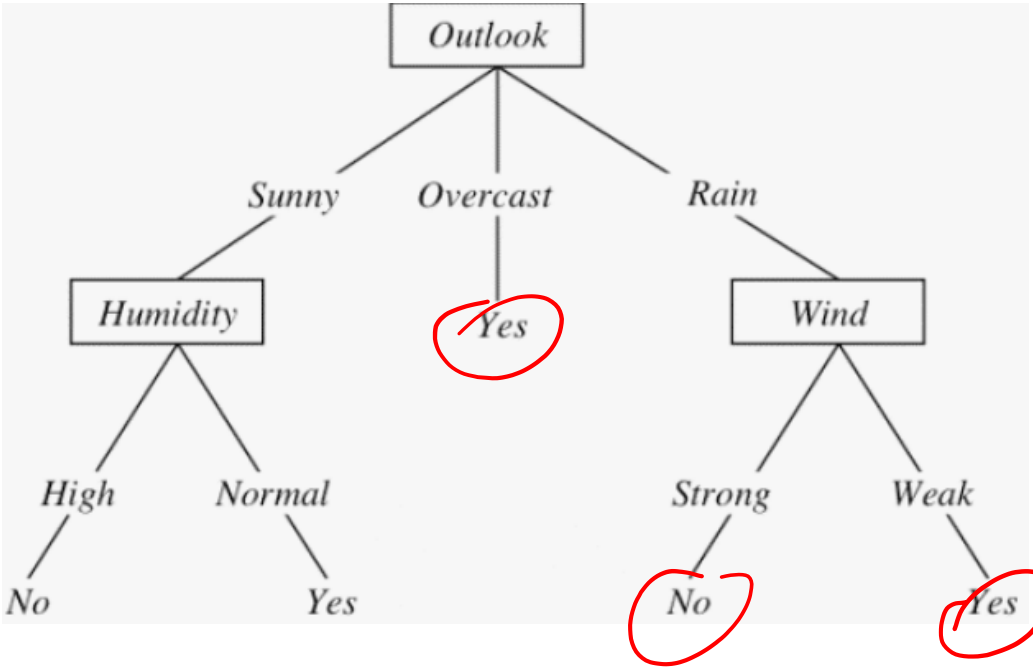
Decision Trees

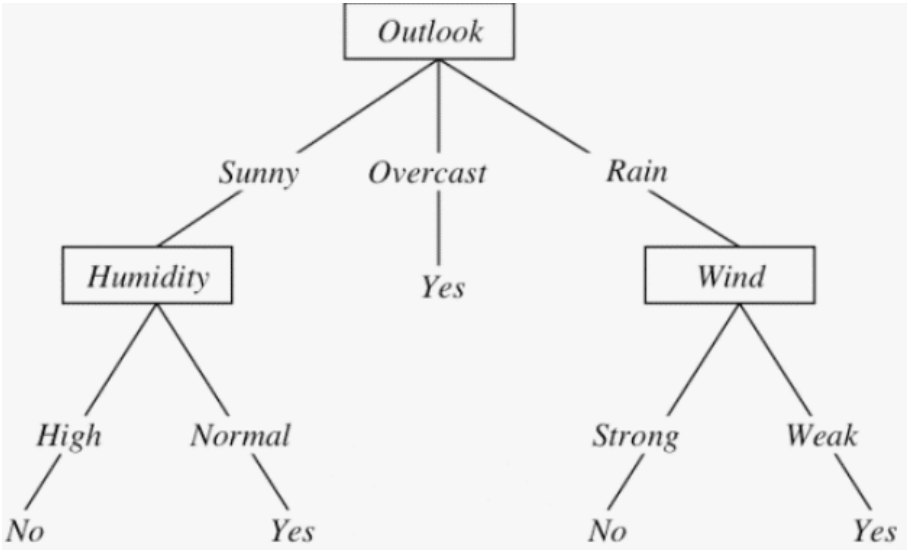
Decision Trees

- Building Decision Trees: Hierarchical and Recursive partitioning of the feature space
- Example: Play tennis?

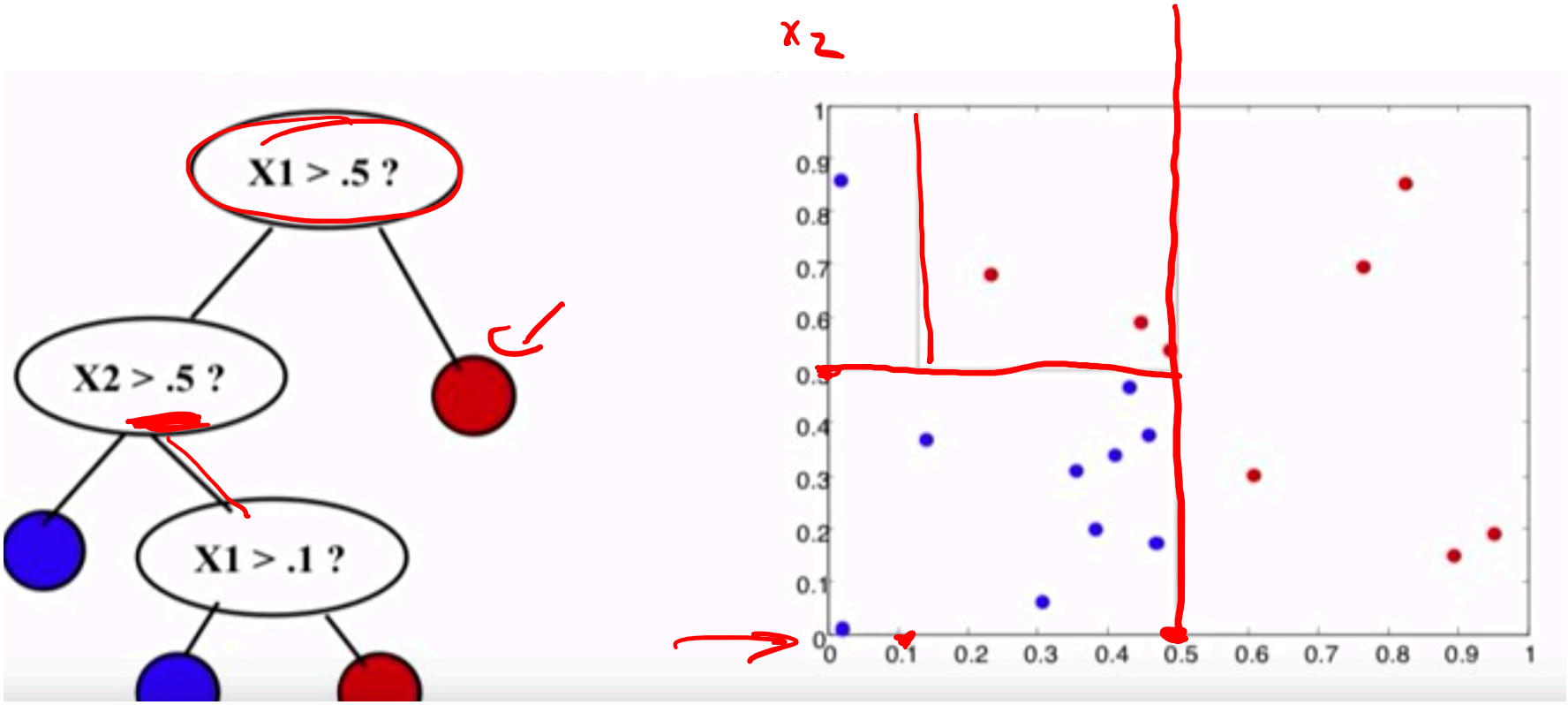


Decision Trees

- Representation
  - Each internal node tests an attribute
  - Each branch is an attribute value
  - Each leaf assigns a classification



Context of Lines



Consistent Decision Trees

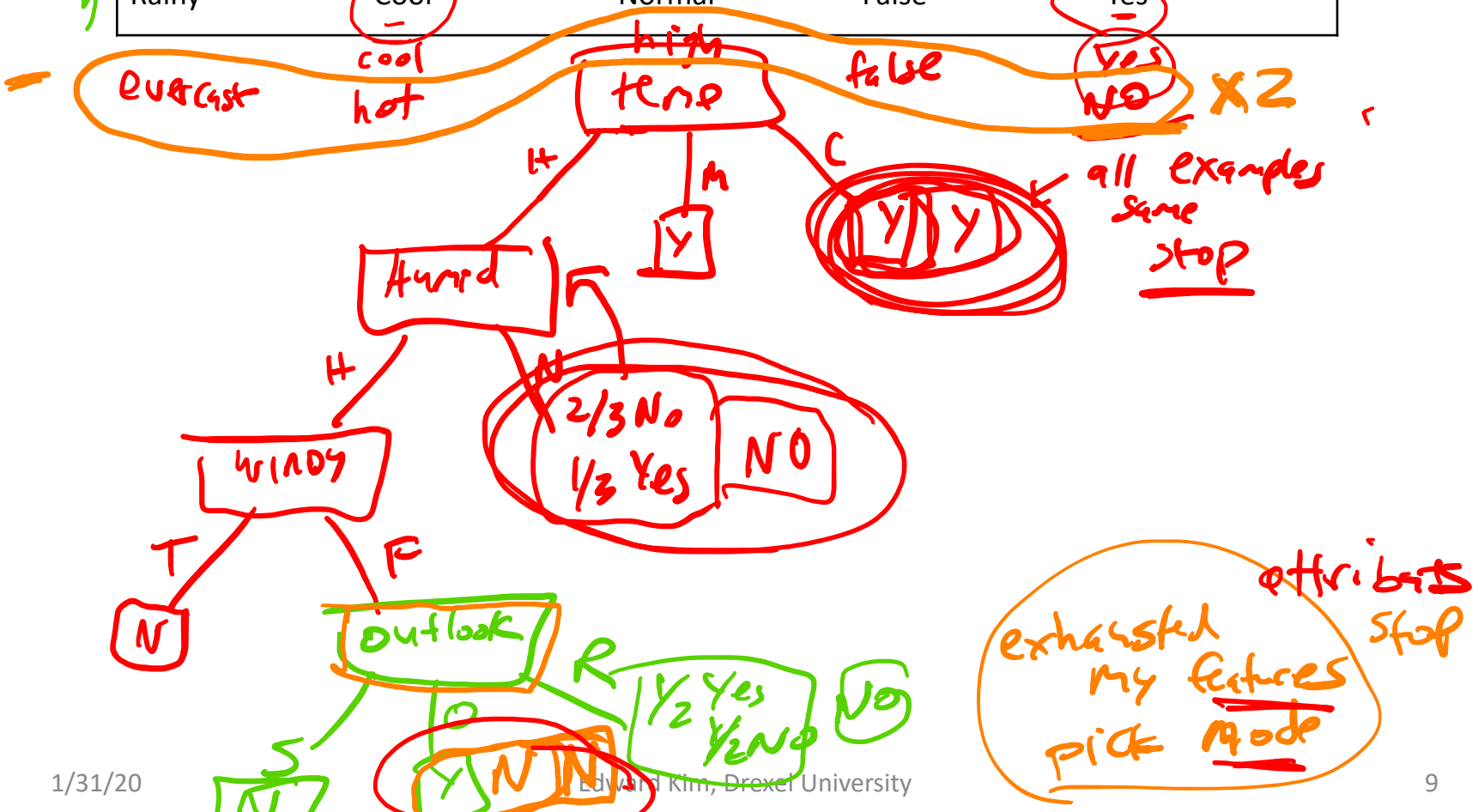
- If we have D binary features and a binary classification system then there are 2^{2^D} decision trees
  - Yikes
- If our data is noiseless and without any randomness, then there is at least one trivially **consistent** decision tree for the data set
  - Fits the data perfectly
  - A tree that splits the data with all pure leaves is called consistent with the data. This is always possible when no two samples (x,y),(x′,y′) have different outcomes y≠y′ but identical features x=x′
- In fact there's many potential ones
- But we'd prefer to find a *compact* one.
  - That is one with a minimal height.

Example

- Example: Situations for which I will/won't play tennis
- **Exercise:** Let's (painfully) build a consistent decision tree from these examples
  - There's a lot of different possible ones

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes



Decision Tree Learning

- In practice it may be impossible to build a consistent decision tree
  - There may be noise
  - And/or there may be randomness
  - Or we don't have enough information/features.
- Ultimately we'd like to find a small (compact) tree consistent with as many training examples as possible.

Decision Tree Learning

- Idea: (recursively) choose "most significant" attribute/feature as root of (sub)tree
  - A greedy algorithm

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Choosing an Attribute

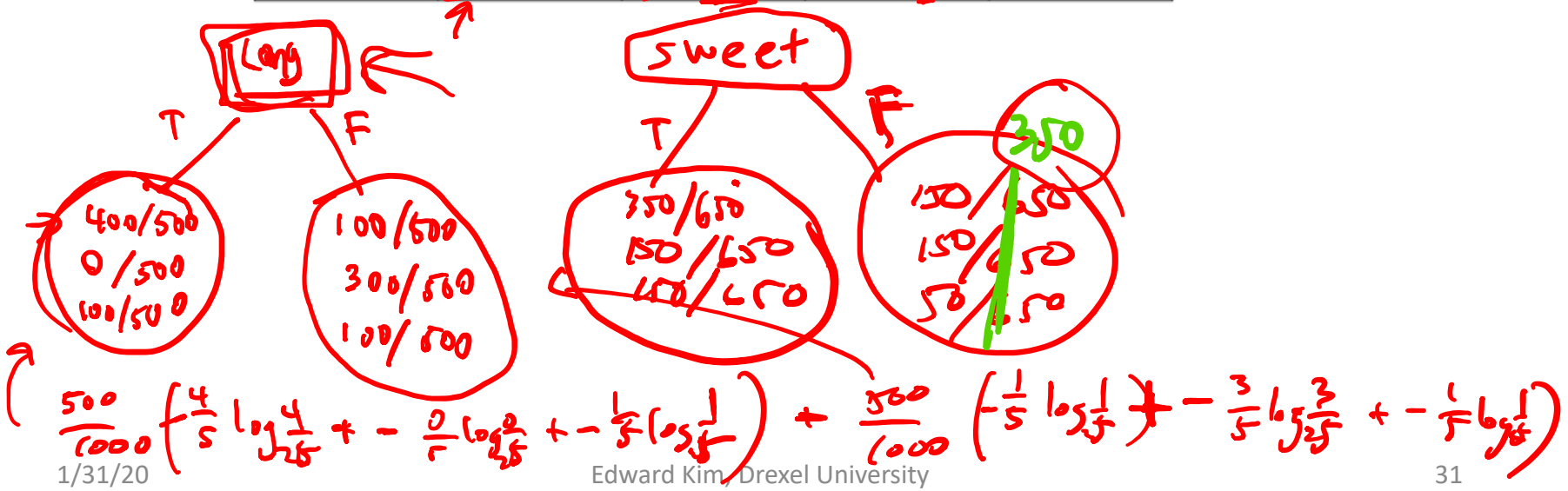
- Idea: A good attribute splits the examples into subsets that contain (ideally) observations from just one class.
- Which feature is the most useful then for splitting the data at first?
  - Long?
  - Sweet?
  - Yellow?

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	**500**	**650**	**800**	**1000**

Choosing an Attribute

- Idea: A good attribute splits the examples into subsets that contain (ideally) observations from just one class.

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	**500**	**650**	**800**	**1000**



Example

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	**500**	**650**	**800**	**1000**

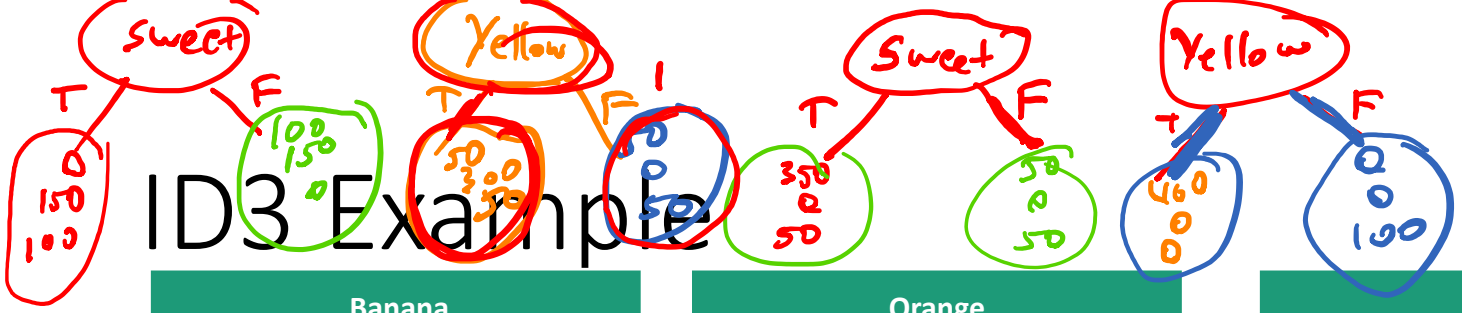
- Building a tree completely using information gain is called the ID3 decision tree algorithm
- Let's build this ID3 DT all the way out!
- To do this we'll need a little more information

Banana			
Long	*Sweet*	*Yellow*	*Count*
F	F	F	50
F	F	T	50
F	T	F	0
F	T	T	0
T	F	F	0
T	F	T	50
T	T	F	0
T	T	T	350

Orange			
Long	*Sweet*	*Yellow*	*Count*
F	F	F	0
F	F	T	150
F	T	F	0
F	T	T	150
T	F	F	0
T	F	T	0
T	T	F	0
T	T	T	0

Other			
Long	*Sweet*	*Yellow*	*Count*
F	F	F	0
F	F	T	0
F	T	F	50
F	T	T	50
T	F	F	50
T	F	T	0
T	T	F	50
T	T	T	0

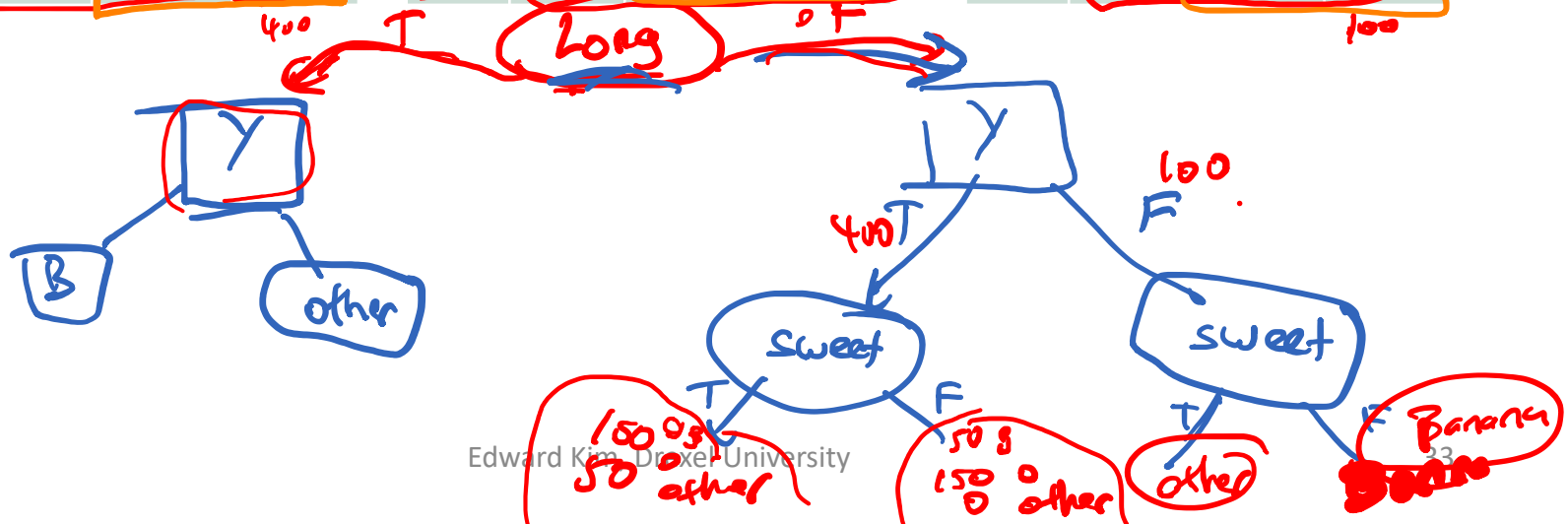
ID3 Example



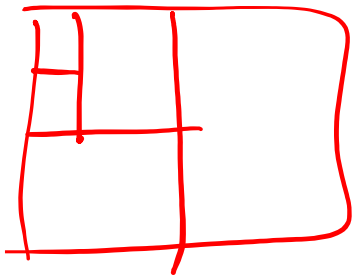
Banana			
~~Long~~	Sweet	Yellow	Count
F	F	F	50
F	F	T	50
F	T	F	0
F	T	T	0
T	F	F	0
T	F	T	50
T	T	F	0
T	T	T	350

Orange			
Long	Sweet	Yellow	Count
F	F	F	0
F	F	T	150
F	T	F	0
F	T	T	150
T	F	F	0
T	F	T	0
T	T	F	0
T	T	T	0

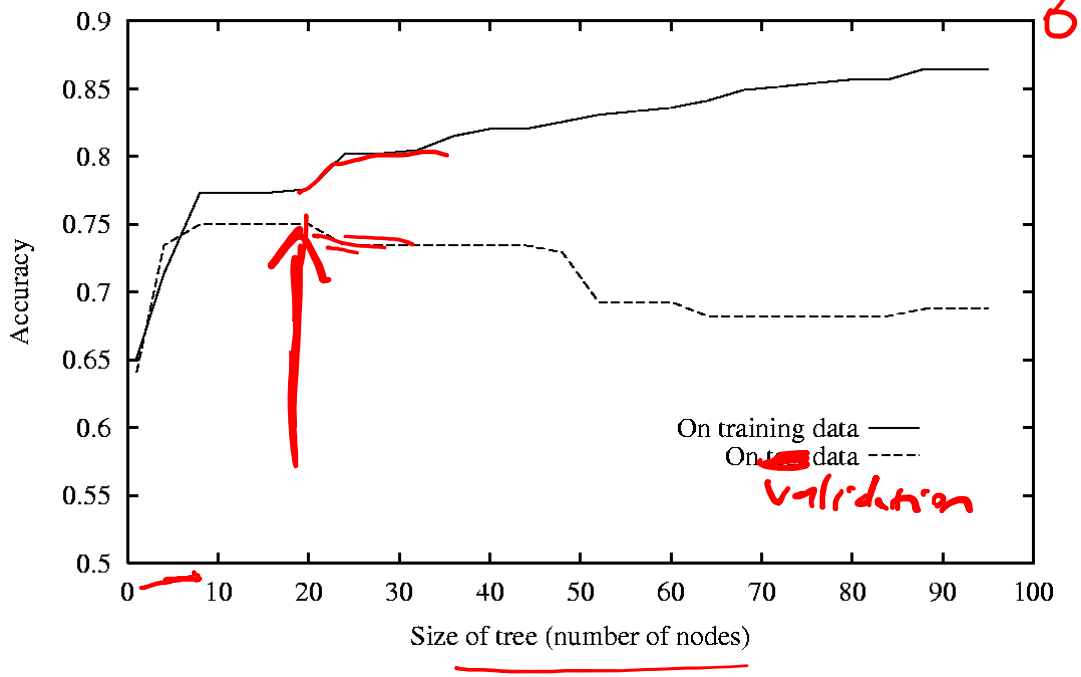
Other			
Long	Sweet	Yellow	Count
F	F	F	0
F	F	T	0
F	T	F	50
F	T	T	50
T	F	F	50
T	F	T	0
T	T	F	50
T	T	T	0

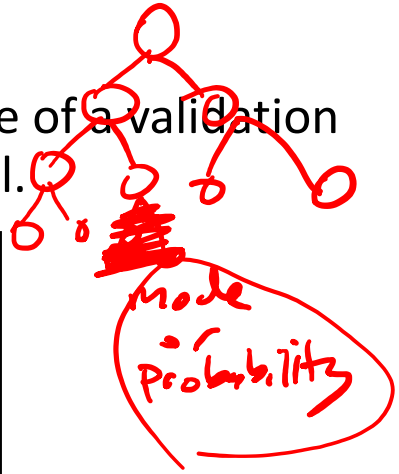


Overfitting



- What's the problem with fitting our data as closely as possible?
  - We may overfit the data!
- Since this is an iterative (or recursive) algorithm, the use of a validation set to decide between different versions is quite natural.





http://jmvidal.cse.sc.edu/talks/decisiontrees/allslides.html

Reduced Error Approaches

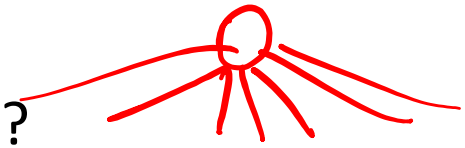
- How can we use the validation set?
- Evaluate it during the growing process.
  - At each *incomplete* branch, just assign the **mode** label of the data going down it.
- Find the level of growth that maximizes the accuracy of the validation set (or conversely minimizes the error).
- Then report test accuracy for that version.
- Reduced Error Growing
  - When you look to split a node based on training data, evaluating after using *validation set*
  - If things get worse, stop
- Reduced Error Pruning
  - Evaluate impact on *validation* set of pruning each possible node (plus those below it)
  - Greedily remove the one that most improve *validation* set accuracy
  - Continue until none help

Other Ideas

- There are other ideas….
- Pruning
  - Try removing each of the splits that happen before leaf nodes and evaluate the validation set on each of them.
  - As long as one of them improves the validation evaluation, remove the one that improves things the most
  - Keep doing this until no removal improves validation evaluation.
- Or maybe we could somehow use statistics to halt the growing process
  - If the information gain provided by a split is above some threshold, split, otherwise don't

Continuous Valued Inputs

- What if we have features that have continuous values?
  - One branch for each value?
    - Bad idea!!!! (impossible?)



$x_i > 0$ $y_i = 1$

$x_i < 0$ $y_i = 0$

- If we know the range of our values and we set how many branches the node should make then
  - Divide range evenly?
  - Somehow do it more intelligently?