Let’s assume something broke down and you are trying to fix it....
Scenario

...and you have no clue what on earth you gonna do that!
So you call your engineer friend and ask him what to do....
And he gives you a recipe to follow

Engineering Flowchart

**DOES IT MOVE?**

- **No**
  - Should it?
    - **No**
      - No Problem
    - **Yes**
      - WD-40

- **Yes**
  - Should it?
    - **Yes**
      - WD-40
    - **No**
      - No Problem

Tape
Scenario 2

How college works

Graduate High School

Go to college

Study hard

Take exams

Study harder

Take more exams

Have ulcers treated

Take final exam

Graduate College

Don't go to college

Work Drive-thru at fast food joint

GraphJam.com
Now seriously...

- The previous charts/figures are called decision trees.
- They are very useful tools to classify and decide.
Chapter 7: “Modeling with Decision Trees”
What type of fruit is it?!

- color = green?
  - No
    - color = red?
      - No
        - shape = round?
          - No: Banana
          - Yes: diameter > 4in?
            - No: Lemon
            - Yes: Grapefruit
  - Yes
    - diameter > 6in?
      - No
        - diameter > 2in?
          - No: Grape
          - Yes: Cherry
      - Yes: Watermelon
How Decision Trees?

• Easy to understand and build
• You can understand the reasoning process by just looking at it.
• Easy to implement and even convert to a series of if-else statements.
Example: Predicting Signups

• You have a new service and you are releasing it for free for a month.
• You are recording all the log data and collecting as much information as possible
• You aim to see who will subscribe and who wouldn’t.
## Log data

**Table 7-1. User behavior and final purchase decision for a web site**

<table>
<thead>
<tr>
<th>Referrer</th>
<th>Location</th>
<th>Read FAQ</th>
<th>Pages viewed</th>
<th>Service chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slashdot</td>
<td>USA</td>
<td>Yes</td>
<td>18</td>
<td>None</td>
</tr>
<tr>
<td>Google</td>
<td>France</td>
<td>Yes</td>
<td>23</td>
<td>Premium</td>
</tr>
<tr>
<td>Digg</td>
<td>USA</td>
<td>Yes</td>
<td>24</td>
<td>Basic</td>
</tr>
<tr>
<td>Kiwitobes</td>
<td>France</td>
<td>Yes</td>
<td>23</td>
<td>Basic</td>
</tr>
<tr>
<td>Google</td>
<td>UK</td>
<td>No</td>
<td>21</td>
<td>Premium</td>
</tr>
<tr>
<td>(direct)</td>
<td>New Zealand</td>
<td>No</td>
<td>12</td>
<td>None</td>
</tr>
<tr>
<td>(direct)</td>
<td>UK</td>
<td>No</td>
<td>21</td>
<td>Basic</td>
</tr>
<tr>
<td>Google</td>
<td>USA</td>
<td>No</td>
<td>24</td>
<td>Premium</td>
</tr>
<tr>
<td>Slashdot</td>
<td>France</td>
<td>Yes</td>
<td>19</td>
<td>None</td>
</tr>
<tr>
<td>Digg</td>
<td>USA</td>
<td>No</td>
<td>18</td>
<td>None</td>
</tr>
<tr>
<td>Google</td>
<td>UK</td>
<td>No</td>
<td>18</td>
<td>None</td>
</tr>
<tr>
<td>Kiwitobes</td>
<td>UK</td>
<td>No</td>
<td>19</td>
<td>None</td>
</tr>
<tr>
<td>Digg</td>
<td>New Zealand</td>
<td>Yes</td>
<td>12</td>
<td>Basic</td>
</tr>
<tr>
<td>Google</td>
<td>UK</td>
<td>Yes</td>
<td>18</td>
<td>Basic</td>
</tr>
<tr>
<td>Kiwitobes</td>
<td>France</td>
<td>Yes</td>
<td>19</td>
<td>Basic</td>
</tr>
</tbody>
</table>

slashdot  UK  no  21  None

**add this line to make the table match the data / code**
Can we predict who will sign up for basic or premium service at the end of the trial period based on the data we’ve collected?
Loading the data

my_data=
['slashdot', 'USA', 'yes', 18, 'None'],
[google', 'France', 'yes', 23, 'Premium'],
['digg', 'USA', 'yes', 24, 'Basic'],
['kiwitobes', 'France', 'yes', 23, 'Basic'],
[google', 'UK', 'no', 21, 'Premium'],
[(direct), 'New Zealand', 'no', 12, 'None'],
[(direct), 'UK', 'no', 21, 'Basic'],
[google', 'USA', 'no', 24, 'Premium'],
['slashdot', 'France', 'yes', 19, 'None'],
['digg', 'USA', 'no', 18, 'None'],
[google', 'UK', 'no', 18, 'None'],
['kiwitobes', 'UK', 'no', 19, 'None'],
['digg', 'New Zealand', 'yes', 12, 'Basic'],
['slashdot', 'UK', 'no', 21, 'None'],
[google', 'UK', 'yes', 18, 'Basic'],
['kiwitobes', 'France', 'yes', 19, 'Basic']
[‘slashdot’, ‘UK’, ‘no’, ‘21’, ‘None’] ]
Decision Trees

- Simple to build and train.
- Once you have it answering questions/classifying is as easy as following branches.
- Tracing back from the node where you ended up with gives a rationale for the final decision.
Building a Decision Tree

```python
class decisionnode:
    def __init__(self, col=-1, value=None, results=None, tb=None, fb=None):
        self.col = col
        self.value = value
        self.results = results
        self.tb = tb
        self.fb = fb
```

- **Col** is the column index of the criteria to be tested.
- **Value** is the value that the column must match to get a true result.
- **tb** and **fb** are *decisionnodes*, which are the next nodes in the tree if the result is true or false, respectively.
- **results** stores a dictionary of results for this branch. This is None for everything except endpoints.
Training the tree

• You have the bulk of data.
• Slowly, you try to find the variables the divide up the data.
• We need to pick the best variable to do so.

• Employs CART (Classification And Regression Trees).
```python
"divideset" is a function that divides a set on a specific column. It can handle numeric or nominal values.

```python
def divideset(rows, column, value):
    # Make a function that tells us if a row is in the first group (true) or the second group (false)
    split_function=None
    if isinstance(value, int) or isinstance(value, float):
        split_function = lambda row: row[column] >= value
    else:
        split_function = lambda row: row[column] == value

    # Divide the rows into two sets and return them
    set1 = [row for row in rows if split_function(row)]
    set2 = [row for row in rows if not split_function(row)]

    return (set1, set2)
```
Is reading the FAQ a good descriptor for subscriptions?

Table 7-1. User behavior and final purchase decision for a web site

<table>
<thead>
<tr>
<th>Referrer</th>
<th>Location</th>
<th>Read FAQ</th>
<th>Pages viewed</th>
<th>Service chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slashdot</td>
<td>USA</td>
<td>Yes</td>
<td>18</td>
<td>None</td>
</tr>
<tr>
<td>Google</td>
<td>France</td>
<td>Yes</td>
<td>23</td>
<td>Premium</td>
</tr>
<tr>
<td>Digg</td>
<td>USA</td>
<td>Yes</td>
<td>24</td>
<td>Basic</td>
</tr>
<tr>
<td>Kiwitobes</td>
<td>France</td>
<td>Yes</td>
<td>23</td>
<td>Basic</td>
</tr>
<tr>
<td>Google</td>
<td>UK</td>
<td>No</td>
<td>21</td>
<td>Premium</td>
</tr>
<tr>
<td>(direct)</td>
<td>New Zealand</td>
<td>No</td>
<td>12</td>
<td>None</td>
</tr>
<tr>
<td>(direct)</td>
<td>UK</td>
<td>No</td>
<td>21</td>
<td>Basic</td>
</tr>
<tr>
<td>Google</td>
<td>USA</td>
<td>No</td>
<td>24</td>
<td>Premium</td>
</tr>
<tr>
<td>Slashdot</td>
<td>France</td>
<td>Yes</td>
<td>19</td>
<td>None</td>
</tr>
<tr>
<td>Digg</td>
<td>USA</td>
<td>No</td>
<td>18</td>
<td>None</td>
</tr>
<tr>
<td>Google</td>
<td>UK</td>
<td>No</td>
<td>18</td>
<td>None</td>
</tr>
<tr>
<td>Kiwitobes</td>
<td>UK</td>
<td>No</td>
<td>19</td>
<td>None</td>
</tr>
<tr>
<td>Digg</td>
<td>New Zealand</td>
<td>Yes</td>
<td>12</td>
<td>Basic</td>
</tr>
<tr>
<td>Google</td>
<td>UK</td>
<td>Yes</td>
<td>18</td>
<td>Basic</td>
</tr>
<tr>
<td>Kiwitobes</td>
<td>France</td>
<td>Yes</td>
<td>19</td>
<td>Basic</td>
</tr>
</tbody>
</table>
Is reading the FAQ a good descriptor for subscriptions?

```python
>>> import treepredict
>>> treepredict.divideset(treepredict.my_data, 2, 'yes')
([['slashdot', 'USA', 'yes', 18, 'None'],
  ['google', 'France', 'yes', 23, 'Premium'],
  ['digg', 'USA', 'yes', 24, 'Basic'],
  ['kiwitobes', 'France', 'yes', 23, 'Basic'],
  ['slashdot', 'France', 'yes', 19, 'None'],
  ['digg', 'New Zealand', 'yes', 12, 'Basic'],
  ['google', 'UK', 'yes', 18, 'Basic'],
  ['kiwitobes', 'France', 'yes', 19, 'Basic']],
[['google', 'UK', 'no', 21, 'Premium'],
 '(direct)', 'New Zealand', 'no', 12, 'None'],
 '(direct)', 'UK', 'no', 21, 'Basic'],
 ['google', 'USA', 'no', 24, 'Premium'],
 ['digg', 'USA', 'no', 18, 'None'],
 ['google', 'UK', 'no', 18, 'None'],
 ['kiwitobes', 'UK', 'no', 19, 'None'],
 ['slashdot', 'UK', 'no', 21, 'None']])
```

Eyeballing the result, it doesn’t appear FAQ is a good predictor.
Is reading the FAQ a good descriptor for subscriptions?

Table 7-2. Outcomes based on Read FAQ column values

<table>
<thead>
<tr>
<th>True</th>
<th></th>
<th>False</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td>Premium</td>
<td></td>
</tr>
<tr>
<td>Premium</td>
<td></td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td></td>
<td>Basic</td>
<td></td>
</tr>
<tr>
<td>None</td>
<td></td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td></td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td></td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

Eyeballing the result, it doesn’t appear FAQ is a good predictor.
Measuring the Mix

- We need a way to measure how mixed a set is.
- We choose the variable that creates two sets with the least possible mixing.
- We will use:
  - Gini impurity
  - Entropy
Counting Unique

```
def uniquecounts(rows):
    results={}  # Create counts of possible results (the last column of # each row is the result)
    for row in rows:
        r=row[len(row)-1]  # The result is the last column
        if r not in results: results[r]=0
        results[r]+=1
    return results
```

- None: 2
- Basic: 4
- Premium: 1
Gini impurity

• It’s a way to measure homogeneity of a set.
• It’s the expected error rate if one of the results of the set is randomly applied to one of the items in the set.

• If: 1 category = error rate is 0
• If: 4 possible results = error rate is 75%

3 out of the 4 chances it will be placed in a wrong result/category
# Probability that a randomly placed item will be in the wrong category

def giniimpurity(rows):
    total=len(rows)
    counts=uniquecounts(rows)
    imp=0
    for k1 in counts:
        p1=float(counts[k1])/total
        for k2 in counts:
            if k1==k2: continue
            p2=float(counts[k2])/total
            imp+=p1*p2
    return imp
Gini impurity

```python
>>> treepredict.giniimpurity(treepredict.my_data)
0.6328125
>>> set1,set2=treepredict.divideset(treepredict.my_data,2,'yes')
>>> treepredict.giniimpurity(set1)
0.53125
>>> treepredict.giniimpurity(set2)
0.53125
>>> set1
[['slashdot', 'USA', 'yes', 18, 'None'],
 ['google', 'France', 'yes', 23, 'Premium'],
 ['digg', 'USA', 'yes', 24, 'Basic'],
 ['kiwitobes', 'France', 'yes', 23, 'Basic'],
 ['slashdot', 'France', 'yes', 19, 'None'],
 ['digg', 'New Zealand', 'yes', 12, 'Basic'],
 ['google', 'UK', 'yes', 18, 'Basic'],
 ['kiwitobes', 'France', 'yes', 19, 'Basic']]
```

$$I_G = \frac{\#\text{none}}{\#\text{total}}(1 - \frac{\#\text{none}}{\#\text{total}}) + \frac{\#\text{basic}}{\#\text{total}}(1 - \frac{\#\text{basic}}{\#\text{total}}) + \frac{\#\text{premium}}{\#\text{total}}(1 - \frac{\#\text{premium}}{\#\text{total}})$$

$$= \frac{2}{8}(\frac{6}{8}) + \frac{5}{8}(\frac{3}{8}) + \frac{1}{8}(\frac{7}{8})$$

$$= 0.53125$$
Entropy

- It is the measure of how much there is a disorder in the set.

\[
p(i) = \frac{\text{frequency(outcome)}}{\text{count(total rows)}}
\]

Entropy = sum of \( p(i) \times \log(p(i)) \) for all outcomes

- We aim to reduce the Entropy.
# Entropy is the sum of p(x)log(p(x)) across all the different possible results

def entropy(rows):

    from math import log
    log2=lambda x:log(x)/log(2)
    results=uniquecounts(rows)

    # Now calculate the entropy
    ent=0.0
    for r in results.keys():
        p=float(results[r])/len(rows)
        ent=ent-p*log2(p)

    return ent
Entropy

```python
>>> treepredict.entropy(treepredict.my_data)
1.5052408149441479
>>> set1, set2 = treepredict.divideset(treepredict.my_data, 2, 'yes')
>>> treepredict.entropy(set1)
1.29879406953985
>>> treepredict.entropy(set2)
1.29879406953985
>>> set1
[['slashdot', 'USA', 'yes', 18, 'None'],
 ['google', 'France', 'yes', 23, 'Premium'],
 ['digg', 'USA', 'yes', 24, 'Basic'],
 ['kiwitobes', 'France', 'yes', 23, 'Basic'],
 ['slashdot', 'France', 'yes', 19, 'None'],
 ['digg', 'New Zealand', 'yes', 12, 'Basic'],
 ['google', 'UK', 'yes', 18, 'Basic'],
 ['kiwitobes', 'France', 'yes', 19, 'Basic']]

   = 0.5 + 0.423 + 0.375
   = 1.298
Now we know how to measure, let’s build the tree...
Recursive Tree Building

Algorithm:
1. Calculate the entropy of the whole set.
2. Get the possible ways to divide the group by possible values of variables.
3. Divide the group according to the variable.
4. Calculate the entropy of the two new groups.
5. Calculate the information gain.
6. Pick the one with the highest information gain and divide according to it.
7. Send both parts to same function and repeat.
Recursive Tree Building

- Maximize *Information Gain*, the difference between the entropy of the current set and the weighted average entropy of the two new groups
  - $\max(H-H(i))$
- Recursively repeat on each branch of tree until Information Gain is $< 0$
  - i.e., stop when you’re creating more disorder
Recursive Tree Building

```python
def buildtree(rows, scoref=entropy):
    if len(rows) == 0:
        return decisionnode()
    current_score = scoref(rows)

    # Set up some variables to track the best criteria
    best_gain = 0.0
    best_criteria = None
    best_sets = None

    column_count = len(rows[0]) - 1
    for col in range(0, column_count):
        # Generate the list of different values in
        # this column
        column_values = {}
        for row in rows:
            column_values[row[col]] = 1
        # Now try dividing the rows up for each value
        # in this column
        for value in column_values.keys():
            (set1, set2) = divideSet(rows, col, value)

            # Information gain
            p = float(len(set1)) / len(rows)
            gain = current_score - p * scoref(set1) - (1 - p) * scoref(set2)
            if gain > best_gain and len(set1) > 0 and len(set2) > 0:
                best_gain = gain
                best_criteria = (col, value)
                best_sets = (set1, set2)

        # Create the subbranches
        if best_gain > 0:
            trueBranch = buildtree(best_sets[0])
            falseBranch = buildtree(best_sets[1])
            return decisionnode(col=best_criteria[0], value=best_criteria[1],
                                 tb=trueBranch, fb=falseBranch)
        else:
            return decisionnode(results=uniqueCounts(rows))
```
Average Weighted Entropy

• You calculate it for every pair by multiplying each set’s entropy by the fraction of the items that ended up in this set.
```python
>>> treepredict.entropy(treepredict.my_data)
1.5052408149441479
>>> set1, set2 = treepredict.divideset(treepredict.my_data, 0, 'slashdot')
>>> treepredict.entropy(set1)
0.0
>>> set1
[['slashdot', 'USA', 'yes', 18, 'None'], ['slashdot', 'France', 'yes', 19, 'None'], ['slashdot', 'UK', 'no', 21, 'None']]
>>> treepredict.entropy(set2)
1.5052408149441479
```
Now you have a trained tree!....let’s see it!
Displaying the Tree

```python
>>> tree = treepredict.buildtree(treepredict.my_data)
>>> treepredict.printtree(tree)
0: google?
   T-> 3:21?
      T-> {'Premium': 3}
      F-> 2:yes?
         T-> {'Basic': 1}
         F-> {'None': 1}
   F-> 0: slashdot?
      T-> {'None': 3}
      F-> 2:yes?
         T-> {'Basic': 4}
         F-> 3:21?
            T-> {'Basic': 1}
            F-> {'None': 3}

```
Graphing the Tree

```python
>>> treepredict.drawtree(tree, jpeg='treeview.jpg')
```
Now we see it....it’s pretty...

Let’s utilize it!
Classifying Observations

- You provide the classifier with your observations → give you prediction

```python
def classify(observation, tree):
    if tree.results != None:
        return tree.results
    else:
        v = observation[tree.col]
        branch = None
        if isinstance(v, int) or isinstance(v, float):
            if v >= tree.value:
                branch = tree.tb
            else:
                branch = tree.fb
        else:
            if v == tree.value:
                branch = tree.tb
            else:
                branch = tree.fb
        return classify(observation, branch)
```

```
>>> reload(treepredict)
<module 'treepredict' from 'treepredict.pyc'>
>>> treepredict.classify([('direct','USA','yes',5),tree]
{'Basic': 4}
```
Problem: It’s too trained!

- This is called overfitting.
- It becomes too specific to the training data.
- It becomes more certain than it should be.
Solution: Prune it!

• One possibility is to stop splitting when entropy reduction is not big enough.

• Drawback:
  • Reduction little by little by many subsequent small splits.
Pruning

Checking pairs of nodes that have a common parent to see if merging them would increase entropy by less than a specified threshold.
>>> treepredict.printtree(tree)
0: google?
  T-> 3: 21?
    T-> {'Premium': 3}
    F-> 2: yes?
      T-> {'Basic': 1}
      F-> {'None': 1}
  F-> 0: slashdot?
    T-> {'None': 3}
    F-> 2: yes?
      T-> {'Basic': 4}
      F-> 3: 21?
        T-> {'Basic': 1}
        F-> {'None': 1}
    F-> {'None': 6, 'Basic': 5}
>>> treepredict.prune(tree, 0.1)
>>> treepredict.printtree(tree)
(same tree)
>>> treepredict.prune(tree, 0.5)
>>> treepredict.printtree(tree)
(same tree)
>>> treepredict.prune(tree, 0.75)
>>> treepredict.printtree(tree)
(same tree)
>>> treepredict.prune(tree, 0.90)
>>> treepredict.printtree(tree)
0: google?
  T-> 3: 21?
    T-> {'Premium': 3}
    F-> 2: yes?
      T-> {'Basic': 1}
      F-> {'None': 1}
    F-> {'None': 6, 'Basic': 5}

>>> treepredict.drawtree(tree, jpeg='pruned-tree.jpeg')
Advantage: Dealing with Missing Data

• Upon finding a missing observation we adapt the prediction function to handle this.
• If the data missing is to decide which branch → follow both branches
• Instead of counting the results → weight the results
Missing Data

>>> # reminder: referer, location, FAQ, pages
>>> treepredict.mdclassify(['google', None, 'yes', None], tree)  
{'Premium': 2.25, 'Basic': 0.25}
>>> treepredict.mdclassify(['google', 'France', None, None], tree)  
{'None': 0.125, 'Premium': 2.25, 'Basic': 0.125}
>>> treepredict.mdclassify(['google', None, None, '14'], tree)  
{'None': 0.5, 'Basic': 0.5}

ex1: location & pages unknown
FAQ=yes, so 1 outcome
faq_weight = 1/1
basic = 1 * 1.0
if pages >20 then 3 outcomes
else 1 outcome
pages_true_weight=3/4
pages_false_weight=1/4
premium = 3 * 3/4, basic = 1.0 * 1/4

ex2: FAQ & pages unknown
if FAQ then 1 outcome
else 1 outcome
faq_true_weight = 1/2
faq_false_weight = 1/2
none = 1 * 0.5, basic = 1 * 0.5
if pages >20 then 3 outcomes
else 2 outcomes (each with weight = 0.5)
pages_true_weight=3/4
pages_false_weight=1/4
premium = 3 * 3/4, basic = 0.5 * 1/4, none = 0.5 * 1/4
Dealing with Numerical Outcomes

- height (in) = \{56, 59, 59, 61, 62, 74, 76, 76, 78\}
  - this list could be categorized as:
    - short (<65”)
    - tall (>72”)
  - or we could use the integers as values
    - we would use variance as our measure of dispersion, not Gini Impurity or Entropy

→ Variance is used to show that some numbers are close to each other and some are not
When use Decision Trees?

• pros
  – Easy to interpret
  – predictive & descriptive
  – categories + numerical data
  – can have nodes with many outcomes (probabilistic outcomes)

• cons
  – only <, > operators on numerical outcomes
  – doesn’t handle many {inputs|outcomes} well
  – can’t uncover complex relationships between inputs
Assignment 7

• Use the Zillow API in the example in the book to build a working decision tree that is going to help me pick a house to buy in Norfolk

• **Deadline**: Next week