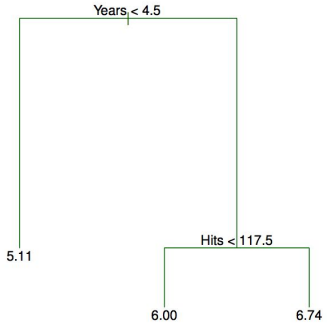
UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

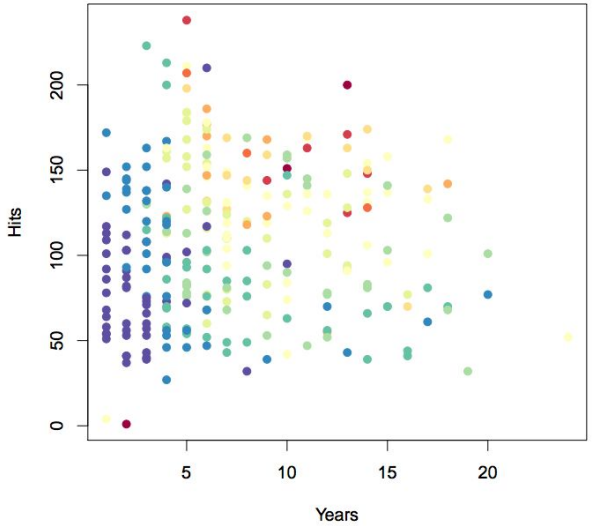
Decision Trees

Machine Learning

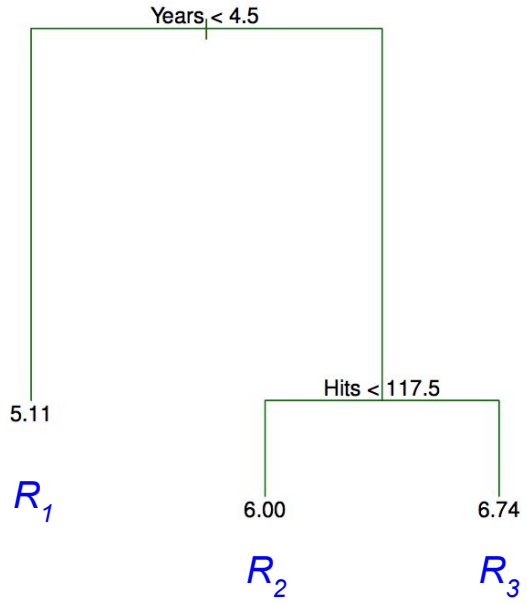
Not Exactly a Tree

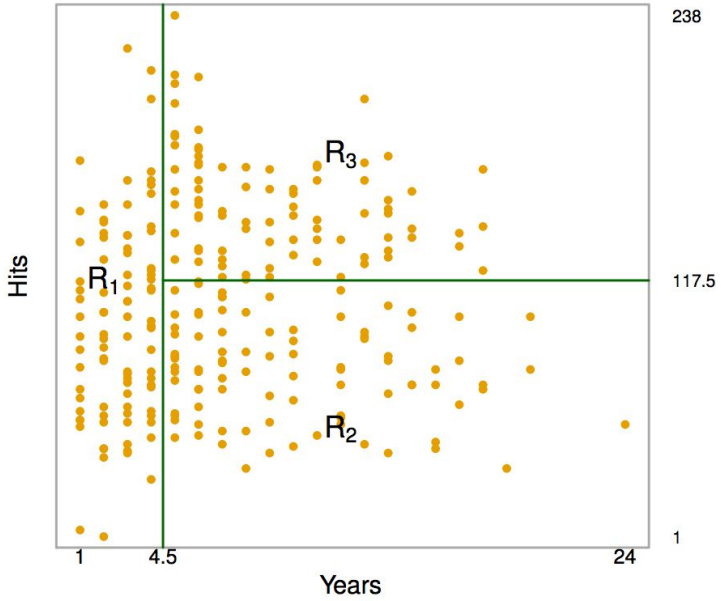
- Segments the predictor space into smaller regions
- Collectively known as decision trees
  - Regression trees
  - Classification trees
- Example: salary for baseball players
  - Task: predict a salary based on: hits & years of experience
  - Salary coded for low (blue / green) to high (yellow / red)
  - Data (right) results in tree:





Alternative Representations



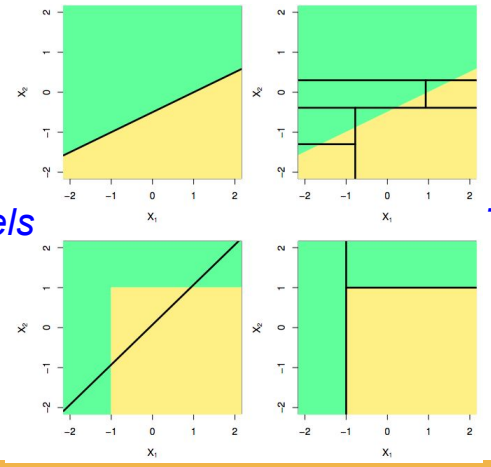


Advantages & Disadvantages

- Advantages
  - Great for interpretation
  - Elegantly handles irrelevant features
- Disadvantages
  - Mediocre individual performance (used in confederation for better performance)
  - Prone to overfitting
- Other random stuff
  - “The most common technique” for classification?
  - Can see performance improvements with bagging, random forests, boosting

Trees vs. Linear Models

- Advantages of trees
  - Trees are easy to explain and interpret
  - Trees can handle qualitative predictors gracefully
- Disadvantage of trees: generally weak performance
  - Good performance requires “boxy” relationships
  - Improve performance via:
    - Bagging
    - Random Forests
    - Boosting



Algorithm: Recursive Splitting

- Goal: split trees so that the splits are better organized
  - Top-down: start at root of tree (all data) ... then splits
  - Greedy: at each step, split on the best predictor with no backtracking
  - Caution: greedy algorithms sometimes produce results which are not globally optimal
- Description
  - Select feature X_j which leads to greatest reduction in classification error
  - Recursively apply algorithm to subtrees
  - Stop splitting when:
    - Subtree is PURE — all instances in subtree belong to the same class
    - There are no more features to split on
    - Subtrees have a number of observations (eg. 5)
- Predictions: use the majority (plurality?) of observations in the subtree

Quantifying Error

- Many classification problems use “classification error rate”

$$E = 1 - \max_k(\hat{p}_{mk})$$

- p_{mk} = portion of training observations in m^{th} region from k^{th} class
- In practice, classification error rate is not sufficiently sensitive

Entropy for Quantifying Purity

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

- Measures total variance among *K* classes
- Names: (Shannon) Entropy

```
def entropy(Y): # Y is a list
    from math import log

    size = len(Y)
    counts = dict()
    for y in Y:
        if y not in counts:
            counts[y] = 0.
        counts[y] += 1.
    entropy = 0.
    for key in counts:
        prob = counts[key] / size
        entropy -= prob * log(prob,2)
    return entropy
```

Entropy — Example & Questions

- Entropy of the “animals” data?
- Entropy if we remove “chicken” and “swan”?
- Entropy if only “scorpion” and “starfish” are left?

animals

name	convering	eggs	lives_in	oxygenates	legs	tail	category
aardvark	hair	0	ground	air	4	0	mammal
antelope	hair	0	ground	air	4	1	mammal
bass	scales	1	water	water	0	1	fish
carp	scales	1	water	water	0	1	fish
chicken	feathers	1	ground	air	2	1	bird
scorpion	exoskeleton	0	ground	air	8	1	invert
starfish	skin	1	water	water	5	0	invert
swan	feathers	1	air	air	2	1	bird

Entropy Alternative: Gini Index

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

- Also measures total variance among *K* classes
  - Observations from a single class —> Gini index will be 0
  - Takes a small value if p_{mk}s are close to 0 or close to 1
- Also called "*node purity*" or "*Gini impurity*"

Data Splitting — The Right Thing?

- Algorithm Overview
  - Measure entropy of data
  - Split the data into subtrees; measure entropy of subtrees
  - Determine whether split results in lower entropy
- Implementation module: split_data
  - Input: X = list of lists; axis = index in x to split on; value of axis
  - Process: find all x ∈ X such that x[axis] == value
  - Return x[0..axis, axis+1..]

split_data

```
def split_data(X, axis, value):
    return_data = []

    for x in X:
        if x[axis] == value:
            reduced_x = x[:axis]
            reduced_x.extend(x[axis+1:])
            return_data.append(reduced_x)
    return return_data
```

Exercise: Change this function

- Accept Y (target) values
- Return Y values split the same way as X

Data Splitting — Choose Feature

- Measure *Information Gain*
  - If we split a feature, what is the change in entropy?
  - If the change is positive (i.e. lower entropy / Gini impurity), this is information gain
- Choose the highest information gain for each tree/subtree
  - sklearn uses binary splitting —> binary tree
  - We will use n-ary splitting: one child for each value in train_x

Implementation: Choose Feature

```
def choose_feature(X, Y):
    entropy = entropy(Y)                    # Get the pre-split entropy
    best_information_gain = 0.
    best_feature = -1
    for i in range(len(X[0])):              # For each feature
        feature_list = [x[i] for x in X]
        values = set(feature_list)          # ... get unique values
        entropy_i = 0.
        for value in values:                # ... split the data
            sub_x, sub_y = split_data(X, Y, i, value)
            prob = len(sub_x) / float(len(X))
            entropy_i += prob * entropy(sub_y)
        info_gain = entropy - entropy_i     # ... determine: good split?
        if info_gain > best_information_gain:  # Best split?
            best_information_gain = info_gain
            best_feature = i
    return best_feature
```

Exercise: “animals”

- Get the animals.csv file
  - Split into X and Y
  - Y = class (mammal, bird, etc.)
  - Do not include animal name. (Why?)
- Run choose_feature against animals.csv data
- Which is the best feature?

One Problem

- Problem:
  - Goal: divide data to get pure nodes
  - Algorithm can exhaust all features
  - What if we have exhausted all features but nodes are not pure?
- Solution:
  - Return the majority of labels in a node
  - Return plurality if no clear majority
- Exercise: implement this

Lab

- Lab 3 on Canvas
- Complete implementation of decision_tree class:
  - 1: Use functions above (entropy, split_data, choose_feature) — modified for class
  - 2: Add (recursive) “fit” function to determine where to split

Problems with Performance

- Performance vs. tree size
  - Recursive splitting eventually overfits
  - Sometimes smaller trees (fewer splits) have higher performance
- Solution # 1
  - Grow the tree while entropy / Gini impurity decrease exceeds some high threshold
  - Bad idea for a greedy algorithm
  - But worthless splits early on may give excellent splits later
- Solution # 2
  - Grow a (very) large tree
  - Prune the tree to get a subtree

Cost Complexity Pruning

- An algorithm for tree pruning
  - Also known as *weakest link pruning*
  - Depends on non-negative tuning parameter, α — trade-off between complexity of subtree and fit to training observations
- Driven by equation

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

- Considerations:
  - $|T|$ is number of leaves in the (sub-)tree T
  - R_m is the rectangle of m^{th} leaf node
  - Select optimal α through cross-validation

Putting It All Together

- Grow the tree
  - Use recursive binary splitting to grow a large tree
  - Constructed from training data
  - Stopping when leaves have no more than X observations
- Prune the tree
  - Use cost complexity pruning to get the best set of subtrees
  - Based on α
- Choose the best α
  - Use k-fold cross-validation
  - Average the results and pick α to minimize average (training) error
- Best subtree corresponds to pruned subtree for α

Next Time

- Hands-on with Decision Trees
- Extensions to Trees:
  - Bagging
  - Random Forest
  - Bagging