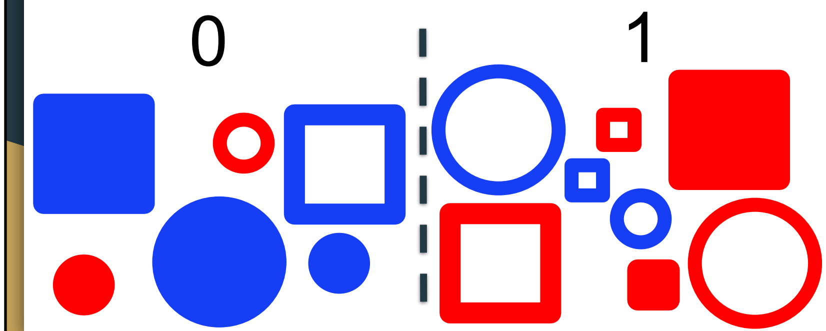
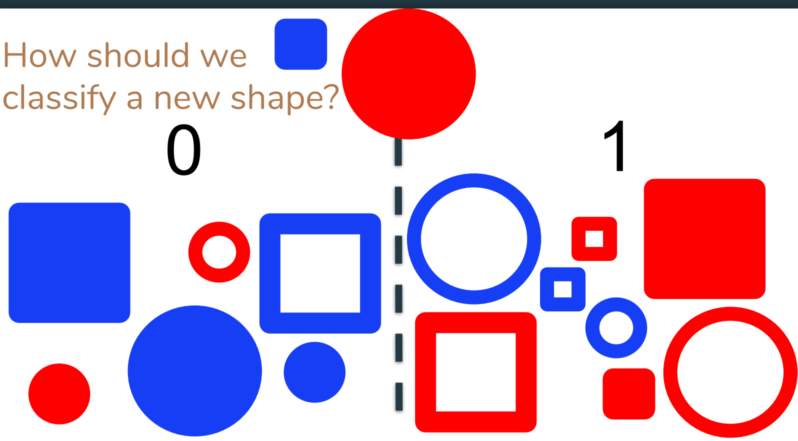
Decision Trees

Classification Problem



How should we classify a new shape?



Classification Problem

isLarge?	isRed?	isFilled?	isCircle?	Class
Yes	No	No	Yes	1
Yes	No	Yes	Yes	0
No	No	No	Yes	1
Yes	No	No	No	0
No	No	No	No	1
No	No	Yes	Yes	0
Yes	No	Yes	No	0
No	Yes	Yes	No	1
Yes	Yes	No	No	1
No	Yes	No	Yes	0
Yes	Yes	Yes	No	1
No	Yes	Yes	Yes	0
Yes	Yes	No	Yes	1
No	Yes	No	No	1

Randomly Permute Rows

isLarge?	isRed?	isFilled?	isCircle?	Class
Yes	No	Yes	Yes	0
No	Yes	No	No	1
Yes	No	No	Yes	1
Yes	No	Yes	No	0
Yes	Yes	No	No	1
No	Yes	Yes	Yes	0
No	No	No	No	1
No	No	No	Yes	1
No	Yes	Yes	No	1
No	Yes	No	Yes	0
Yes	Yes	Yes	No	1
Yes	Yes	No	Yes	1
Yes	No	No	No	0
No	No	Yes	Yes	0

Training and Test Data

isLarge?	isRed?	isFilled?	isCircle?	Class
Yes	No	Yes	Yes	0
No	Yes	No	No	1
Yes	No	No	Yes	1
Yes	No	Yes	No	0
Yes	Yes	No	No	1
No	Yes	Yes	Yes	0
No	No	No	No	1
No	No	No	Yes	1
No	Yes	Yes	No	1
No	Yes	No	Yes	0
Yes	Yes	Yes	No	1
Yes	Yes	No	Yes	1
Yes	No	No	No	0
No	No	Yes	Yes	0

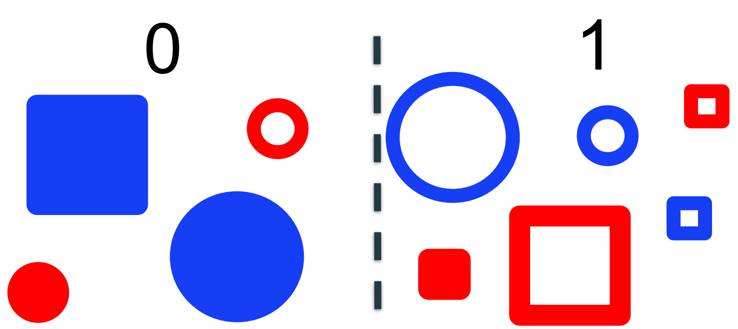
Training data

Test data

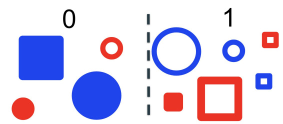
X

y

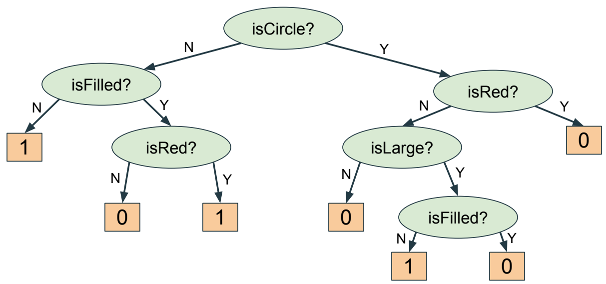
Training Data



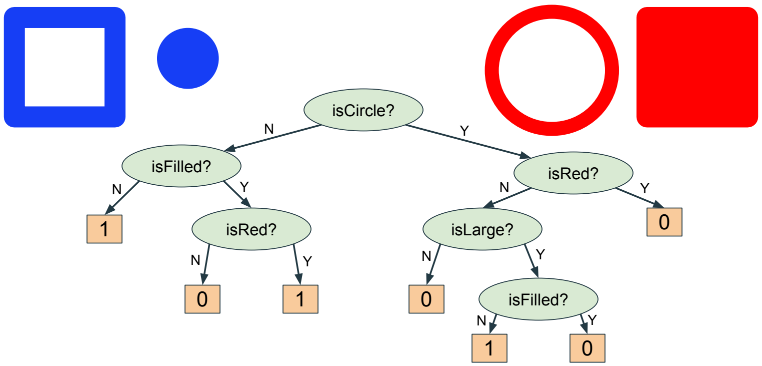
Decision Trees



Make a sequence of decisions based on features



Evaluate Tree on Test Data



Building a Decision Tree

- The tree should predict labels of training data accurately
- The tree should be compact, so that it generalizes well to non-training data without overfitting

Bad Algorithm for Building a Tree

- Generate all possible trees and pick the smallest tree that is consistent with the training data

Good Algorithm for Building a Tree

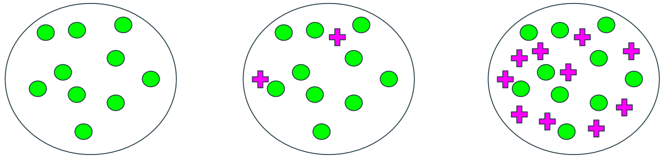
- Greedily choose “best” feature by which to split training examples and make this the root node of a (sub)-tree
- Recursively generate subtrees for the split training examples using remaining features
- Stop when leaves are perfectly classified

How do we determine the greedy choice, i.e., “best” feature to use to split training examples?

- The greedy choice feature should aim to minimize the depth of the tree
- A perfect feature choice divides the examples into sets, each of which are all 0 or all 1 and thus will be leaves of the tree
- A poor feature choice divides the examples into sets with the same proportion of 0 and 1 classes as the undivided set

Entropy

Entropy is a measure of uncertainty or impurity.

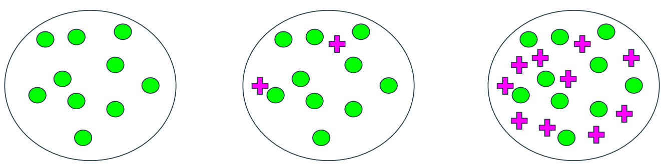


Minimum impurity | Small impurity | Large impurity

Acquisition of information corresponds to a reduction in entropy.

Entropy

$$\text{Entropy} = \sum_i -p_i \log_2 p_i \qquad \text{where } p_i \text{ is the probability of class } i$$



Minimum impurity | Small impurity | Large impurity

$\text{Entropy} = -(10/10)\log_2(10/10) + -(0/10)\log_2(0/10) = 0$

$\text{Entropy} = -(10/12)\log_2(10/12) + -(2/12)\log_2(2/12) = 0.65$

$\text{Entropy} = -(10/20)\log_2(10/20) + -(10/20)\log_2(10/20) = 1$

Information Gain

The *information gain* when we divide the data using a particular feature is the reduction in entropy.

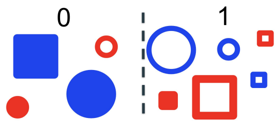
When deciding what feature to test at the root of a tree, we look at the entropy at the root node (parent) and the entropy at the children of the root node.

Information Gain = entropy(parent) - [weighted average entropy(children)]

Using different features at the root to split the data will result in different children with different entropies.

We greedily choose the feature that maximizes information gain

Information Gain (IG) Examples

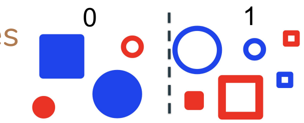


IG(feature) = entropy(parent) -
[weighted average entropy(children)]

IG(isLarge?) = $-(4/10)\log_2(4/10) + -(6/10)\log_2(6/10)$ - Entropy of parent
[(6/10) * entropy(small examples) +
(4/10) * entropy(large examples)] Weighted average entropy of two children

Weights of two children

Information Gain (IG) Examples



IG(feature) = entropy(parent) -
[weighted average entropy(children)]

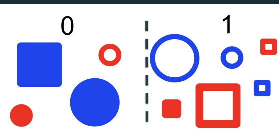
IG(isLarge?) = $-(4/10)\log_2(4/10) + -(6/10)\log_2(6/10)$ -
[(6/10) * entropy(small examples) +
(4/10) * entropy(large examples)]

$= -(4/10)\log_2(4/10) + -(6/10)\log_2(6/10) -$
$[\ (6/10) * (-(2/6)\log_2(2/6) + -(4/6)\log_2(4/6)) +$
$(4/10) * (-(2/4)\log_2(2/4) + -(2/4)\log_2(2/4))\ ]$

$\approx 0.53 + 0.44 -$
$[\ (6/10) * (0.53 + 0.39) +$
$(4/10) * (0.5 + 0.5)\ ]$

≈ 0.018

Information Gain (IG) Examples



IG(feature) = entropy(parent) -
[weighted average entropy(children)]

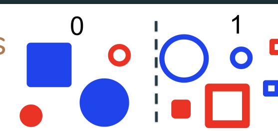
IG(isRed?) = $-(4/10)\log_2(4/10) + -(6/10)\log_2(6/10)$ -
[(5/10) * entropy(blue examples) +
(5/10) * entropy(red examples)]

$= -(4/10)\log_2(4/10) + -(6/10)\log_2(6/10) -$
$[\ (5/10) * (-(2/5)\log_2(2/5) + -(3/5)\log_2(3/5)) +$
$(5/10) * (-(2/5)\log_2(2/5) + -(3/5)\log_2(3/5))\ ]$

$\approx 0.53 + 0.44 -$
$[\ (5/10) * (0.53 + 0.44) +$
$(5/10) * (0.53 + 0.44)\ ]$

≈ 0.000

Information Gain (IG) Examples



IG(feature) = entropy(parent) -
[weighted average entropy(children)]

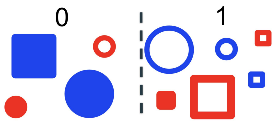
IG(isFilled?) = $-(4/10)\log_2(4/10) + -(6/10)\log_2(6/10)$ -
[(6/10) * entropy(hollow examples) +
(4/10) * entropy(filled examples)]

$= -(4/10)\log_2(4/10) + -(6/10)\log_2(6/10) -$
$[\ (6/10) * (-(1/6)\log_2(1/6) + -(5/6)\log_2(5/6)) +$
$(4/10) * (-(3/4)\log_2(3/4) + -(1/4)\log_2(1/4))\ ]$

$\approx 0.53 + 0.44 -$
$[\ (6/10) * (0.43 + 0.22) +$
$(4/10) * (0.31 + 0.50)\ ]$

≈ 0.256

Information Gain (IG) Examples



IG(feature) = entropy(parent) - [weighted average entropy(children)]

$$\begin{aligned} IG(isCircle?) &= -(4/10)\log_2(4/10) + -(6/10)\log_2(6/10) - \\ &\quad [\ (5/10) * entropy(square\ examples) + \\ &\quad (5/10) * entropy(circle\ examples)\ ] \\ &= -(4/10)\log_2(4/10) + -(6/10)\log_2(6/10) - \\ &\quad [\ (5/10) * (-(1/5)\log_2(1/5) + -(4/5)\log_2(4/5)) + \\ &\quad (5/10) * (-(3/5)\log_2(3/5) + -(2/5)\log_2(2/5))\ ] \\ &\approx 0.53 + 0.44 - \\ &\quad [\ (5/10) * (0.46 + 0.26) + \\ &\quad (5/10) * (0.44 + 0.53)\ ] \\ &\approx 0.125 \end{aligned}$$

Variations

- What if our features are not binary (e.g., red/blue, small/large, circle/square) but can take on many discrete values (e.g., red/green/blue/purple/orange, small/medium/large, circle/triangle/rhombus/square)? What if we have continuous, real-valued features?
- In our tree building algorithm, to end the recursion we said to “stop when leaves are perfectly classified.” What if data cannot be perfectly classified?
- Pruning a tree may yield more compact trees that are better predictors on new data (i.e., less prone to overfitting)

Overfitting

- On some problems, a large tree will be constructed when there is actually no pattern to be found
- Consider the problem of trying to predict whether the roll of a die will come up 6 or not. Our features are the weather outside when the die was rolled, the name of the roller, and whether the die landed on a table or the floor.
- If the die is fair, the right thing to learn is a tree with a single node that says “no”
- But our decision tree learning algorithm will seize on any pattern it can find in the input. If it turns out that there are 2 rolls when Wendy rolled the die on the floor when it was rainy out, then the algorithm may construct a path that predicts 6 in that case.
- This problem is called *overfitting*
- Overfitting becomes more likely as the hypothesis space and the number of inputs grows, and less likely as we increase the number of training examples

Pros and Cons of Decision Trees

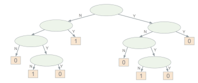
<u>Pros</u>

- Easy to understand how to implement
- Work well in practice for many problems
- Easy to explain model predictions, i.e., interpretable

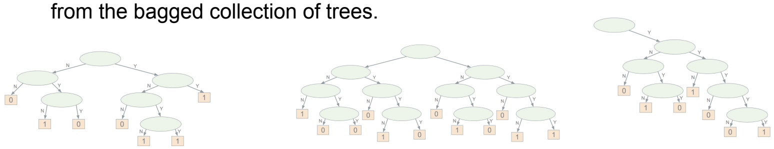
<u>Cons</u>

- Need to store large tree
- No principled pruning method to avoid overfitting
- Not effective for all problems, e.g., the majority function, which returns 1 if and only if more than half the inputs are 1, and returns 0 otherwise (requires an exponentially large decision tree)

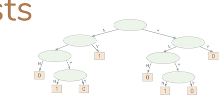
Ensemble Methods: Bagging



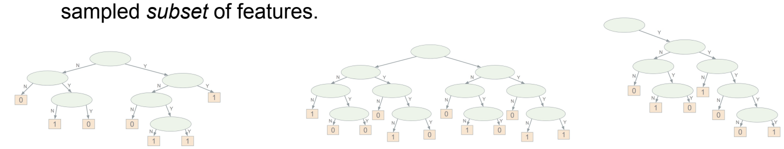
Bagging. Take repeated samples from the training set to generate many different bootstrapped training sets. Construct a tree for each bootstrapped training set, creating a collection of trees. To make a prediction for a new example, take the majority classification prediction from the bagged collection of trees.



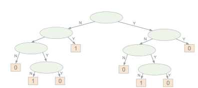
Ensemble Methods: Random Forests



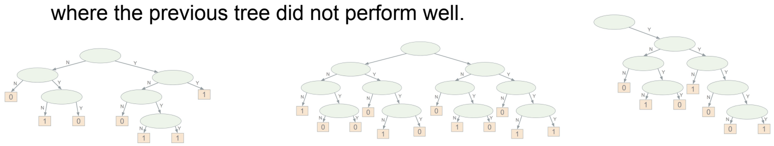
Random Forest. As with bagging, a collection of trees is created from bootstrapped training samples. With random forests, however, each time we construct a node in a tree by choosing greedily among the features, we choose among *all* features but rather among a randomly sampled *subset* of features.



Ensemble Methods: Boosting



Boosting. Trees are learned sequentially. Consecutive trees are based on the error from the previous tree. When an input is misclassified by a tree, its weight is increased so that the next tree is more likely to classify it correctly. Trees are kept small so that the boosting approach learns slowly. With each subsequent tree, we improve the fit in areas where the previous tree did not perform well.



Overview

- ML Algorithms
  - Supervised Learning
    - Non-Parametric
      - Decision Trees
      - kNN
      - Support Vector Machines
      - Collaborative Filtering
    - Parametric
      - Regression Models
        - Linear Regression
      - Linear Classifiers
        - Perceptron
        - Logistic Regression
      - Non-Linear Classifiers
        - Neural Networks
  - Unsupervised Learning
    - Hierarchical Clustering
    - K-Means
    - Gaussian Mixture Models
    - Dimensionality Reduction

Hidden Markov Models