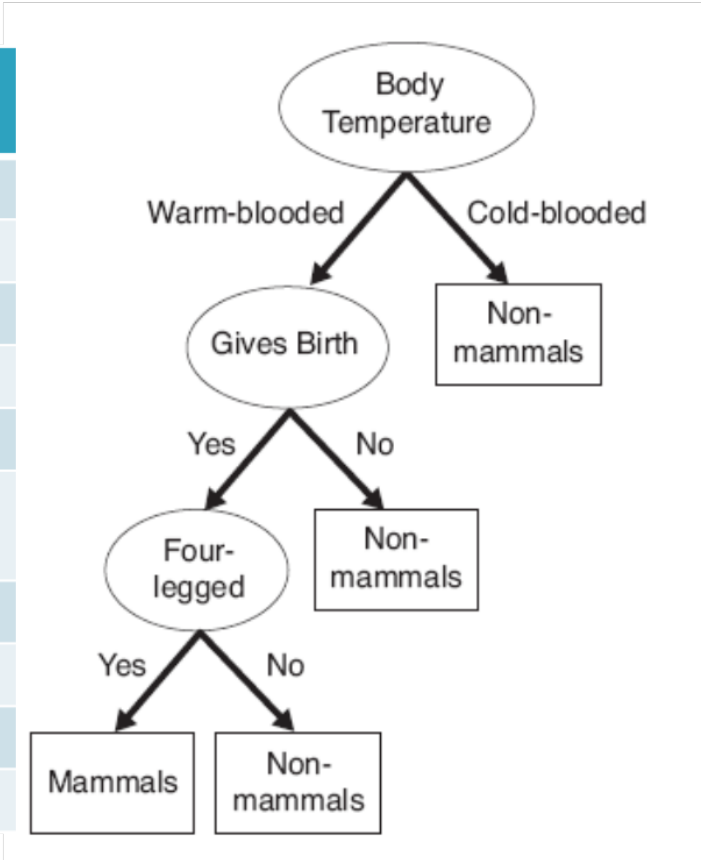
Over-fitting of Trees

Overfitting Example

Training Set

Name	Body Temp.	Gives Birth	4 legged	Hibernates	Class (Mammal?)
Porcupine	Warm	Yes	Yes	Yes	Yes
Cat	Warm	Yes	Yes	No	Yes
Bat	Warm	Yes	No	Yes	No
Whale	Warm	Yes	No	No	No
Salamander	Cold	No	Yes	Yes	No
Komodo Dragon	Cold	No	Yes	No	No
Python	Cold	No	No	Yes	No
Salmon	Cold	No	No	No	No
Eagle	Warm	No	No	No	No
Guppy	Cold	Yes	No	No	No

Two training records are mislabeled.

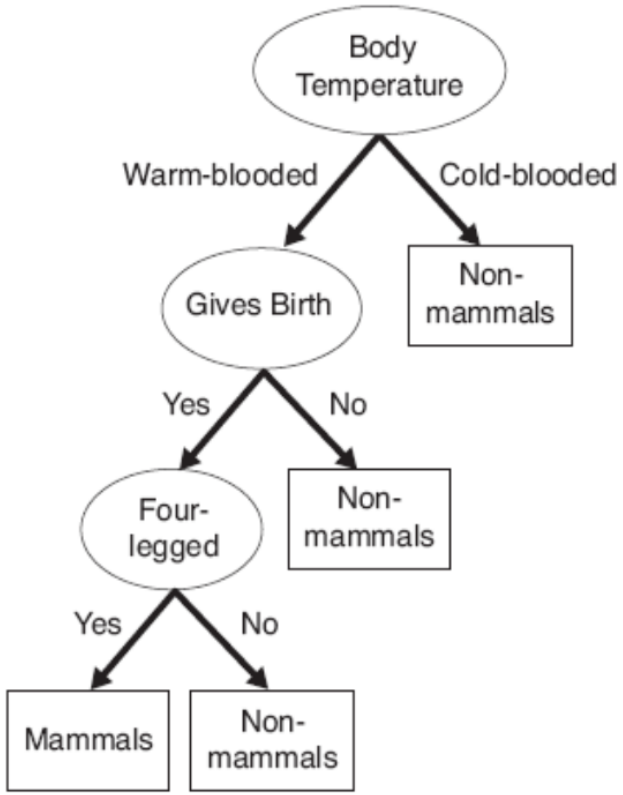


Tree perfectly fits training data.

Overfitting Example

Testing Set

Name	Body Temp.	Gives Birth	4 legged	Hibernates	Class (Mammal?)
Human	Warm	Yes	No	No	Yes
Pigeon	Warm	No	No	No	No
Elephant	Warm	Yes	Yes	No	Yes
Leopard Shark	Cold	Yes	No	No	No
Turtle	Cold	No	Yes	No	No
Penguin	Cold	No	No	No	No
Eel	Cold	No	No	No	No
Dolphin	Warm	Yes	No	No	Yes
Spiny Anteater	Warm	No	Yes	Yes	Yes
Gila Monster	Cold	No	Yes	Yes	No

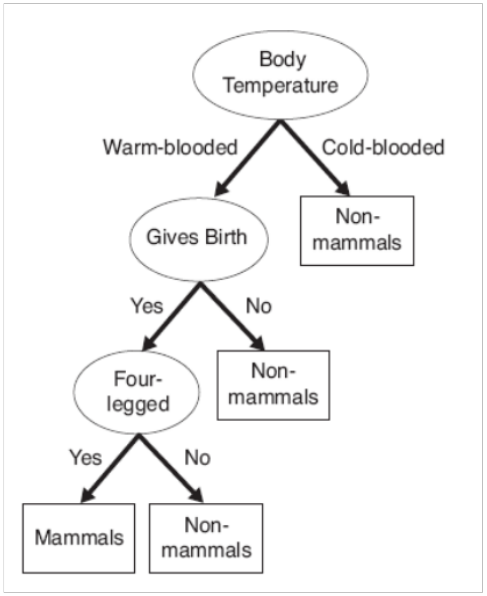


Test error rate: 30%

Overfitting Example

Testing Set

Name	Body Temp.	Gives Birth	4 legged	Hibernates	Class (Mammal?)
Human	Warm	Yes	No	No	Yes
Pigeon	Warm	No	No	No	No
Elephant	Warm	Yes	Yes	No	Yes
Leopard Shark	Cold	Yes	No	No	No
Turtle	Cold	No	Yes	No	No
Penguin	Cold	No	No	No	No
Eel	Cold	No	No	No	No
Dolphin	Warm	Yes	No	No	Yes
Spiny Anteater	Warm	No	Yes	Yes	Yes
Gila Monster	Cold	No	Yes	Yes	No

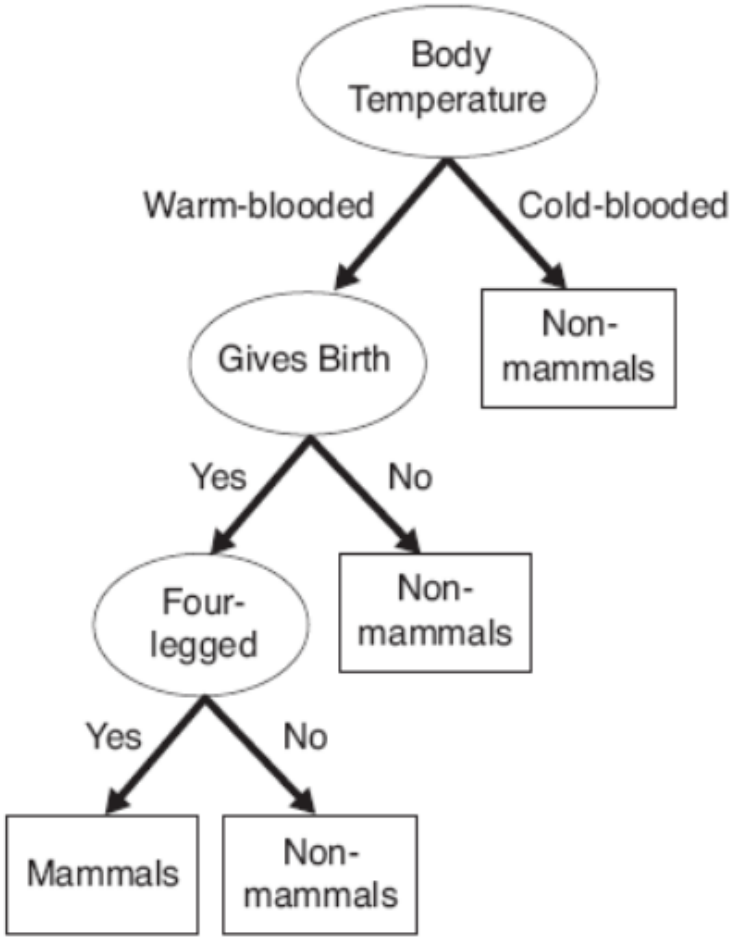


Test error rate: 30%

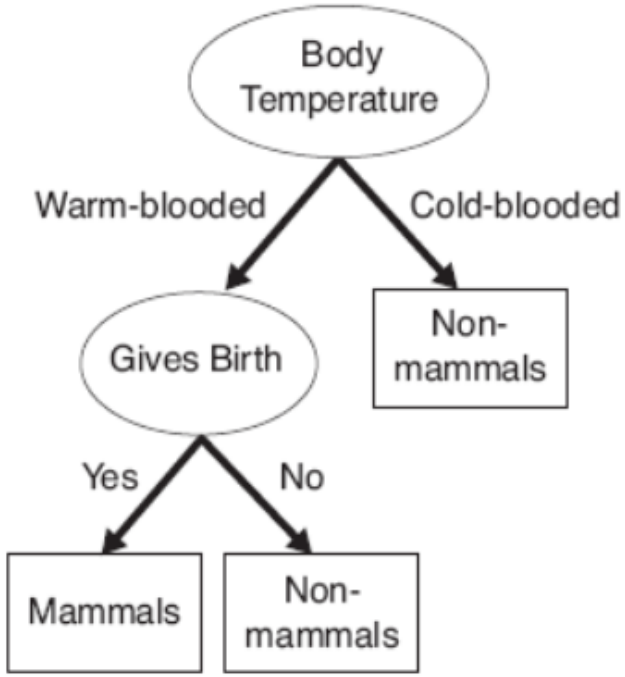
Reasons for misclassifications:

- Mislabeled records in training data
- "Exceptional case"
  - Unavoidable
  - Minimal error rate achievable by any classifier

Overfitting Example



(a) Model M1



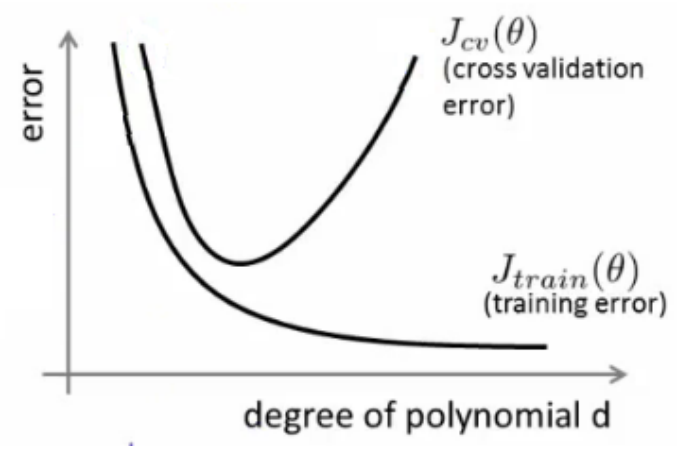
(b) Model M2

Over-fitting and Decision Trees

- Tree tries to reach till you get to all pure nodes, however unusual they are. These unusual points may not be there in the test sets or in general.
- The likelihood of over-fitting occurring increases as a tree gets deeper.
- Whats the way to detect?

Detection

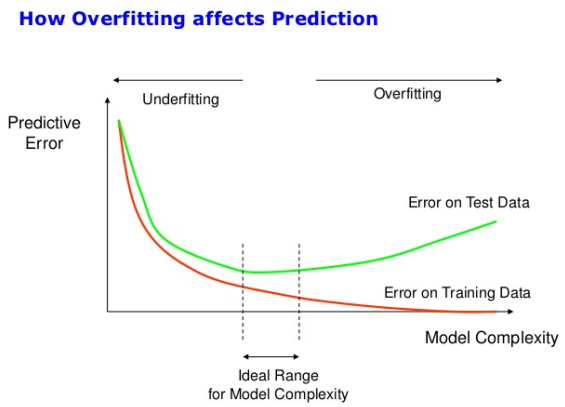
Roughly speaking, over-fitting typically occurs when the ratio $\frac{ComplexityOfTheModel}{TrainingSize}$ is too high.



- If your data is in two dimensions, you have 10000 points in the training set and the model is a line, you are likely to under-fit.
- If your data is in two dimensions, you have 10 points in the training set and the model is 100-degree polynomial, you are likely to over-fit.

(Ref:Why Is Overfitting Bad in Machine Learning? - StackOverflow)

Solution



- In many texts, "iterations" is used on X axis, which is easy to imagine in case of Neural Networks, where each epoch refines model by fitting better question/weights.
- In case of Machine Learning, there no iterations. How to detect Over-fitting in case of Decision Tree?
- Initially when, tree is small it has not fit the data yet, so its under fitting phase. Losses are HIGH for both, training as well as testing (validation)
- As more levels get added in tree or more degrees in polynomial regression, under-fitting reduces and losses come down.
- At one point, training loss still reduces but validation loss starts jumping up. Thats over fitting. Model is fitting training data TOO tightly and is

Over-fitting and Decision Trees

Solution : Pruning

- Tree pruning identifies and removes subtrees within a decision tree that are likely to be due to noise and sample variance in the training set used to induce it.
- Pruning will result in decision trees being created that are not consistent with the training set used to build them.
- But we are more interested in created prediction models that generalize well to new data!
- Solutions: Pre-pruning (Early Stopping), Post-pruning

Pre-pruning Techniques

- Stop creating subtrees when the number of instances in a partition falls below a threshold
- Information gain measured at a node is not deemed to be sufficient to make partitioning the data worthwhile
- Depth of the tree goes beyond a predefined limit
- Benefits: Computationally efficient; works well for small datasets.
- Downsides: Stopping too early will fail to create the most effective trees.

Post-pruning Techniques

- Decision tree initially grown to its maximum size
- Then examine each branch
- Branches that are deemed likely to be due to over-fitting are pruned.
- Post-pruning tends to give better results than pre-pruning
- Which is faster? Post-pruning is more computationally expensive than pre-pruning because entire tree is grown
- Techniques: Reduced Error Pruning; Cost Complexity Pruning

Decision Trees for Regression

Regression Trees

- Target Attribute:
  - Decision (Classification) Trees: qualitative
  - Regression Trees: continuous
- Decision trees: reduce the entropy in each subtree
- Regression trees: reduce the variance in each subtree
- Idea: adapt ID3 algorithm measure of Information Gain to use variance rather than node impurity

Regression Trees

When predicting a numeric variable, the idea of a tree construction remains the same, but the quality criteria changes. Variance around the mean:

$$D = \frac{1}{\ell}\sum_{i=1}^{\ell}(y_i - \frac{1}{\ell}\sum_{i=1}^{\ell} y_i)^2,$$

where ℓ is the number of samples in a leaf, y_i is the value of the target variable. Simply put, by minimizing the variance around the mean, we look for features that divide the training set in such a way that the values of the target feature in each leaf are roughly equal.

(Ref: https://www.kaggle.com/kashnitsky/topic-3-decision-trees-and-knn)

Regression Trees Example

Let's generate some data distributed by the function $f(x) = e^{-x^2} + 1.5 * e^{-(x-2)^2}$ with some noise. Then we will train a tree on it and show what predictions it makes.

(Ref: https://www.kaggle.com/kashnitsky/topic-3-decision-trees-and-knn)

Regression Trees Example

```
n_train = 150
n_test = 1000
noise = 0.1

def f(x):
    x = x.ravel()
    return np.exp(-x ** 2) + 1.5 * np.exp(-(x - 2) ** 2)

def generate(n_samples, noise):
    X = np.random.rand(n_samples) * 10 - 5
    X = np.sort(X).ravel()
    y = np.exp(-X ** 2) + 1.5 * np.exp(-(X - 2) ** 2) + \
    np.random.normal(0.0, noise, n_samples)
    X = X.reshape((n_samples, 1))
    return X, y

X_train, y_train = generate(n_samples=n_train, noise=noise)
X_test, y_test = generate(n_samples=n_test, noise=noise)
```

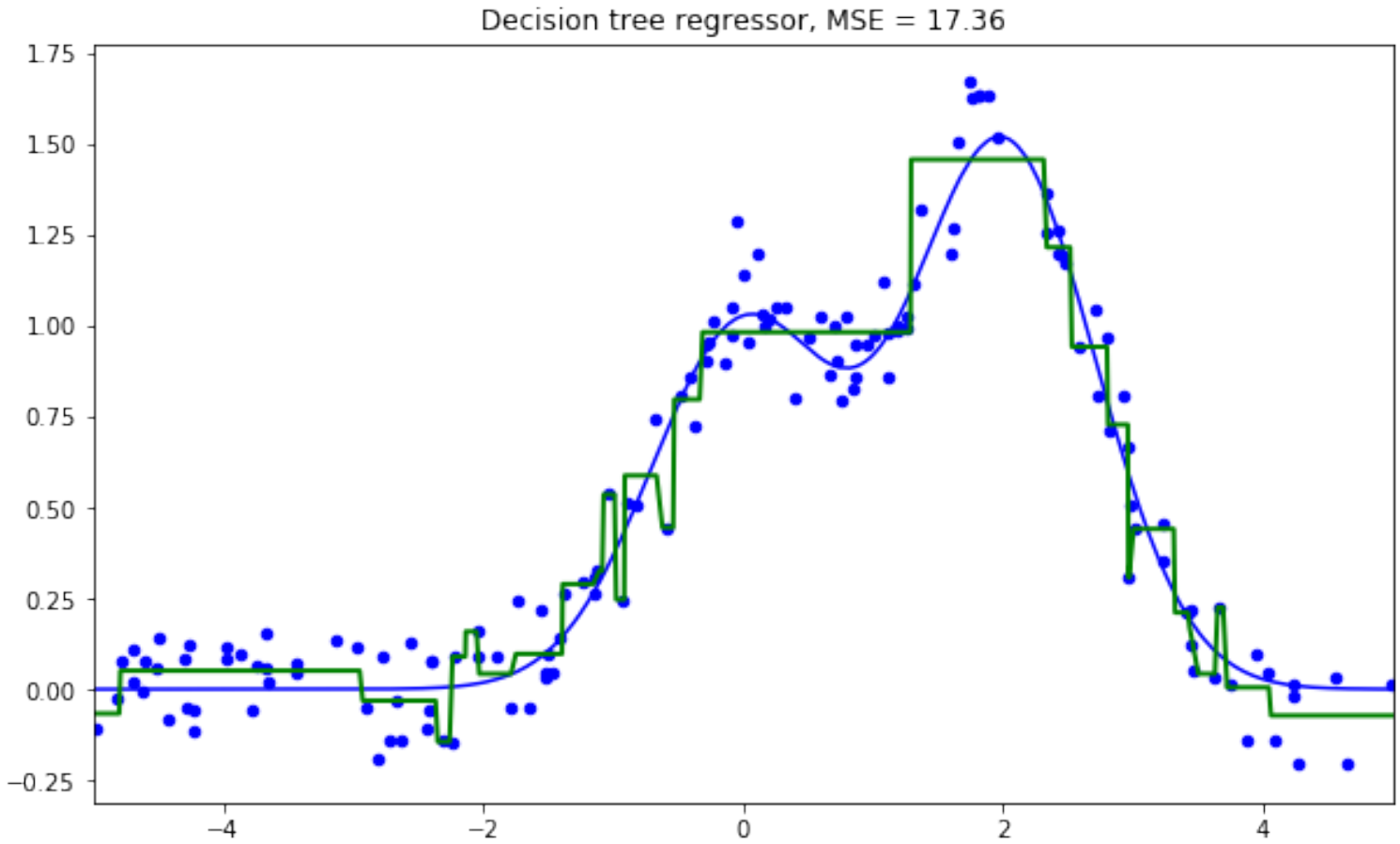
(Ref: https://www.kaggle.com/kashnitsky/topic-3-decision-trees-and-knn)

Regression Trees Example

```
from sklearn.tree import DecisionTreeRegressor

reg_tree = DecisionTreeRegressor(max_depth=5, random_state=17)

reg_tree.fit(X_train, y_train)
reg_tree_pred = reg_tree.predict(X_test)

plt.figure(figsize=(10, 6))
plt.plot(X_test, f(X_test), "b")
plt.scatter(X_train, y_train, c="b", s=20)
plt.plot(X_test, reg_tree_pred, "g", lw=2)
plt.xlim([-5, 5])
plt.title("Decision tree regressor, MSE = %.2f" % np.sum((y_test -
     reg_tree_pred) ** 2))
plt.show()
```

(Ref: https://www.kaggle.com/kashnitsky/topic-3-decision-trees-and-knn)

Binary outcome



We see that the decision tree approximates the data with a piecewise constant function.

(Ref: https://www.kaggle.com/kashnitsky/topic-3-decision-trees-and-knn)

Regression trees

The fitting algorithm for learning such a tree is as follows:

- Consider splitting on every possible unique value of every single variable in the data. Pick the 'best' split from amongst all of these options.
- Partition the training data into two classes based on step 1.
- Now, iteratively apply step 1 separately to each partition of the data.
- Continue splitting each subset until an appropriate stopping criteria has been reached.
- Calculate the mean of all the training data in a terminal node (i.e., 'leaf') of the learned tree structure. Use this as the predicted value for future inputs that fall within the same bucket.

Stopping criterion

There are many commonly used stopping criterion, often used simultaneously (if any is satisfied stop splitting the partition):

- minimum number of training samples in a node
- maximum number of splits
- minimum improvement in the best split
- maximum depth of the tree

In practice, particularly for larger datasets, the maximum number of splits is the mostly commonly used.

Regression Tree Splits

Classification Trees

- Gain: "goodness of the split"
- larger gain = better split (better test condition)

Regression Trees

- Impurity (variance) at a node:
- Select feature to split on that minimizes the weighted variance across all resulting partitions:

Decision Tree Algorithms

Types of Decision Tree Algorithms: ID3

- ID3 or Iterative Dichotomizer, was the first of three Decision Tree implementations developed by Ross Quinlan
- Uses Information Gain as splitting criterion.
- It does not apply any pruning procedure

Types of Decision Tree Algorithms: CART

- CART stands for Classification and Regression Trees.
- Uses Gini Impurity instead.
- Gini Impurity is a measure of the homogeneity (or "purity") of the nodes.
- It constructs binary trees, namely each internal node has exactly two outgoing edges.
- It can handle both numeric and categorical variables

Types of Decision Tree Algorithms: C4.5

- Improved version on ID 3 by Quinlan.
- Uses Information Gain as splitting criterion.
- Accepts both continuous and discrete features;
- Handles incomplete data points;
- Solves over-fitting problem by pruning

Types of Decision Tree Algorithms: Comparison

	Splitting Criteria	Attribute type	Missing values	Pruning Strategy	Outlier Detection
ID3	Information Gain	Handles only Categorical value	Do not handle missing values.	No pruning is done	Susceptible to outliers
CART	Towing Criteria	Handles both Categorical & Numeric value	Handle missing values.	Cost-Complexity pruning is used	Can handle Outliers
C4.5	Gain Ratio	Handles both Categorical & Numeric value	Handle missing values.	Error Based pruning is used	Susceptible to outliers

Types of Decision Tree Algorithms: C5.0

- Evolution of ID3, better than C4.5
- Uses Information Gain Ratio as splitting criterion.
- Handles missing values well.

Advantages and Disadvantages of Trees

Advantages

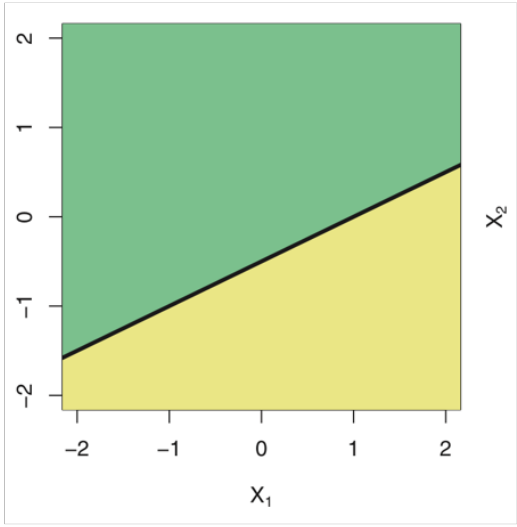
- Trees are very easy to explain
- Easier to explain than linear regression
- Trees can be displayed graphically and interpreted by a non-expert
- Decision trees may more closely mirror human decision-making

Disadvantages

- Trees usually do not have same level of predictive accuracy as other data mining algorithms
- But, predictive performance of decision trees can be improved by aggregating trees. Techniques: bagging, boosting, random forests

Compared to Linear Models

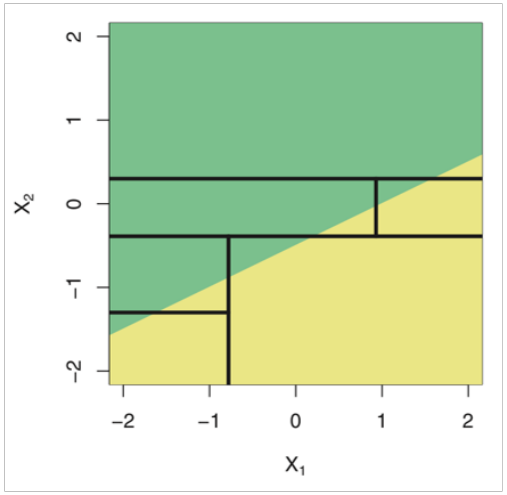
Two-classes: {green, blue}



Linear model can perfectly separate the two regions.

- What about a decision tree?

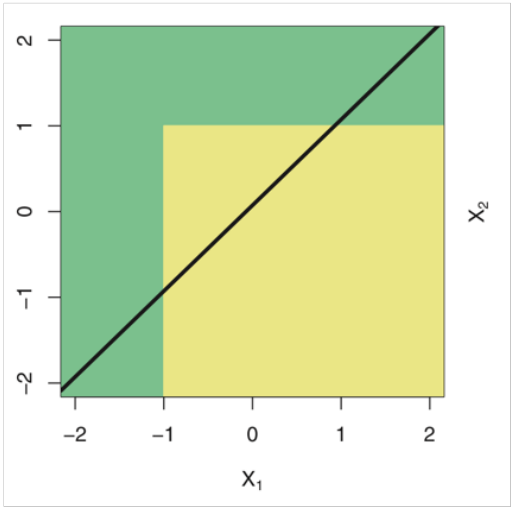
Decision boundary: linear



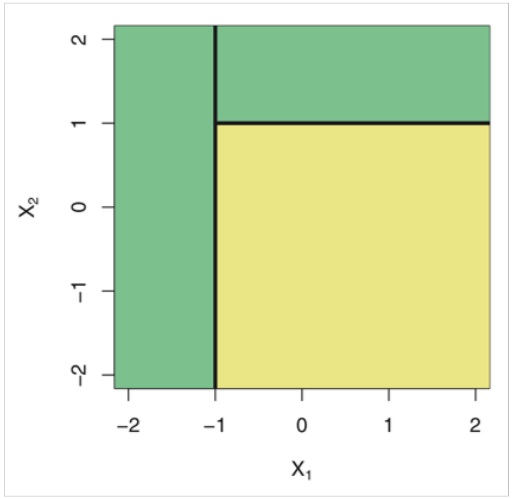
Decision tree cannot separate regions.

Compared to Linear Models

Two-classes: {green, blue}



Decision boundary: nonlinear



Linear model cannot perfectly separate the two regions.

- What about a decision tree?

A Decision tree can!

Decision Trees (Recap)

- A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature.
- The arcs coming from a node labeled with a feature are labeled with each of the possible values of the feature.
- Each leaf of the tree is labeled with a class or a probability distribution over the classes.
- For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of *If-Then-Else* decision rules.
- The deeper the tree, the more complex the decision rules and the fitter the model.

Pros and Cons of Single Decision Tree

Pros

- Simple interpretations
- Can be built quickly.
- Does built-in Feature selection. So, if any of the features does not get used in Splitting, it is, sort-of, ignored, not selected in prediction.

Cons

- Small changes in data affects structure of tree
- So, instead of single tree, a collection of Trees is preferred

Such collection is called as 'Ensemble'.

Ensemble Methods

- Currently using one single classifier induced from training data as our model, to predict class of test instance
- What if we used multiple decision trees?
- Motivation: committee of experts working together are likely to better solve a problem than a single expert
- But no "group think": each model should make predictions independently of other models in the ensemble
- In practice: methods work surprisingly well, usually greatly improve decision tree accuracy

Decision Tree with Scikit-Learn

Classification and Regression Trees (CART)

```
# Decision Tree Classifier
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
# load the iris datasets
dataset = datasets.load_iris()
# fit a CART model to the data
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

(Ref: Machine Learning Algorithm Recipes in scikit-learn - Jason Brownlee)

Ensemble

For Complex Decision Making

- How do I handle data for large number of variables?
- More complex the decision making, more experts are needed.
- Linux, for example, is such a complex system that building it took hundreds of experts.
- Ensemble means collection of models, meaning collection of experts!!

What is Ensemble?

- Definition: An ensemble is a set of elements that collectively contribute to a whole.
- A familiar example is a musical ensemble, which blends the sounds of several musical instruments to create a beautiful harmony, or architectural ensembles, which are a set of buildings designed as a unit.
- In ensembles, the (whole) harmonious outcome is more important than the performance of any individual part.

Theorem about Ensemble

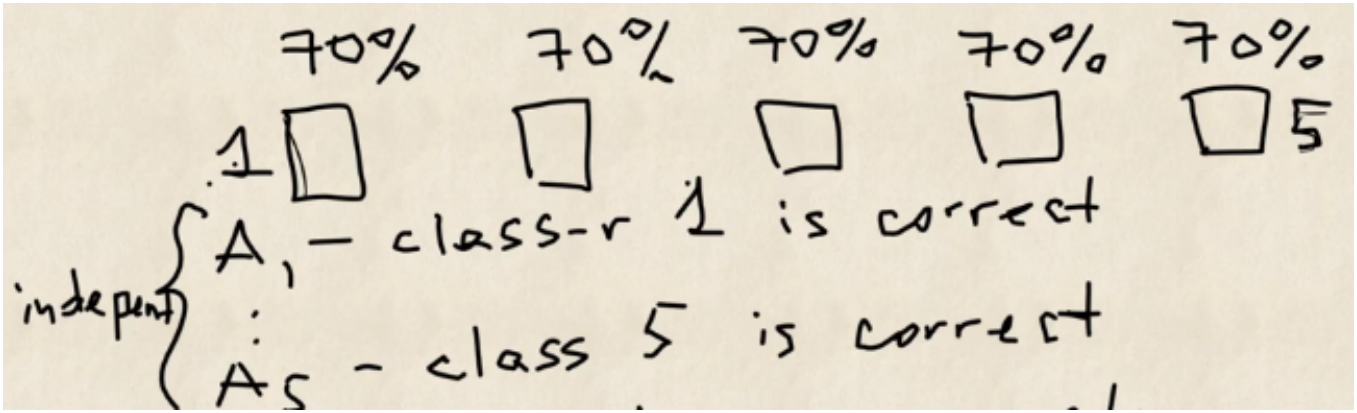
- Condorcet's jury theorem (1784) is about an ensemble in some sense.
- It states that, if each member of the jury makes an independent judgment and the probability of the correct decision by each juror is more than 0.5, then the probability of the correct decision by the whole jury increases with the total number of jurors and tends to one.
- On the other hand, if the probability of being right is less than 0.5 for each juror, then the probability of the correct decision by the whole jury decreases with the number of jurors and tends to zero.

Idea behind Ensemble

- Rational: 'No Free Lunch' Theorem: Even popular base classifiers will perform poorly on some datasets, where the learning classifier and data distribution do not match well
- Intuitive Justification: When combing multiple, independent, and diverse, decisions each of which is at least more accurate than random guessing then random errors cancel each other out, and correct decisions are reinforced

Idea behind Ensemble

- 5 classifiers each with 0.7 accuracy.
- All of them give predictions.
- Some match , some don't.
- Majority voting means $>= 3$ should be correct. So when all 5 or 4 or 3 are correct.



Idea behind Ensemble

- Probability that all 5 classifiers would be correct is $(0.7) * (0.7) \ldots = (0.7)^5$
- Probability that 4 classifiers would be correct and one incorrect is $(0.7)^4 \times (0.3)^1$
- Binomial theorems says that as there are $\binom{1}{5}$ arrangements of this are possible. See, the incorrect one can be any one of the 5. So the probability is $\binom{1}{5}(0.7)^4 \times (0.3)^1$
- Probability that 3 classifiers would be correct and two incorrect is $\binom{2}{5}(0.7)^3 \times (0.3)^2$
- Total probability is to add them up = $(0.7)^5 + \binom{1}{5}(0.7)^4 \times (0.3)^1 + \binom{2}{5}(0.7)^3 \times (0.3)^2 \approx 0.84$

Did you see? Individually they were 0.7 each, but together 0.84!!! If we take $N = 1000$ classifiers, then the total accuracy goes to 0.99

Jury to Ensemble?

$\mu = \sum_{i=m}^{N} \binom{N}{i} p^i (1-p)^{N-i}$
Where,

- N is the total number of jurors;
- m is a minimal number of jurors that would make a majority, that is $m = \frac{N+1}{2}$;
- $\binom{N}{i}$ is the number of i-combinations from a set with N elements.
- p is the probability of the correct decision by a juror;
- μ is the probability of the correct decision by the whole jury.

It can be seen that if $p > 0.5$, then $\mu > p$. In addition, if $N \to \infty$, then $\mu \to 1$.

Wisdom of the crowd

- In 1906, Francis Galton visited a country fair in Plymouth where he saw a contest being held for farmers.
- 800 participants tried to estimate the weight of a slaughtered bull.
- The real weight of the bull was 1198 pounds. Although none of the farmers could guess the exact weight of the animal, the average of their predictions was 1197 pounds.

A similar idea for error reduction is adopted in the field of Machine Learning, called Ensemble.

Ensemble Rationale

Another Example:

- Assume we have 25 binary classifiers
- Each has error rate: $\epsilon = 0.35$
- If all 25 classifiers are identical:
- They will vote the same way on each test instance.
- Majority wins, so the max one, which is $\epsilon = 0.35$ is the final answer.

Ensemble Rationale

- Ensemble method only makes a wrong prediction if more than half of the base classifiers predict incorrectly.
- The probability that the ensemble classifier make's a wrong prediction (for half the set, ie from 13 to 25) is, binomial, ie:

$$e_{\text{ensemble}} = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

- 6% much less than 35%

(Ref: "A Survey of Ensemble Classification - SMU", "Tan, Steinbach , and Kumar")

Ensemble Rationale

Conditions necessary for an ensemble classifier to perform better than a single classifier:

- Base classifiers should be independent of each other
- Base classifiers should not do worse than a classifier doing random guessing
- Example: for two-class problem, base classifier error rate: $\epsilon < .5$

When to Ensemble?

- Given a fixed-size number of training samples, our model will increasingly suffer from the "curse of dimensionality" if we increase the number of features.
- The challenge of individual, un-pruned decision trees is that the model often ends up being too complex for the underlying training data – decision trees are prone to over-fitting.

For Complex Decision Making

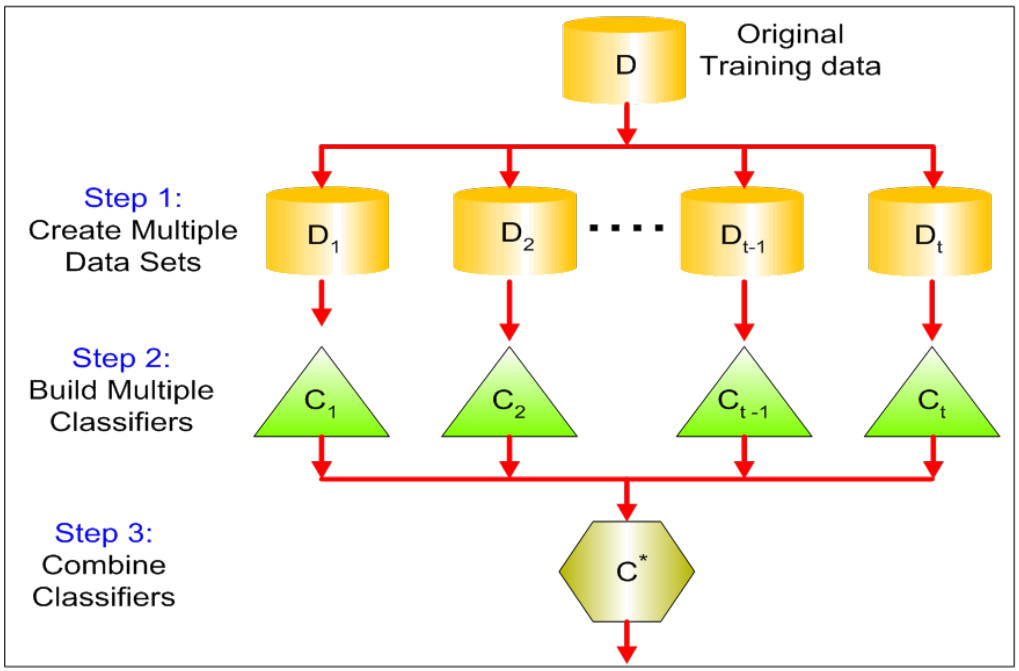
- Ensemble are called as“meta-algorithms”: approaches to combine several machine learning techniques into one predictive model
- Bagging and random forests are “bagging” algorithms that aim to reduce the complexity of models that overfit the training data.
- In contrast, boosting is an approach to increase the complexity of models that suffer from high bias, that is, models that underfit the training data.

Ensemble Characteristics

- Build multiple models from the same training data by creating each model on a modified version of the training data.
- Make a final, ensemble prediction by aggregating the predictions of the individual models
- Classification prediction: Let each model have a vote on the correct class prediction. Assign the class with the most votes.
- Regression prediction: Measure of central tendency (mean or median)

Ensemble Constructing

- By manipulating the training set.
- By manipulating the input features.
- By manipulating the class labels.
- By manipulating the learning algorithm.



(Reference: Basics of Ensemble Learning Explained in Simple English - Tavish Shrivastava)

Ensemble Approaches

Decision Tree Model Ensembles

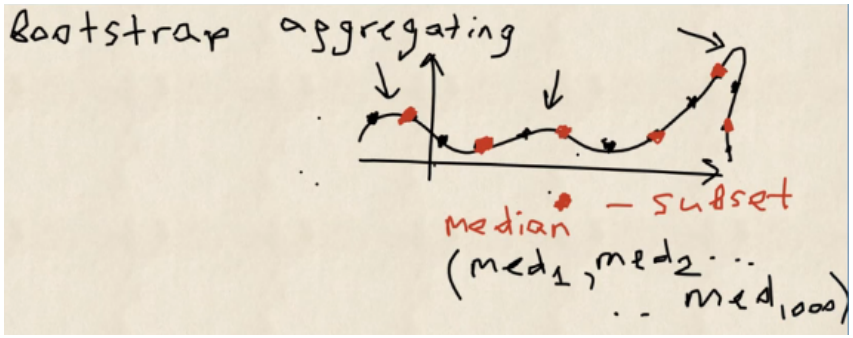
- Bagging
- Boosting
- Random Forests

Bagging (Bootstrapping)

- Stands for **B**ootstrap **Agg**regat**ing**
- It was proposed by Leo Breiman in 1994.
- A way to decrease the variance of your prediction
- By generating additional data for training from your original data-set
- Using combinations with repetitions to produce multi-sets of the same cardinality/size as your original data.

Statistical Idea of Bootstrapping

- Say, you have a complex distribution and you are asked to find its median.
- If it was some standard distribution like Normal, then formulae are available
- Here the trick is to sample, say randomly pick 10 points, and then find median of them.
- Do this multiple times, say , 1000. You will have an array of medians. Find its median as answer.



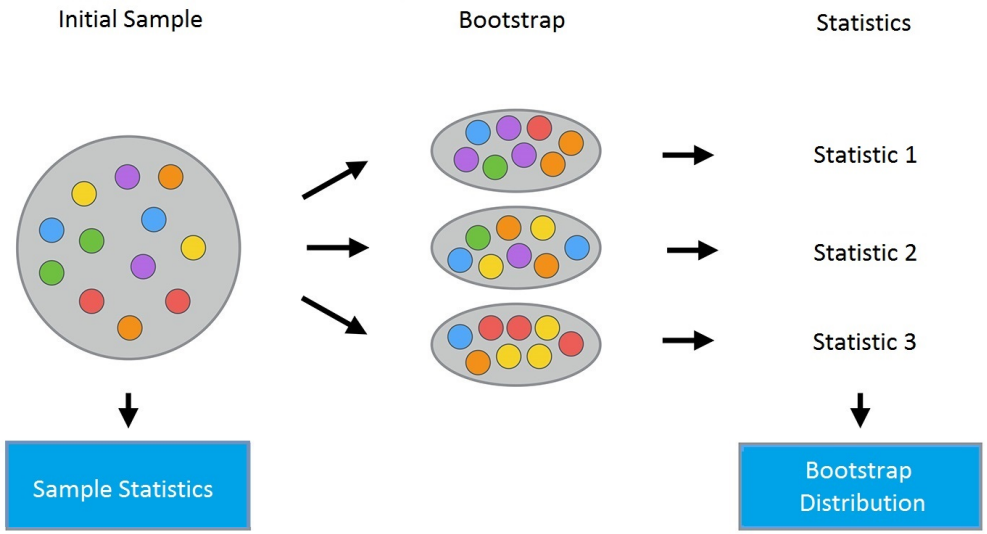
Bootstrap is used for Bagging.

Idea of Bagging

- Let there be a sample X of size N.
- We can make a new sample from the original sample by drawing N elements from the latter randomly and uniformly, with replacement.
- In other words, we select a random element from the original sample of size N and do this N times.
- All elements are equally likely to be selected, thus each element is drawn with the equal probability $\frac{1}{N}$.
- Due to 'With replacement' mode, in all cases total is always N.
- Note that, because we put the balls back, there may be duplicates in the new sample. Let's call this new sample X_1

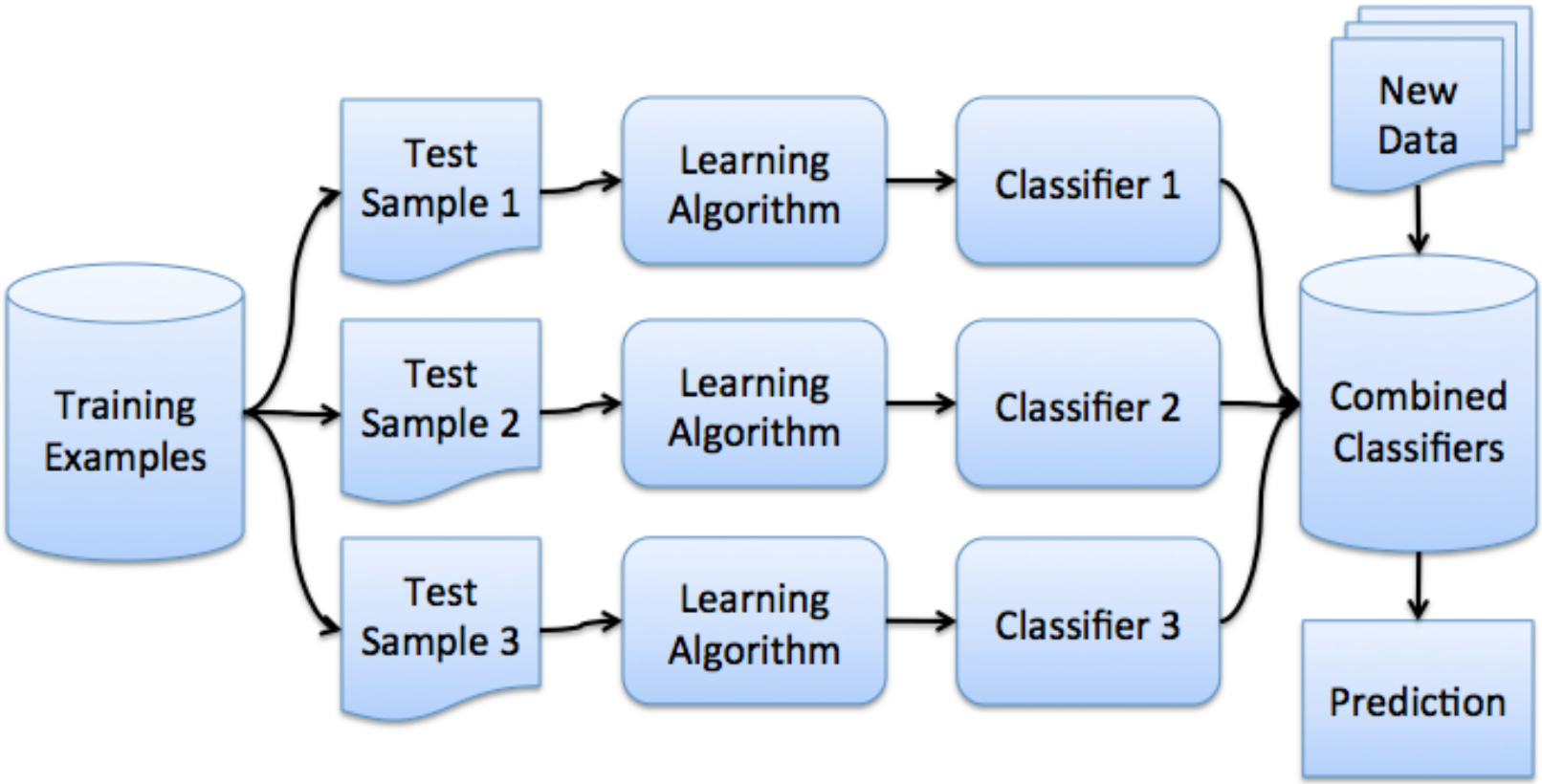
Bagging

By repeating this procedure M times, we create M bootstrap samples $X_1, \ldots, X_M$. In the end, we have a sufficient number of samples and can compute various statistics of the original distribution.



Distribution of the statistics (say, median) from each sample creates the Bootstrap distribution.
Condition for Bagging: Samples/rows must be idd (independently identically distributed)

Bagging


Training Examples
Test Sample 1
Test Sample 2
Test Sample 3
Learning Algorithm
Learning Algorithm
Learning Algorithm
Classifier 1
Classifier 2
Classifier 3
New Data
Combined Classifiers
Prediction

Bagging

- We train a number (ensemble) of decision trees from bootstrap samples of our training set.
- Bootstrap sampling means drawing random samples from our training set with replacement.
- E.g., if our training set consists of 7 training samples, our bootstrap samples

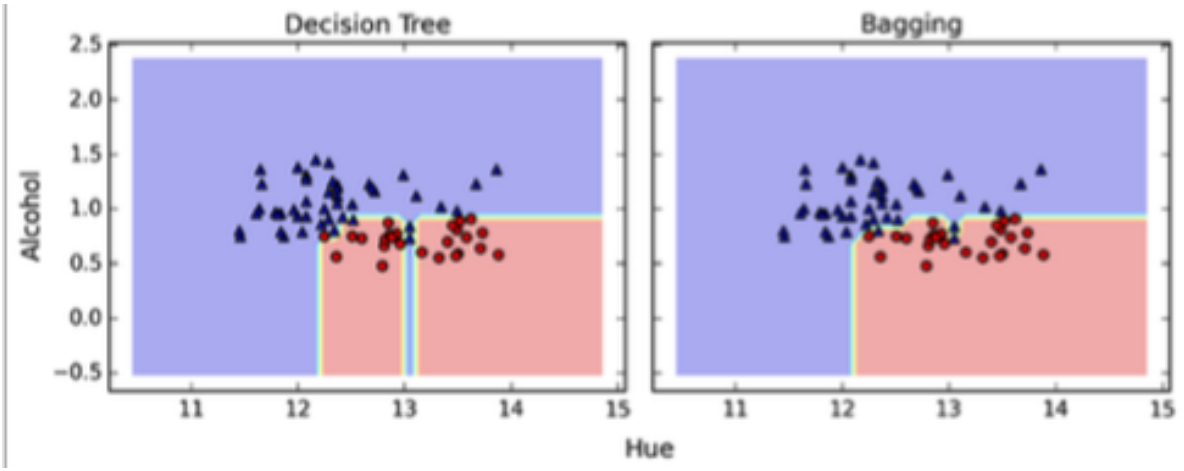
Sample indices	Bagging round 1	Bagging round 2	...
1	2	7	...
2	2	3	...
3	1	2	...
4	3	1	...
5	7	1	...
6	2	7	...
7	4	7	...
	C_1	C_2	C_m

Bagging

- After we trained our (m) decision trees, we can use them to classify new data via majority rule.
- For instance, we'd let each decision tree make a decision and predict the class label that received more votes.
- Typically, this would result in a less complex decision boundary, and the bagging classifier would have a lower variance (less overfitting) than an individual decision tree.

Bagging

- Below is a plot comparing a single decision tree (left) to a bagging classifier (right)



(Reference: https://sebastianraschka.com/faq/docs/bagging-boosting-rf.html)

Boosting

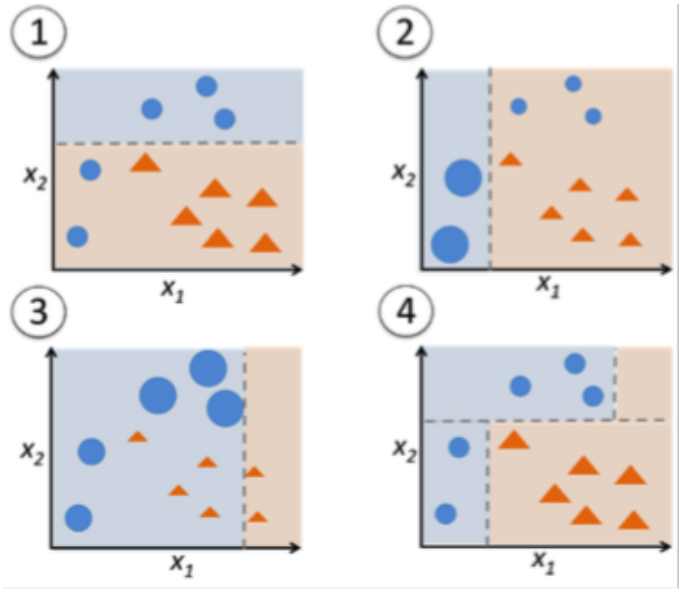
- Takes a random sample from training set
- Trains a model
- Uses original data from Training to test how good this new model is.
- For the next model, when we chose sample from the training set, the points which got mis-classified previously are weighted more to be picked up now.
- Train the next model.
- We test both models together by assembling their individual output.

Boosting

- In contrast to bagging, we use very simple classifiers as base classifiers, so-called "weak learners."
- Picture these weak learners as "decision tree stumps" – decision trees with only 1 splitting rule.

Boosting

- We start with one decision tree stump (1) and "focus" on the samples it got wrong.
- In the next round, we train another decision tree stump that attempts to get these samples right (2);
- Again, this 2nd classifier will likely get some other samples wrong, so do it one more time



Boosting

- Unlike bagging, in the classical boosting the subset creation is not random and depends upon the performance of the previous models
- Every new subsets contains the elements that were (likely to be) misclassified by previous models.

Boosting

- Boosting works by iteratively creating models and adding them to the ensemble
- Iteration stops when a predefined number of models have been added
- Each new model added to the ensemble is biased to pay more attention to instances that previous models mis-classified (weighted dataset).

Boosting Example

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

- Suppose that *Instance #4* is hard to classify.
- Weight for this instance will be increased in future iterations, as it gets misclassified repeatedly.
- Examples not chosen in previous round (*Instances #1, #5*) also may have better chance of being selected in next round.
  - *Why?* Predictions in previous round are likely to be wrong since they weren't trained on.

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*

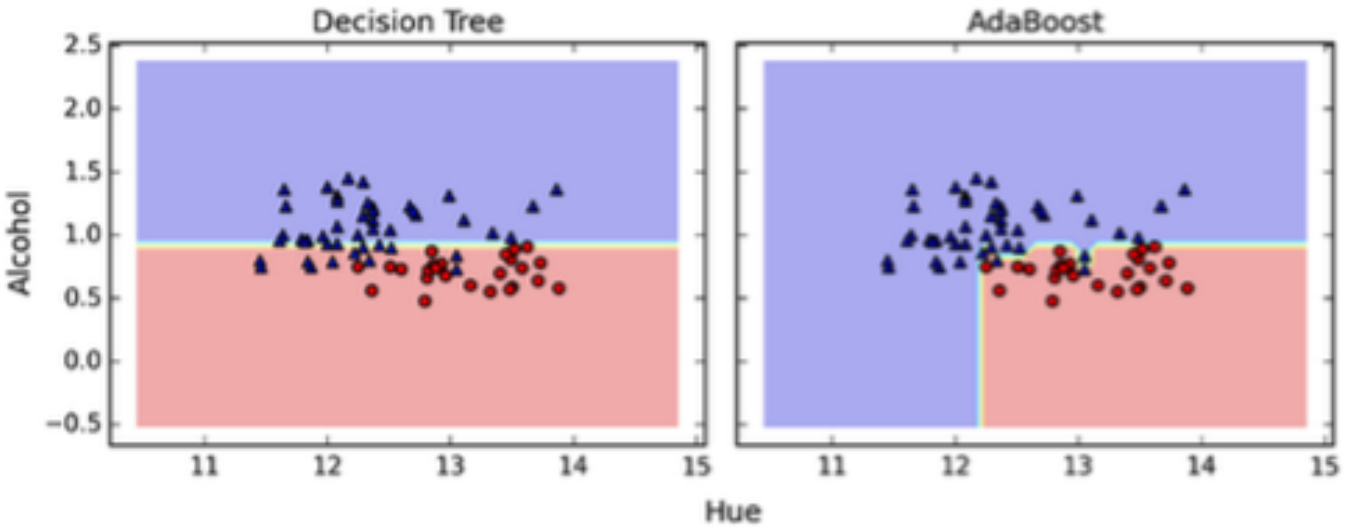
- As boosting rounds proceed, instances that are the hardest to classify become even more prevalent.

Prediction

- Once the set of models have been created the ensemble makes predictions using a weighted aggregate of the predictions made by the individual models.
- The weights used in this aggregation are simply the confidence factors associated with each model.
- Several different boosting algorithms exist
- Different by:
  - How weights of training instances are updated after each boosting round
  - How predictions made by each classifier are combined. Each boosting round produces one base classifier

AdaBoost

- AdaBoost is a popular boosting algorithm
- "Adaptive" or "incremental" learning from mistakes.
- Eventually, we will come up with a model that has a lower bias than an individual decision tree (thus, it is less likely to underfit the training data)



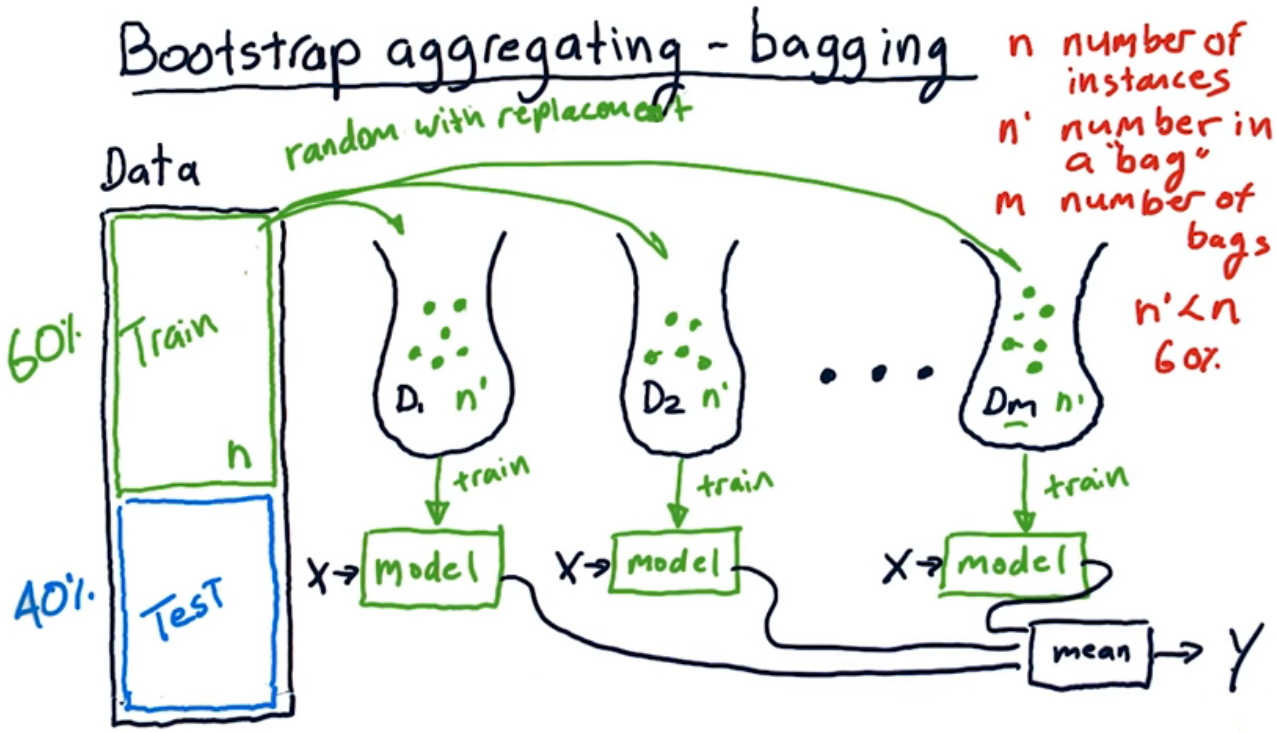
Bagging vs. Boosting

- Boosting: Sample with nonuniform distribution
- Unlike bagging where each instance had equal chance of being selected
- Boosting Motivation: focus on instances that are harder to classify
- How: give harder instances more weight in future rounds

Bagging (recap)

- Parallel ensemble: each model is built independently
- Aim to decrease variance, not bias
- Suitable for high variance low bias models (complex models)
- An example of a tree based method is random forest, which develop fully grown trees (note that RF modifies the grown procedure to reduce the correlation between trees)

Bagging (recap)

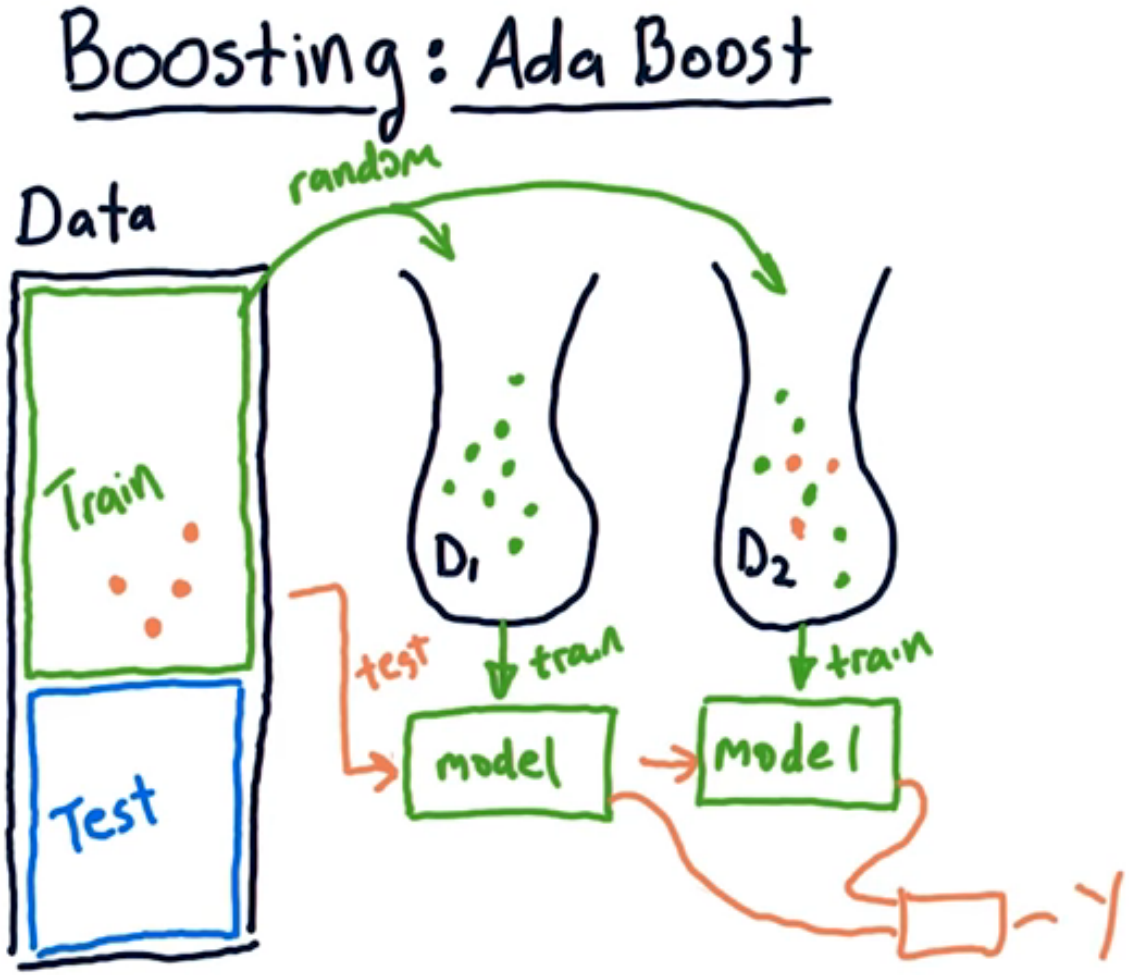


(Reference: Bootstrap aggregating bagging - Udacity)

Boosting (recap)

- Sequential ensemble: try to add new models that do well where previous models lack
- Aim to decrease bias, not variance
- Suitable for low variance high bias models
- An example of a tree based method is gradient boosting

Boosting (recap)



(Reference: Bootstrap aggregating bagging - Udacity)

Ensemble Choices

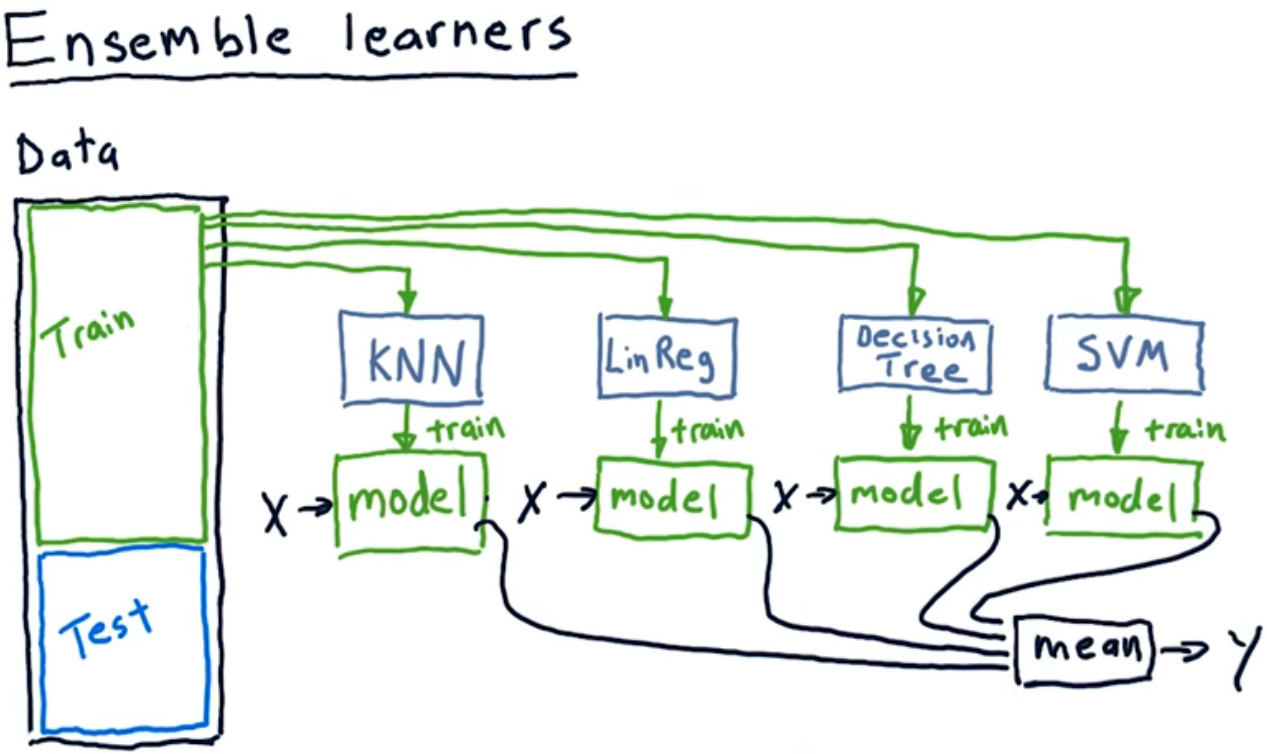
- Which approach should we use?
- Bagging is simpler to implement and parallelize than boosting and, so, may be better with respect to ease of use and training time.

Ensemble Choices

Empirical results indicate:

- Boosted decision tree ensembles were the best performing model of those tested for datasets containing up to 4,000 descriptive features.
- Random forest ensembles (based on bagging) performed better for datasets containing more that 4,000 features.

Ensemble Methods (Recap)



(Reference: Bootstrap aggregating bagging - Udacity)

Ensemble Methods (Recap)

- Advantages:
  - Astonishingly good performance
  - Modeling human behavior: making judgment based on many trusted advisors, each with their own specialty
- Disadvantage:
  - Combined models rather hard to analyze. Tens or hundreds of individual models
  - Current research: making models more comprehensible

Ensemble with Scikit-Learn

(Ref: Ensemble Machine Learning Algorithms in Python with scikit-learn - Jason Brownlee)

Read Data

Import Pima Inidans dataset and split into Features (X) and target (Y)

```
import pandas

url =
     "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/
     pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
     'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
```

Bagged Decision Trees

- Bagging performs best with algorithms that have high variance. A popular example are decision trees, often constructed without pruning.
- In the example shown, see usage of the BaggingClassifier with the Classification and Regression Trees algorithm (DecisionTreeClassifier). A total of 100 trees are created.

Bagged Decision Trees

```
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees,
     random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

0.770745044429
```

Running the example, we get a robust estimate of model accuracy.

Boosting Algorithms

- Boosting ensemble algorithms creates a sequence of models that attempt to correct the mistakes of the models before them in the sequence.
- Once created, the models make predictions which may be weighted by their demonstrated accuracy and the results are combined to create a final output prediction.

Ada Boost

```
# AdaBoost Classification
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier

seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.760457796309
```

Running the example provides a mean estimate of classification accuracy.

Stochastic Gradient Boosting

```
# Stochastic Gradient Boosting Classification
from sklearn import model_selection
from sklearn.ensemble import GradientBoostingClassifier

seed = 7
num_trees = 100
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Running the example provides a mean estimate of classification accuracy.

```
0.764285714286
```

Random Forest

Random Forests

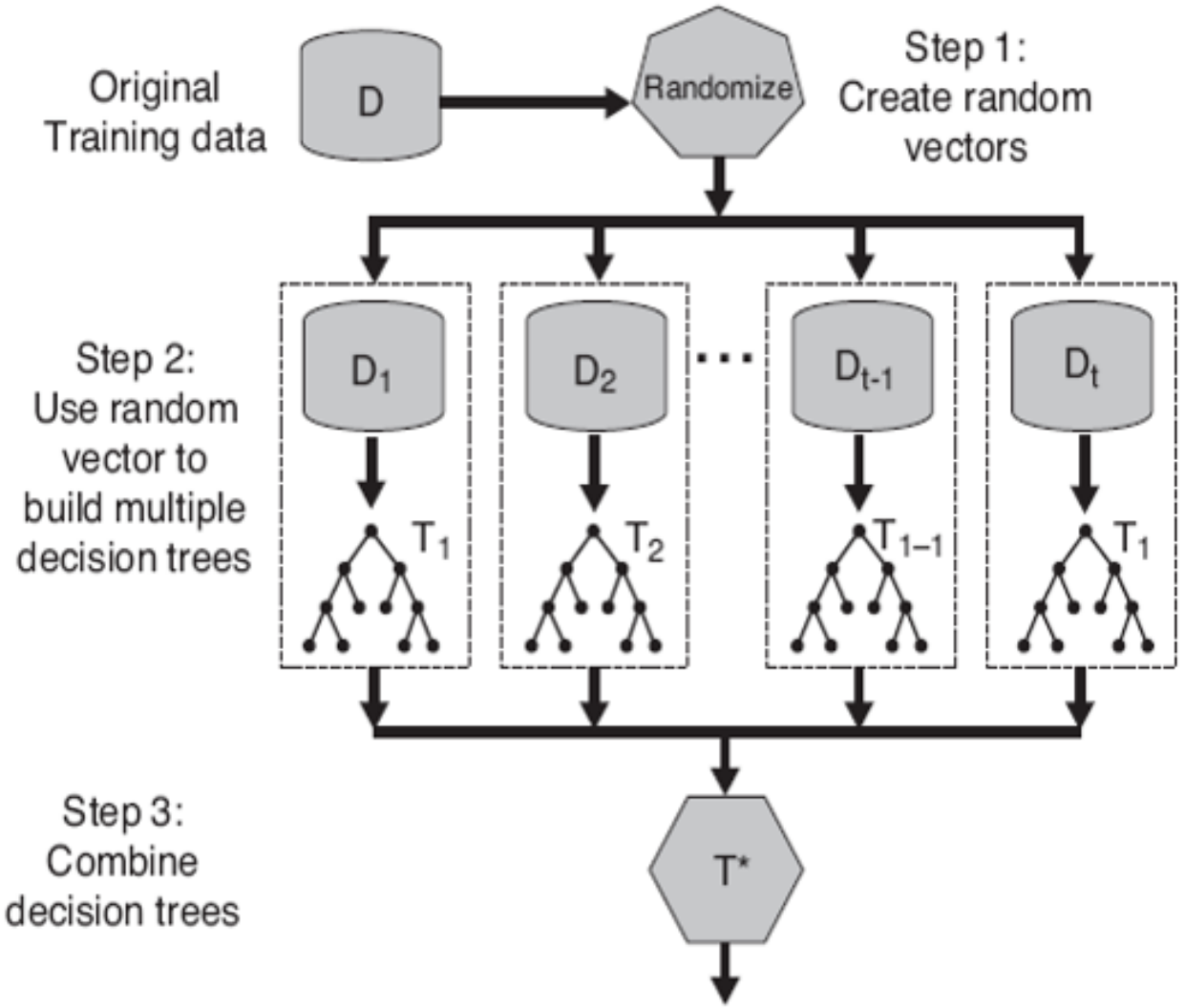
- Leo Breiman managed to apply bootstrapping not only in statistics but also in machine learning.
- He, along with Adel Cutler, extended and improved the random forest algorithm proposed by Tin Kam Ho.
- They combined the construction of uncorrelated trees using CART, bagging, and the random subspace method.
- Decision trees are a good choice for the base classifier in bagging because they are quite sophisticated and can achieve zero classification error on any sample.

See "Working with Leo Breiman on Random Forests, Adele Cutler" video at Youtube.

Random Forests

- Actually a bagging algorithm
- Also here, we draw random bootstrap samples from our training set.
- However, in addition to the bootstrap samples, we also draw random subsets of features for training the individual trees;
- In bagging, we provide each tree with the full set of features.
- Due to the random feature selection, the trees are more independent of each other compared to regular bagging, which often results in better predictive performance (due to better variance-bias trade-offs), and it's also faster than bagging, because each tree learns only from a subset of features.

Random Forest



Random Forests

- Random forests fit many decision trees to the same training data.
- First takes a random sample of features to grow each decision tree.
- Evaluate. Compare with previous trees.
- A forest is produced.
- A joint classification model is produced using all the trees.

Random Forest Algorithm

- Let the number of instances be equal to ℓ, and the number of feature dimensions be equal to d.
- Choose L as the number of individual models in the ensemble.
- For each model l, choose the number of features $dl < d$. As a rule, the same value of dl is used for all the models.
- For each model l, create a training set by selecting dl features at random from the whole set of d features.
- Train each model.
- Apply the resulting ensemble model to a new instance by combining the results from all the models in L. You can use either majority voting or aggregation of the posterior probabilities.

Random Forest Algorithm

Constructing a random forest of N trees goes as follows:

- For each $k = 1, \ldots, N$:
  - Generate a bootstrap sample X_k.
  - Build a decision tree b_k on the sample X_k:
    - Pick the best feature dimension according to the given criteria.
    - Split the sample by this feature to create a new tree level. Repeat this procedure until the sample is exhausted.
    - Building the tree until any of its leaves contains no more than n_{min} instances or until a certain depth is reached.
    - For each split, we first randomly pick m features from the d original ones and then search for the next best split only among the subset.

Random Forest Algorithm

The final classifier is defined by:
$a(x) = \frac{1}{N} \sum_{k=1}^{N} b_k(x)$

- We use the majority voting for classification and the mean for regression.
- For classification problems, it is advisable to set $m = \sqrt{d}$.
- For regression problems, we usually take $m = \frac{d}{3}$, where d is the number of features.
- It is recommended to build each tree until all of its leaves contain only $n_{\text{min}} = 1$ examples for classification and $n_{\text{min}} = 5$ examples for regression.
- You can see random forest as bagging of decision trees with the modification of selecting a random subset of features at each split.

Task

There are 7 jurors in the courtroom. Each of them individually can correctly determine whether the defendant is guilty or not with 80% probability. How likely is the jury will make a correct verdict jointly if the decision is made by majority voting?

- 20.97%
- 80.00%
- 83.70%
- 96.66%

Answer XXX

Random Forest Splits

- Selection of random subset of features to determine each split
- How large should subset be?
  - Typically root(K), for K features works well for classification
  - (K / 3) for regression
- Tree is grown to full size with pruning
- Overall prediction is majority vote from all individually trained trees
- Different variations of Random Forests exist

Comparison with Decision Trees and Bagging

- In a random forest, the best feature for a split is selected from a random subset of the available features while, in bagging, all features are considered for the next best split.
- "Normal" decision tree induction utilizes full set of features to determine each split
- Random forest injects randomness

Pros and cons of random forests

Pros

- High prediction accuracy; will perform better than linear algorithms in most problems; the accuracy is comparable with that of boosting.
- Robust to outliers, thanks to random sampling.
- Insensitive to the scaling of features as well as any other monotonic transformations due to the random subspace selection.
- Doesn't require fine-grained parameter tuning, works quite well out-of-the-box. With tuning, it is possible to achieve a 0.5–3
- Efficient for datasets with a large number of features and classes.
- Handles both continuous and discrete variables equally well.

Pros and cons of random forests

Pros

- Rarely overfits. In practice, an increase in the tree number almost always improves the composition. But, after reaching a certain number of trees, the learning curve is very close to the asymptote.
- There are developed methods to estimate feature importance.
- Works well with missing data
- Provides means to weight classes on the whole dataset as well as for each tree sample.
- Easily parallelized and highly scalable.

Pros and cons of random forests

Cons

- In comparison with a single decision tree, Random Forest's output is more difficult to interpret.
- There are no formal p-values for feature significance estimation. Performs worse than linear methods in the case of sparse data: text inputs, bag of words, etc.
- Unlike linear regression, Random Forest is unable to extrapolate. But, this can be also regarded as an advantage because outliers do not cause extreme values in Random Forests.
- In the case of categorical variables with varying level numbers, random forests favor variables with a greater number of levels. The tree will fit more towards a feature with many levels because this gains greater accuracy.
- The resulting model is large and requires a lot of RAM.

Random Forest (Recap)

- Random Forest is an ensemble technique used for classification, regression.
- It combines the output of weaker techniques in order to get a stronger result.
- The weaker technique in this case is a decision tree.

Random Forest (Recap)

- Decision trees work by splitting the and re-splitting the data by features.
- If a decision tree is split along good features, it can give a decent predictive output.
- Random Forest works by averaging decision tree output, but it's a bit more complicated than that.

Random Forest (Recap)

- It also ranks an individual tree's output, by comparing it to the known output from the training data.
- This allows it to rank features. Some of the decision trees will perform better, and so the features within the tree will be deemed more important.

Random Forest with Scikit-Learn

(Ref: Ensemble Machine Learning Algorithms in Python with scikit-learn - Jason Brownlee)

Random Forest Classification

- Random forest is an extension of bagged decision trees.
- Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers.
- Specifically, rather than greedily choosing the best split point in the construction of the tree, only a random subset of features are considered for each split.

Random Forest Classification

```
# Random Forest Classification
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier
import pandas
url =
     "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/
     pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
     'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
max_features = 3
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = RandomForestClassifier(n_estimators=num_trees,
     max_features=max_features)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

0.770745044429
```

Running the example provides a mean estimate of classification accuracy.

Thanks . . .

- Feel free to follow me at:
  - Github (github.com/yogeshhk) for open-sourced Data Science training material, etc.
  - Kaggle (www.kaggle.com/yogeshkulkarni) for Data Science datasets and notebooks.
  - Medium (yogeshharibhaukulkarni.medium.com) and also my Publications:
    - Desi Stack https://medium.com/desi-stack
    - TL;DR,W,L https://medium.com/tl-dr-w-l
- Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com