CPSC 340:
Machine Learning and Data Mining

Decision Trees

Original version of these slides by Mark Schmidt, with modifications by Mike Gelbart.

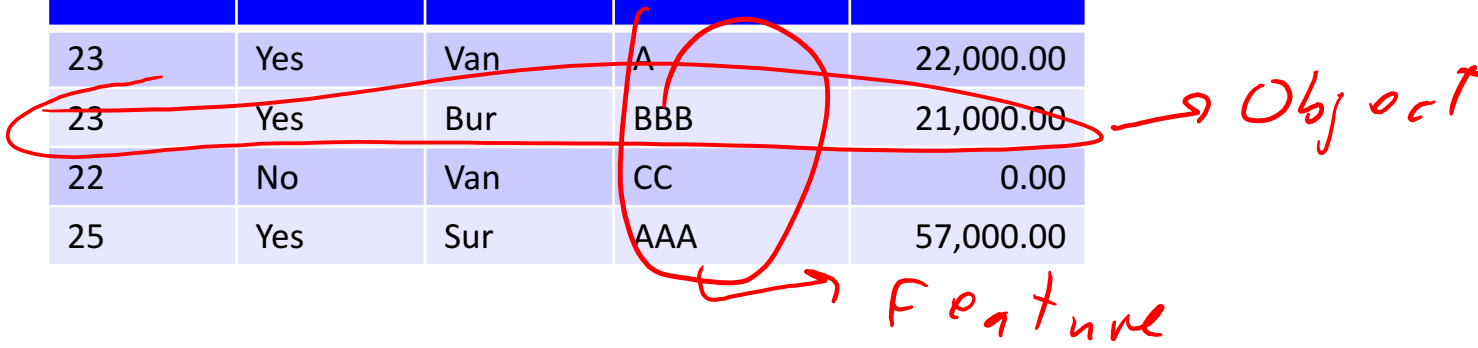
Admin

- Assignment 0 is due Wednesday at 9pm (in 2 days)
- Assignment 1 should be released Wednesday, due a week later
  - If you want to work with a partner, you both must request it BEFORE a1 release
  - Instructions in the Homework Submission Instructions document
- Important webpages:
  - https://www.cs.ubc.ca/getacct/
  - https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/home
  - https://piazza.com/class/j9uk5ecmb7e4ks
- Tutorials and office hours start this week.
  - See course homepage for tutorial topics and office hours schedule.
- Auditing
  - No room for official auditors.
  - Unofficial auditors, please do not take seats if others are standing.

Last Time: Data Representation and Exploration

- We discussed object-feature representation:
  - Examples: another name we'll use for objects.

Age	Job?	City	Rating	Income
23	Yes	Van	A	22,000.00
23	Yes	Bur	BBB	21,000.00
22	No	Van	CC	0.00
25	Yes	Sur	AAA	57,000.00



- We discussed summary statistics and visualizing data.

Motivating Example: Food Allergies

- You frequently start getting an upset stomach
- You suspect an adult-onset food allergy.

Motivating Example: Food Allergies

- To solve the mystery, you start a food journal:

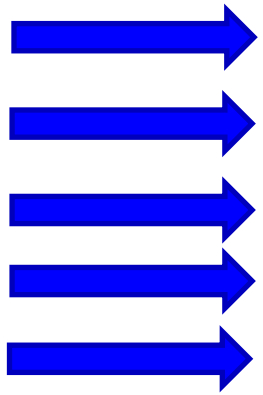
Egg	Milk	Fish	Wheat	Shellfish	Peanuts	…	Sick?
0	0.7	0	0.3	0	0		1
0.3	0.7	0	0.6	0	0.01		1
0	0	0	0.8	0	0		0
0.3	0.7	1.2	0	0.10	0.01		1
0.3	0	1.2	0.3	0.10	0.01		1

- But it's hard to find the pattern:
  - You can't isolate and only eat one food at a time.
  - You may be allergic to more than one food.
  - The quantity matters: a small amount may be ok.
  - You may be allergic to specific interactions.

Supervised Learning

- We can formulate this as supervised learning:

Egg	Milk	Fish	Wheat	Shellfish	Peanuts	…	Sick?
0	0.7	0	0.3	0	0		1
0.3	0.7	0	0.6	0	0.01		1
0	0	0	0.8	0	0		0
0.3	0.7	1.2	0	0.10	0.01		1
0.3	0	1.2	0.3	0.10	0.01		1



- Input for an object (day of the week) is a set of features (quantities of food).
- Output is a desired class label (whether or not we got sick).
- Goal of supervised learning:
  - Use data to find a model that outputs the right label based on the features.
  - Model predicts whether foods will make you sick (even with new combinations).

Supervised Learning

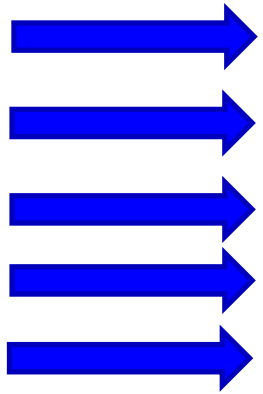
- General supervised learning problem:
  - Take features of objects and corresponding labels as inputs.
  - Find a model that can accurately ***predict the labels of new objects***.

- This is the most successful machine learning technique:
  - Spam filtering, optical character recognition, Microsoft Kinect, speech recognition, classifying tumours, etc.

- We'll first focus on categorical labels, which is called "classification".
  - The model is a called a "classifier".

Naïve Supervised Learning: "Predict Mode"

Egg	Milk	Fish	Wheat	Shellfish	Peanuts	…	Sick?
0	0.7	0	0.3	0	0		1
0.3	0.7	0	0.6	0	0.01		1
0	0	0	0.8	0	0		0
0.3	0.7	1.2	0	0.10	0.01		1
0.3	0	1.2	0.3	0.10	0.01		1



- A very naïve supervised learning method:
  - Count how many times each label occurred in the data (4 vs. 1 above).
  - Always predict the most common label, the "mode" ("sick" above).
- This ignores the features, so is only accurate if we only have 1 label.
- There is no unique "right" way to use the features.
  - Today we'll consider a classic way known as decision tree learning.

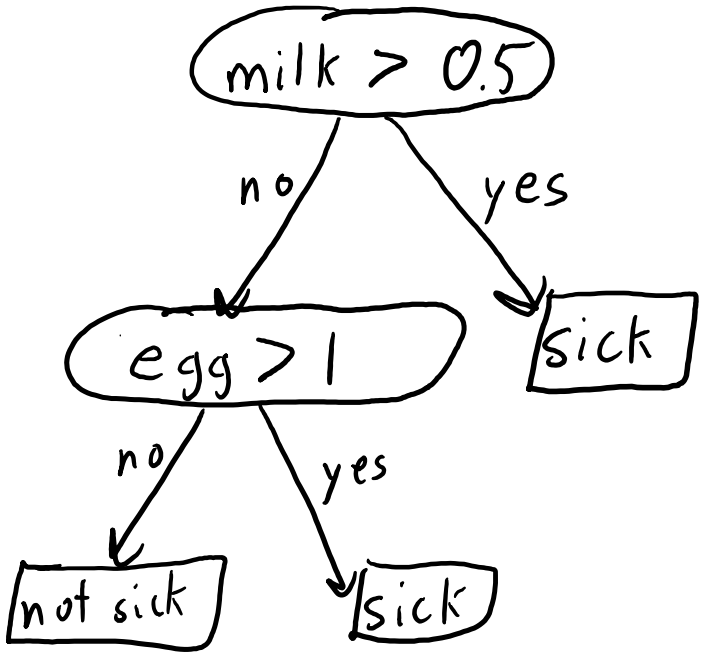
Decision Trees

- Decision trees are simple programs consisting of:
  - A nested sequence of "if-else" decisions based on the features (splitting rules).
  - A class label as a return value at the end of each sequence.

- Example decision tree:

```
if (milk > 0.5)
        {
                return 'sick'
        }
        else
        {
                if (egg > 1)
                        return 'sick'
                else
                        return 'not sick'
        }
```

Can draw sequences of decisions as a tree:



Supervised Learning as Writing A Program

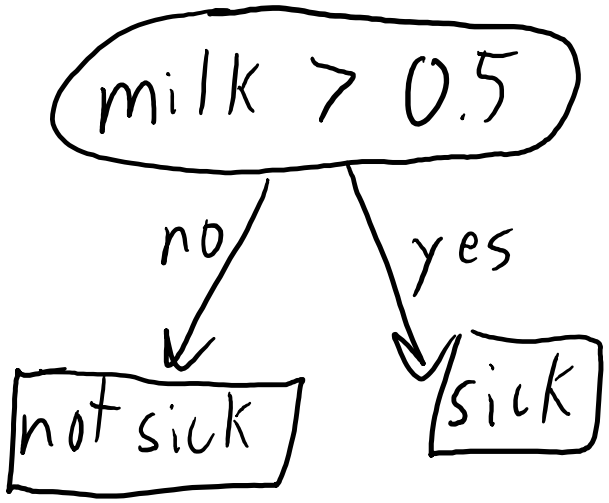
- There are many possible decision trees.
  - We're going to search for one that is good at our supervised learning problem.

- So our input is data and the output will be a program.
  - This is called "training" the supervised learning model.
  - Different than usual input/output specification for writing a program.

- Supervised learning is useful when you have lots of labeled data BUT:
  1. problem is too complicated to write a program ourselves, or
  2. human expert can't explain why you assign certain labels, or
  3. we don't have a human expert for the problem.

Learning A Decision Stump

- We'll start with "decision stumps":
  - Simple decision tree with 1 splitting rule based on thresholding 1 feature.



- How do we find the best "rule" (feature, threshold, and leaf labels)?
  1. Define a 'score' for the rule.
  2. Search for the rule with the best score.

Decision Stump: Accuracy Score

- Most intuitive score: classification accuracy.
  - "If we use this rule, how many objects do we label correctly?"
- Computing classification accuracy for (egg > 1):
  - Find most common labels if we use this rule:
    - When (egg > 1), we were "sick" both times.
    - When (egg <= 1), we were "not sick" three out of four times.
  - Compute accuracy:
    - Rule (egg > 1) is correct on 5/6 objects.
- Scores of other rules:
  - (milk > 0.5) obtains lower accuracy of 4/6 .
  - (egg > 0) obtains optimal accuracy of 6/6.
  - () obtains "baseline" accuracy of 3/6, as does (egg > 2).

Egg	Milk	Fish	…	Sick?
1	0.7	0		1
2	0.7	0		1
0	0	0		0
0	0.7	1.2		0
2	0	1.2		1
0	0	0		0

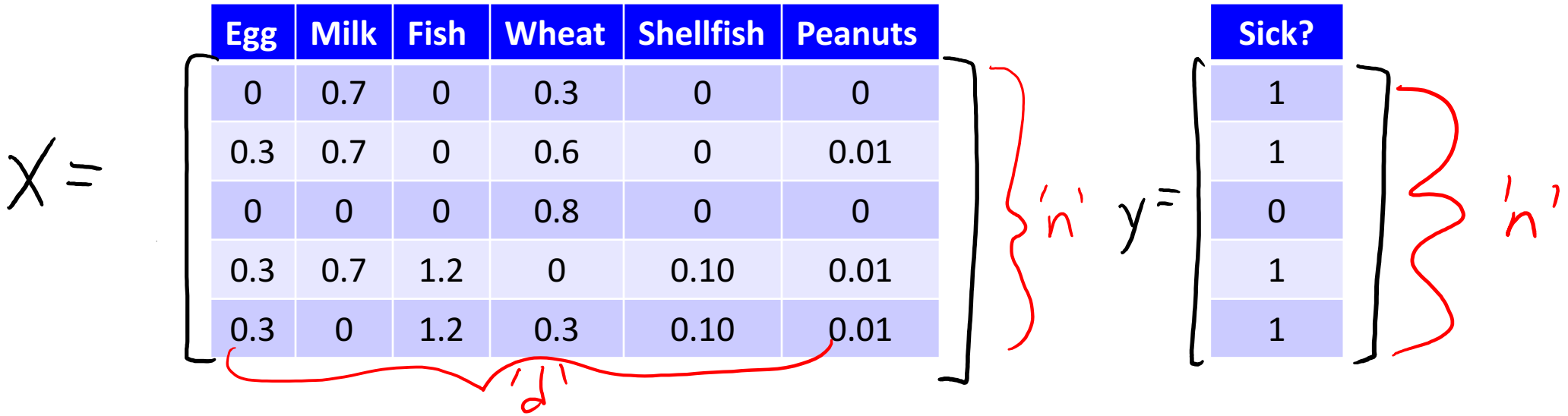
Decision Stump: Rule Search (Attempt 1)

- Accuracy "score" evaluates quality of a rule.
  - Find the best rule by maximizing score.
- Attempt 1 (exhaustive search):

Compute score of (egg > 0)	Compute score of (milk > 0)	…
Compute score of (egg > 0.01)	Compute score of (milk > 0.01)	…
Compute score of (egg > 0.02)	Compute score of (milk > 0.02)	…
Compute score of (egg > 0.03)	Compute score of (milk > 0.03)	…
…	…	…
Compute score of (egg > 99.99)	Compute score of (milk > 0.99)	…

- As you go, keep track of the highest score.
- Return highest-scoring rule (variable, threshold, and leaf values).

Supervised Learning Notation (MEMORIZE THIS)



- Feature matrix 'X' has rows as objects, columns as features.
  - x_{ij} is feature 'j' for object 'i' (quantity of food 'j' on day 'i').
  - x_i is the list of all features for object 'i' (all the quantities on day 'i').
  - x^j is column 'j' of the matrix (the value of feature 'j' across all objects).
- Label vector 'y' contains the labels of the objects.
  - y_i is the label of object 'i' (1 for "sick", 0 for "not sick").

Supervised Learning Notation (MEMORIZE THIS)

$X =$

Egg	Milk	Fish	Wheat	Shellfish	Peanuts
0	0.7	0	0.3	0	0
0.3	0.7	0	0.6	0	0.01
0	0	0	0.8	0	0
0.3	0.7	1.2	0	0.10	0.01
0.3	0	1.2	0.3	0.10	0.01

'n' 'd'

$y =$

Sick?
1
1
0
1
1

'n'

- Training phase:
  - Use 'X' and 'y' to find a 'model' (like a decision stump).
- Prediction phase:
  - Given an object x_i, use the 'model' to predict a label 'yhat_i' ("sick" or "not sick").
- Training error:
  - Fraction of times our prediction 'yhat_i' does not equal the true y_i label.

$\hat{y}_i$

Decision Stump Learning Pseudo-Code

Input: feature matrix X and label vector y

for each feature 'j' (column of 'X')

for each threshold 't'

set 'y_yes' to most common label of objects 'i' satisfying rule ($x_{ij} > t$)

set 'y_no' to most common label of objects not satisfying rule.

set '$\hat{y}$' to be our predictions for each object 'i' based on the rule.

compute error 'E', number of objects where $\hat{y}_i \neq y_i$ ($\hat{y}_i = y\_yes$ if satisfied, $\hat{y}_i = y\_no$ if not satisfied)

store the rule (j, t, y_yes, y_no) if it has the lowest error so far.

Output: an optimal decision stump rule (the "model")

Cost of Decision Stumps (Attempt 1)

- How much does this cost?
- Assume we have:
  - 'n' objects (days that we measured).
  - 'd' features (foods that we measured).
  - 'k' thresholds (>0, >0.01, >0.02,…)
- Computing the score of one rule costs O(n):
  - We need to go through all 'n' examples.
  - See notes on webpage for review of "O(n)" notation.
- To compute scores for d*k rules, total cost is O(ndk).
  - But 'k' might be huge
- Can we do better?

Speeding up Rule Search

- We can ignore rules outside feature ranges:
  - E.g., we never have (egg > 50) in our data.
  - These rules can never improve accuracy.
  - Restrict thresholds to range of features.

- Most of the thresholds give the same score.
  - If we never have (0.5 < egg < 1) in the data,
    - then (egg < 0.6) and (egg < 0.9) have the same score.
  - Restrict thresholds to values in data.

Decision Stump: Rule Search (Attempt 2)

- Attempt 2 (search only over features in data):

Compute score of (eggs > 0)	Compute score of (milk > 0.5)	…
Compute score of (eggs > 1)	Compute score of (milk > 0.7)	…
Compute score of (eggs > 2)	Compute score of (milk > 1)	…
Compute score of (eggs > 3)	Compute score of (milk > 1.25)	…
Compute score of (eggs > 4)		…

- Now at most ‘n’ thresholds for each feature.
- We only consider O(nd) rules instead of O(dk) rules:
  - Total cost changes from O(ndk) to $O(n^2d)$.

Decision Stump: Rule Search (Attempt 3)

- Do we have to compute the score from scratch?
  - Rule (egg > 1) and (egg > 2) have same decisions, except when (egg == 2).
- We can actually compute the best rule involving 'egg' in O(n log n):
  - Sort the examples based on 'egg', and use these positions to re-arrange 'y'.
  - Go through the sorted values in order, updating the counts of #sick and #not-sick that both satisfy and don't satisfy the rules.
  - With these counts, it's easy to compute the classification accuracy (see bonus slide).
- Sorting costs O(n log n) per feature.
- Total cost of updating counts is O(n) per feature.
- Total cost is reduced from $O(n^2d)$ to $O(nd \log n)$.
- This is a good runtime:
  - O(nd) is the size of data, so O(nd log n) is same as looking at data, up to a log factor.
  - We can apply this algorithm to huge datasets.

(pause)

Decision *Tree* Learning

- Decision stumps have only 1 rule based on only 1 feature.
  - Very limited class of models: usually not very accurate for most tasks.

- Decision trees allow sequences of splits based on multiple features.
  - Very general class of models: can get very high accuracy.
  - However, it's computationally infeasible to find the best decision tree.

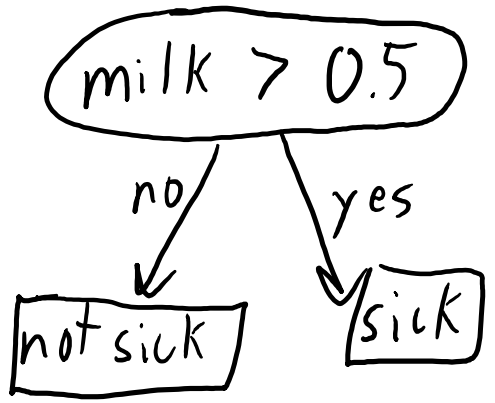
- Most common decision tree learning algorithm in practice:
  - Greedy recursive splitting.

Example of Greedy Recursive Splitting

- Start with the full dataset:

Egg	Milk	…	Sick?
0	0.7		1
1	0.7		1
0	0		0
1	0.6		1
1	0		0
2	0.6		1
0	1		1
2	0		1
0	0.3		0
1	0.6		0
2	0		1

Find the decision stump with the best score:



Split into two smaller datasets based on stump:

Egg	Milk	…	Sick?
0	0		0
1	0		0
2	0		1
0	0.3		0
2	0		1

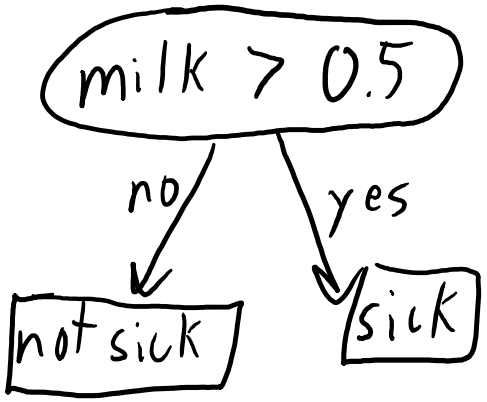
milk ≤ 0.5

Egg	Milk	…	Sick?
0	0.7		1
1	0.7		1
1	0.6		1
2	0.6		1
0	1		1
1	0.6		0

> 0.5

Greedy Recursive Splitting

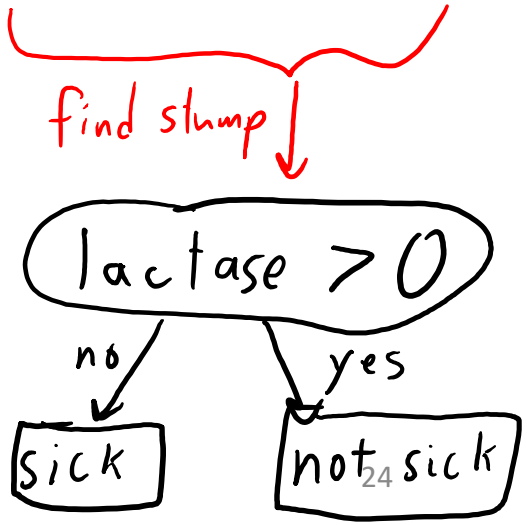
We now have a decision stump and two datasets:



Egg	Milk	…	Sick?
0	0		0
1	0		0
2	0		1
0	0.3		0
2	0		1

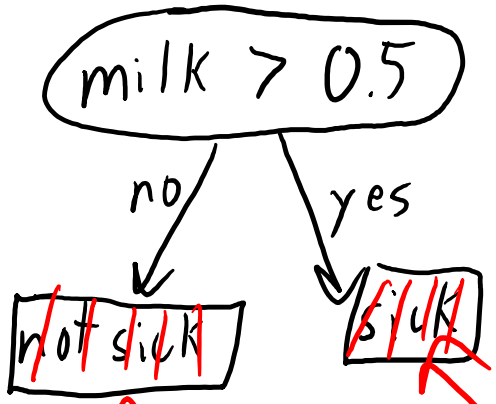
Egg	Milk	…	Sick?
0	0.7		1
1	0.7		1
1	0.6		1
2	0.6		1
0	1		1
1	0.6		0

Fit a decision stump to each leaf's data.



Greedy Recursive Splitting

We now have a decision stump and two datasets:



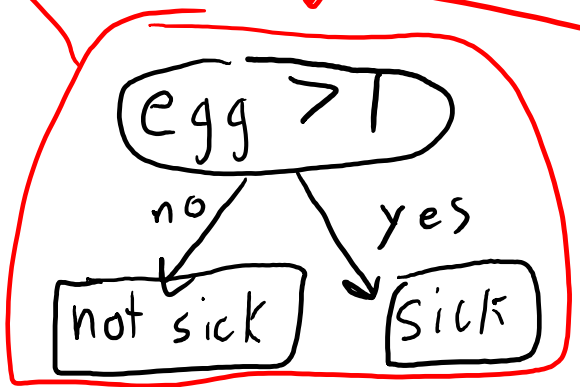
Egg	Milk	…	Sick?
0	0		0
1	0		0
2	0		1
0	0.3		0
2	0		1

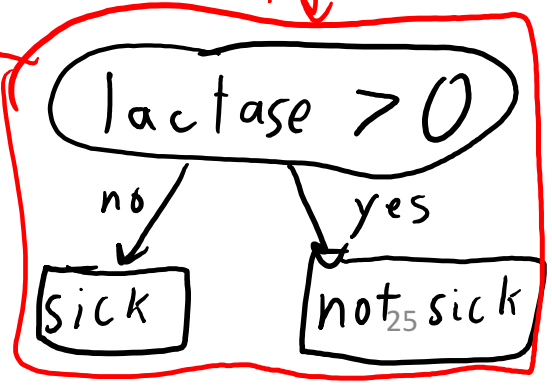
Egg	Milk	…	Sick?
0	0.7		1
1	0.7		1
1	0.6		1
2	0.6		1
0	1		1
1	0.6		0

find stump

find stump

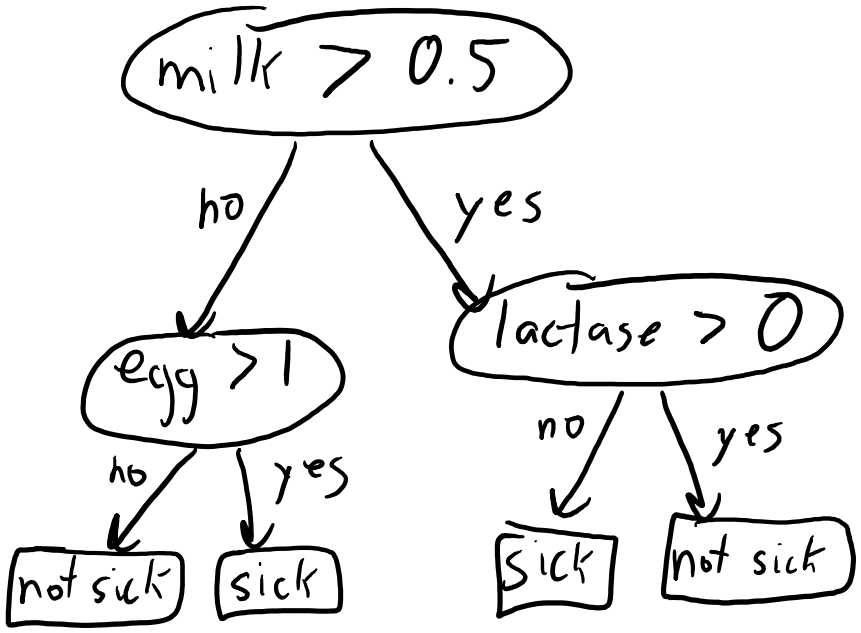
Fit a decision stump to each leaf's data.
Then add these stumps to the tree.





Greedy Recursive Splitting

This gives a "depth 2" decision tree:



It splits the two datasets into four datasets:

milk ≤ 0.5 data

Egg	Milk	...	Sick?
0	0		0
1	0		0
2	0		1
0	0.3		0
2	0		1

milk > 0.5 data

Egg	Milk	...	Sick?
0	0.7		1
1	0.7		1
1	0.6		1
2	0.6		1
0	1		1
1	0.6		0

milk ≤ 0.5, egg ≤ 1

Egg	Milk	...	Sick?
0	0		0
1	0		0
0	0.3		0

Much more accurate!

milk ≤ 0.5, egg > 1

Egg	Milk	...	Sick?
2	0		1
2	0		1

milk > 0.5, lactase ≤ 0

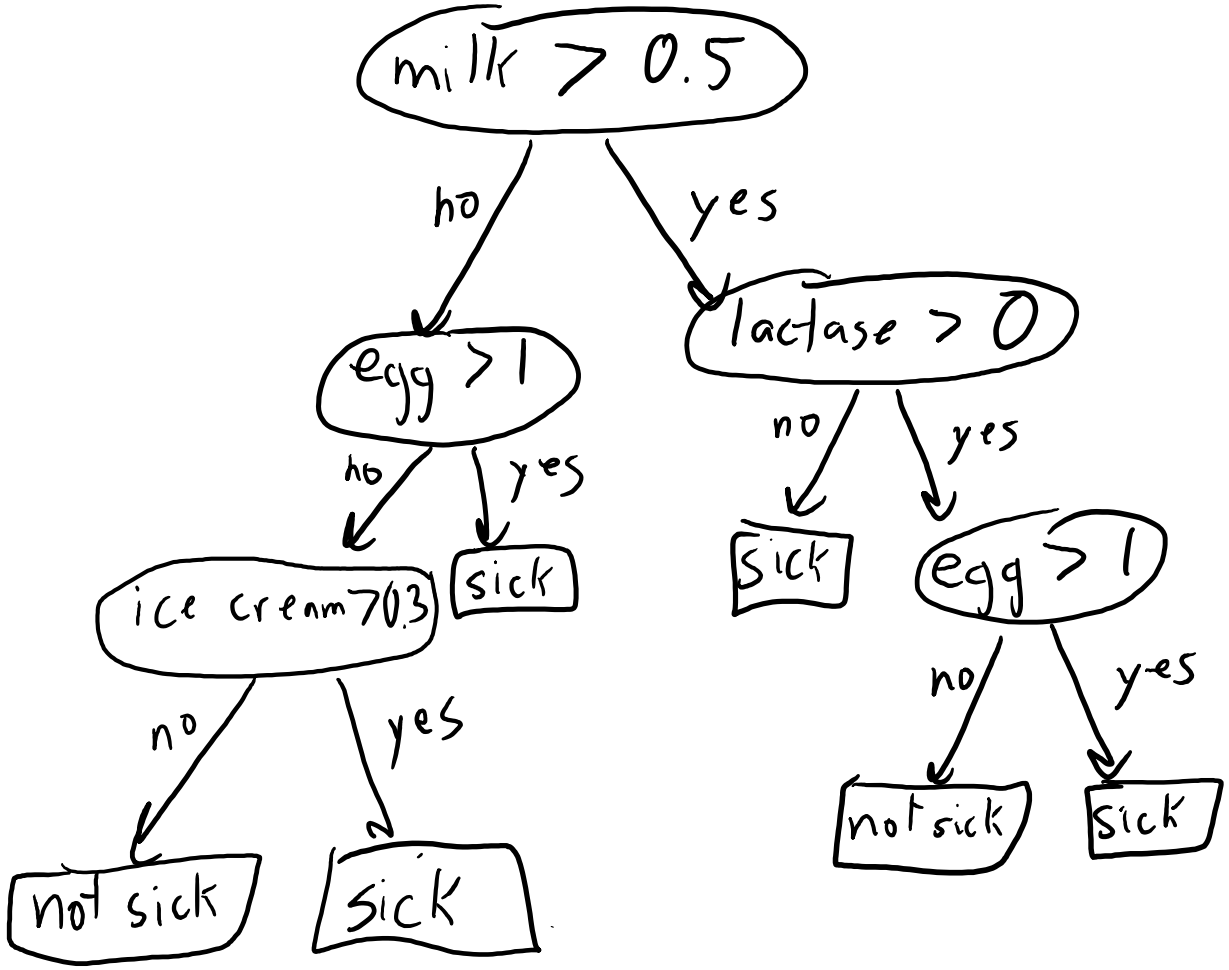
Egg	Milk	...	Sick?
0	0.7		1
1	0.7		1
1	0.6		1
2	0.6		1

milk > 0.5, lactase > 0

Egg	Milk	...	Sick?
1	0.6		0

Greedy Recursive Splitting

We could try to split the four leaves to make a "depth 3" decision tree:



We might continue splitting until:

- The leaves each have only one label.
- We reach a user-defined maximum depth.

Discussion of Decision Tree Learning

- Advantages:
  - Interpretable.
  - Fast to learn.
  - Very fast to classify
- Disadvantages:
  - Hard to find optimal set of rules.
  - Greedy splitting often not accurate, requires very deep trees.
- Issues:
  - Can you revisit a feature?
    - Yes, knowing other information could make feature relevant again.
  - More complicated rules?
    - Yes, but searching for the best rule gets much more expensive.
  - Is accuracy the best score?
    - No, there may be no split that increase accuracy. Alternative: information gain (bonus slides).
  - What depth?

Summary

- Supervised learning:
  - Using data to write a program based on input/output examples.
- Decision trees: predicting a label using a sequence of simple rules.
- Decision stumps: simple decision tree that is very fast to fit.
- Greedy recursive splitting: uses a sequence of stumps to fit a tree.
  - Very fast and interpretable, but not always the most accurate.

Other Considerations for Food Allergy Example

- What types of preprocessing might we do?
  - Data cleaning: check for and fix missing/unreasonable values.
  - Summary statistics:
    - Can help identify “unclean” data.
    - Correlation might reveal an obvious dependence (“sick” ⇔ “peanuts”).
  - Data transformations:
    - Convert everything to same scale? (e.g., grams)
    - Add foods from day before? (maybe “sick” depends on multiple days)
    - Add date? (maybe what makes you “sick” changes over time).
  - Data visualization: look at a scatterplot of each feature and the label.
    - Maybe the visualization will show something weird in the features.
    - Maybe the pattern is really obvious!
- What you do might depend on how much data you have:
  - Very little data:
    - Represent food by common allergic ingredients (lactose, gluten, etc.)?
  - Lots of data:
    - Use more fine-grained features (bread from bakery vs. hamburger bun)?

How do we fit stumps in O(dn log n)?

- Let's say we're trying to find the best rule involving milk:

Egg	Milk	…
0	0.7	
1	0.7	
0	0	
1	0.6	
1	0	
2	0.6	
0	1	
2	0	
0	0.3	
1	0.6	
2	0	

Sick?
1
1
0
1
0
1
1
1
0
0
1

First grab the milk column and sort it (using the sort positions to re-arrange the sick column). This step costs O(n log n) due to sorting.

Milk
0
0
0
0
0.3
0.6
0.6
0.6
0.7
0.7
1

Sick?
0
0
0
0
0
1
1
0
1
1
1

Now, we'll go through the milk values in order, keeping track of #sick and #not sick that are above/below the current value. E.g., #sick above 0.3 is 5.

With these counts, accuracy score is (sum of most common label above and below)/n.

How do we fit stumps in O(dn log n)?

Milk	Sick?
0	0
0	0
0	0
0	0
0.3	0
0.6	1
0.6	1
0.6	0
0.7	1
0.7	1
1	1

Start with the baseline rule () which is always "satisfied":
If satisfied, #sick=5 and #not-sick=**6**.
If not satisfied, #sick=0 and #not-sick=0.
This gives accuracy of (6+0)/n = 6/11.

Next try the rule (milk > 0), and update the counts based on these 4 rows:
If satisfied, #sick=**5** and #not-sick=2.
If not satisfied, #sick=0 and #not-sick=**4**.
This gives accuracy of (5+4)/n = 9/11, which is better.

Next try the rule (milk > 0.3), and update the counts based on this 1 row:
If satisfied, #sick=**5** and #not-sick=1.
If not satisfied, #sick=0 and #not-sick=**5**.
This gives accuracy of (5+5)/n = 10/11, which is better.
(and keep going until you get to the end…)

How do we fit stumps in O(dn log n)?

Milk
0
0
0
0
0.3
0.6
0.6
0.6
0.7
0.7
1

Sick?
0
0
0
0
0
1
1
0
1
1
1

Notice that for each row, updating the counts only costs O(1).
Since there are O(n) rows, total cost of updating counts is O(n).

Instead of 2 labels (sick vs. not-sick), consider the case of 'k' labels:

- Updating the counts still costs O(n), since each row has one label.
- But computing the 'max' across the labels costs O(k), so cost is O(kn).

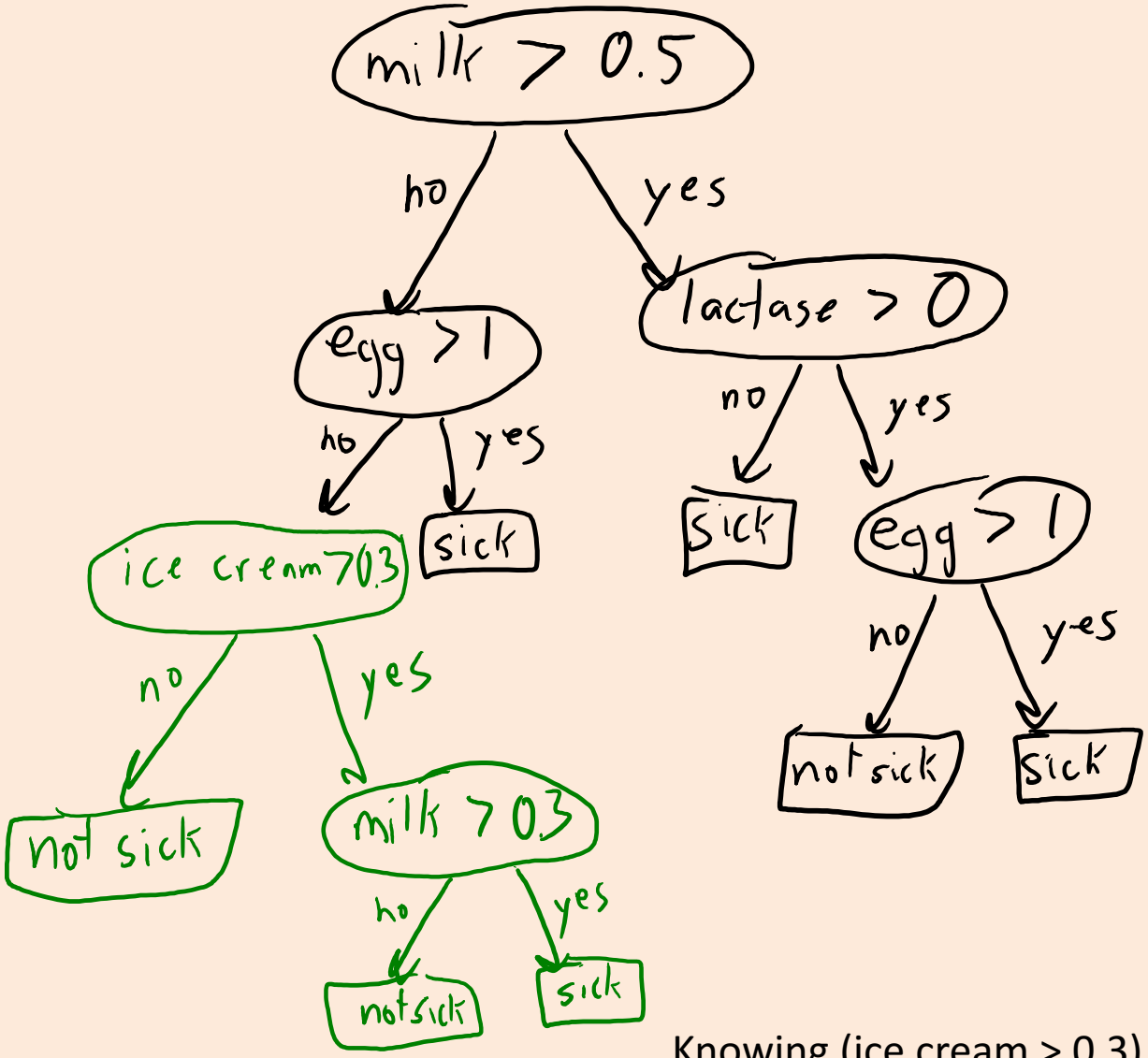
With 'k' labels, you can decrease cost using a "max-heap" data structure:

- Cost of getting max is O(1), cost of updating heap for a row is O(log k).
- But k <= n (each row has only one label).
- So cost is in O(log n) for one row.

Since the above shows we can find best rule in one column in O(n log n), total cost to find best rule across all 'd' columns is O(nd log n).

Can decision trees re-visit a feature?

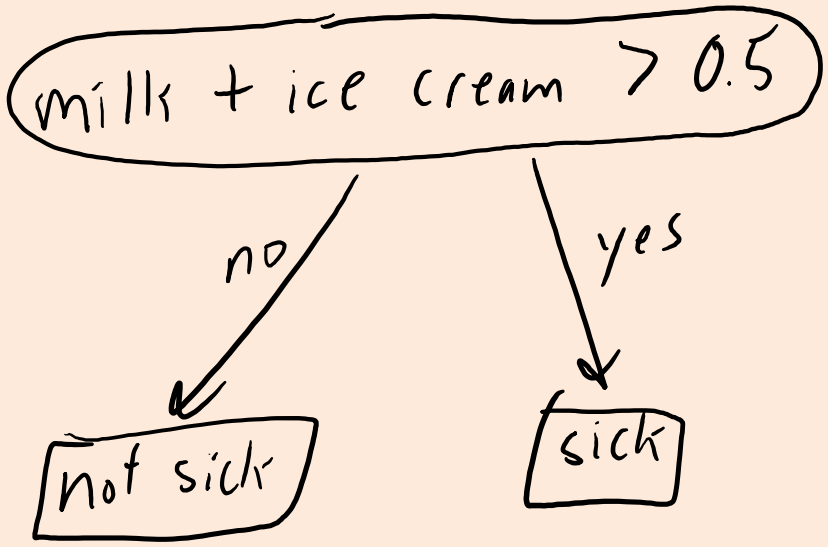
- Yes.



Knowing (ice cream > 0.3) makes small milk quantities relevant.

Can decision trees have more complicated rules?

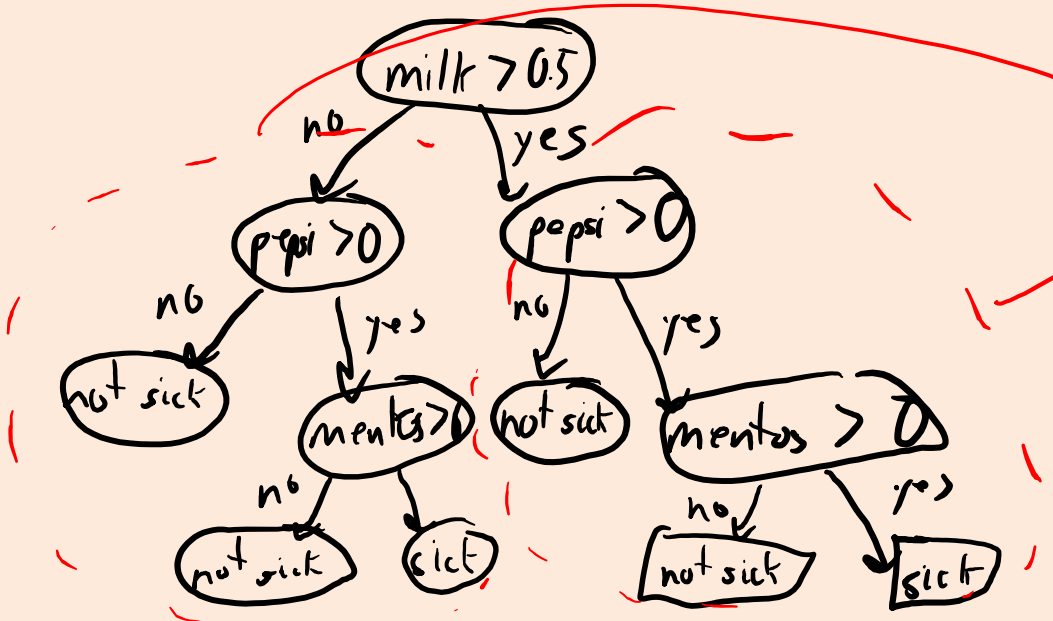
- Yes:



- But searching for best rule can get expensive.

Does being greedy actually hurt?

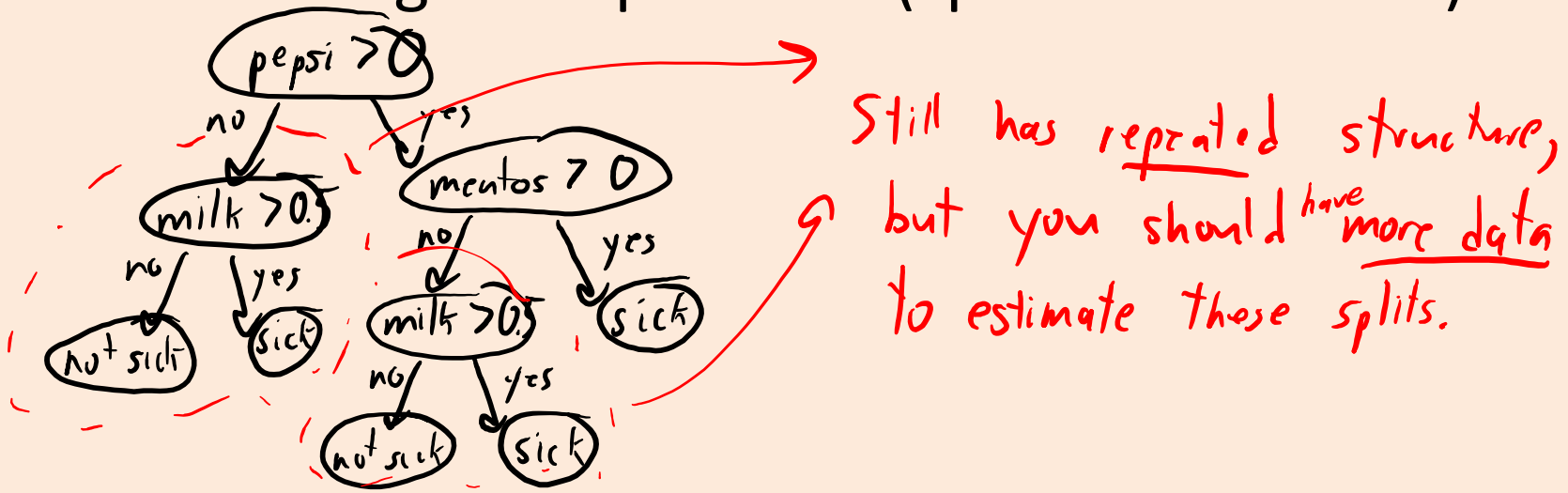
- Can't you just go deeper to correct greedy decisions?
  - Yes, but you need to "re-discover" rules with less data.
- Consider that you are allergic to milk (and drink this often), and also get sick when you (rarely) combine diet coke with mentos.
- Greedy method should first split on milk (helps accuracy the most):



Get same sub-tree, you need learn it twice and with less data

Does being greedy actually hurt?

- Can't you just go deeper to correct greedy decisions?
  - Yes, but you need to "re-discover" rules with less data.
- Consider that you are allergic to milk (and drink this often), and also get sick when you (rarely) combine diet coke with mentos.
- Greedy method should first split on milk (helps accuracy the most).
- Non-greedy method could get simpler tree (split on milk later):



Which score function should a decision tree used?

- Shouldn't we just use accuracy score?
  - For leafs: yes, just maximize accuracy.
  - For internal nodes: maybe not.
    - There may be no simple rule like (egg > 0.5) that improves accuracy.

- Most common score in practice: information gain.
  - Choose split that decreases entropy ("randomness") of labels the most.
  - Motivation: try to make split data "less random" or "more predictable".
    - Might then be easier to find high-accuracy on the "less random" split data.

Decision Trees with Probabilistic Predictions

- Often, we'll have multiple 'y' values at each leaf node.
- In these cases, we might return probabilities instead of a label.

- E.g., if in the leaf node we 5 have "sick" objects and 1 "not sick":
  - Return $p(y = \text{"sick"} \mid x_i) = 5/6$ and $p(y = \text{"not sick"} \mid x_i) = 1/6$.

- In general, a natural estimate of the probabilities at the leaf nodes:
  - Let 'n_k' be the number of objects that arrive to leaf node 'k'.
  - Let 'n_{kc}' be the number of times (y == c) in the objects at leaf node 'k'.
  - Maximum likelihood estimate for this leaf is $p(y = c \mid x_i) = n_{kc}/n_k$.

Alternative Stopping Rules

- There are more complicated rules for deciding when *not* to split.

- Rules based on minimum sample size.
  - Don't split any nodes where the number of objects is less than some 'm'.
  - Don't split any nodes that create children with less than 'm' objects.
    - These types of rules try to make sure that you have enough data to justify decisions.

- Alternately, you can use a validation set (see next lecture):
  - Don't split the node if it decreases an approximation of test accuracy.