CPSC 340: Machine Learning and Data Mining

Decision Trees

Bonus slides

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)?

Julia Decision Stump Code (not O(n log n) yet)

Input: feature matrix X and label vector y

```
(n, d) = size(X)
minError = sum(y != mode(y))     compute error if you don't split (user-defined function "mode")
minRule = []
for j = 1:d                      for each feature 'j'
    for i = 1:n                      for each example 'i'
        t = X[i,j]                       set threshold to feature 'j' in example 'i'!
        y_above = mode(y[X[:,j].> t])    find mode of label vector when feature 'j' is above threshold
        y_below = mode(y[X[:,j].<= t])   find mode of label vector when feature 'j' is below threshold.
        yhat = fill(y_above, n)          classify all examples based on threshold
        yhat[X[:,j].<= t] = y_below
        error = sum(yhat != y)           count the number of errors.
        if error < minError              store this rule if it has the lowest error so far.
            minError = error
            minRule = [j  t]
```

Going from $O(n^2d)$ to $O(nd \log n)$ for Numerical Features

- Do we have to compute score from scratch?
  - As an example, assume we eat integer number of eggs:
    - So the rules (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, same as runtime up to a log factor.
  - We can apply this algorithm to huge datasets.

How do we fit stumps in O(nd 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.

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.

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

How do we fit stumps in O(nd 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(nd 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

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).

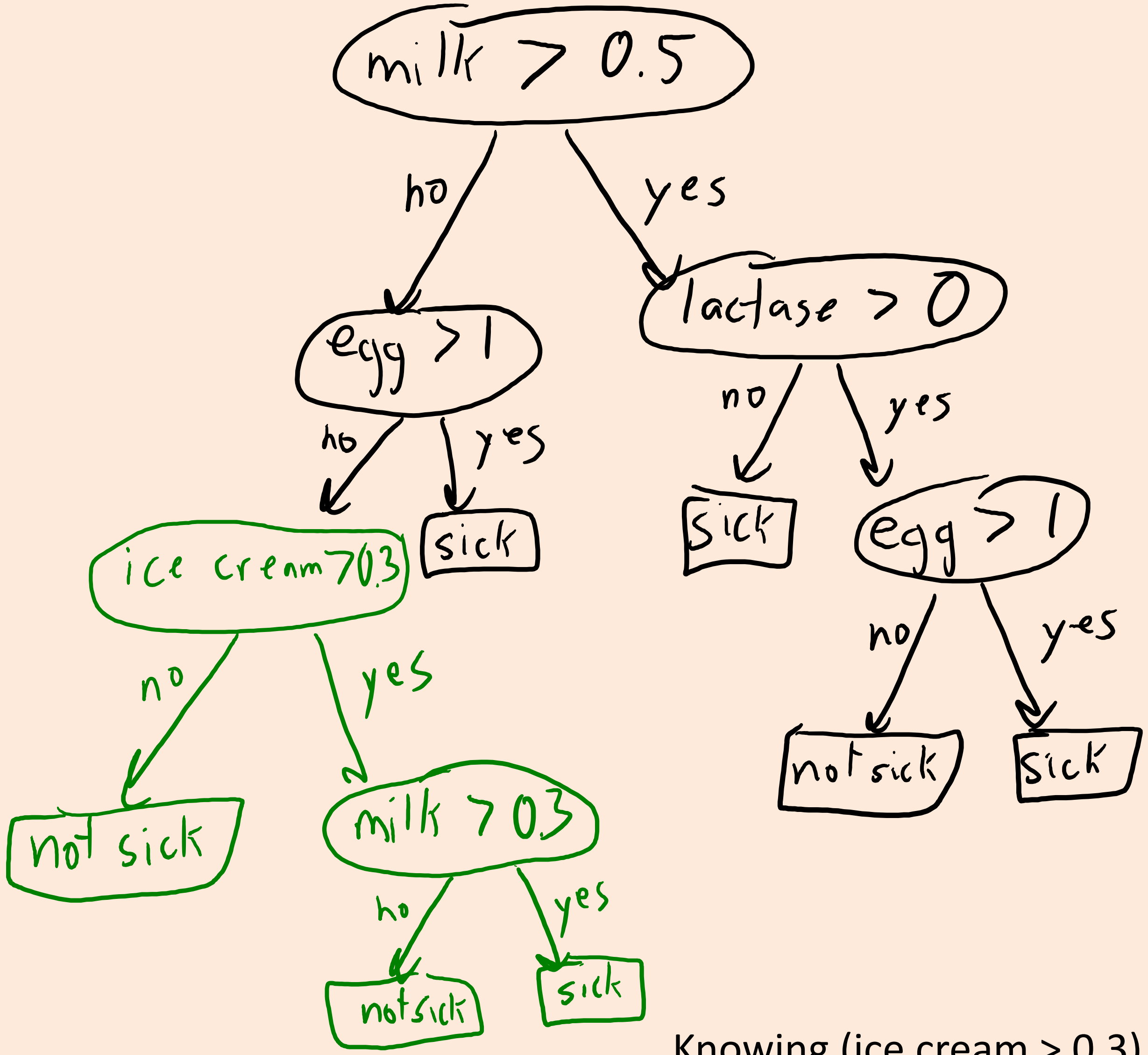
With 'k' labels, you can decrease cost using a "max-heap" data structure:

- 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?

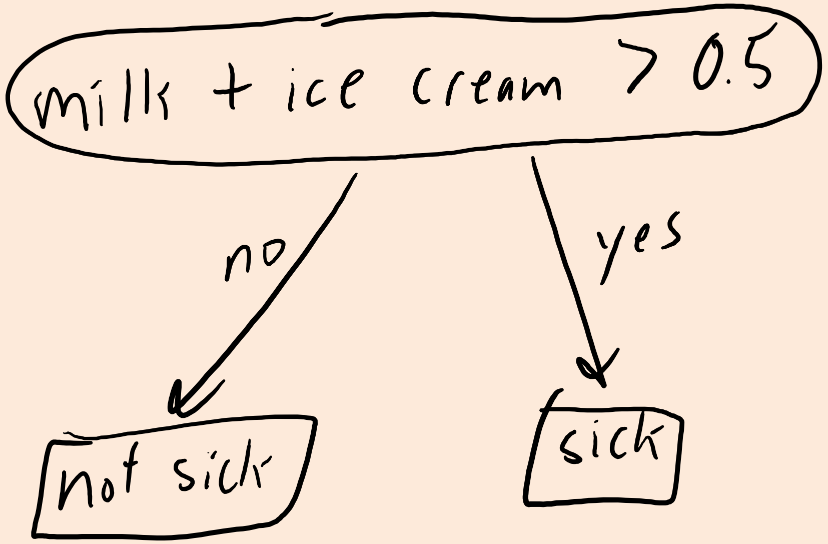
- Yes.



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

Can decision trees have more complicated rules?

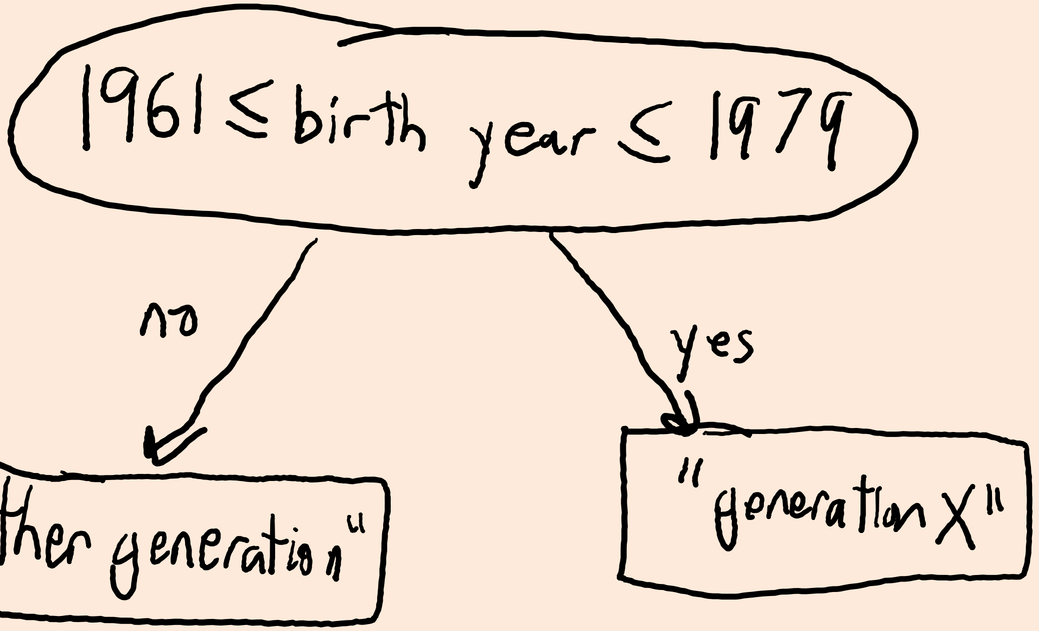
- Yes!
- Rules that depend on more than one feature:



- But now searching for the best rule can get expensive.

Can decision trees have more complicated rules?

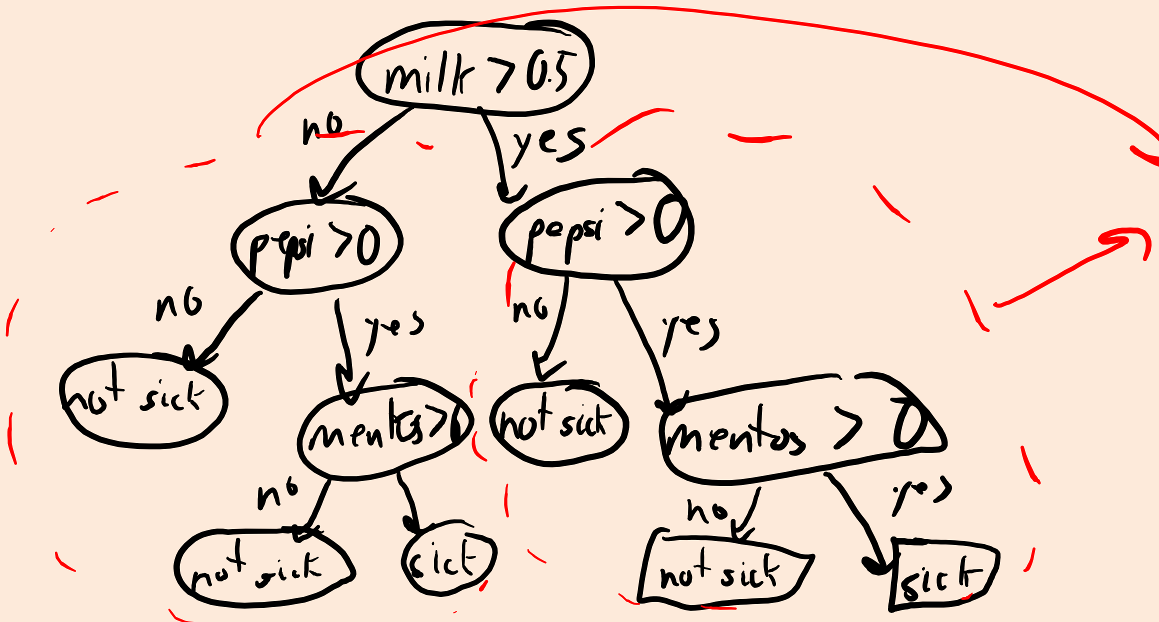
- Yes!
- Rules that depend on more than one threshold:



- "Very Simple Classification Rules Perform Well on Most Commonly Used Datasets"
  - Consider decision stumps based on multiple splits of 1 attribute.
  - Showed that this gives comparable performance to more-fancy methods on many datasets.

Does being greedy actually hurt?

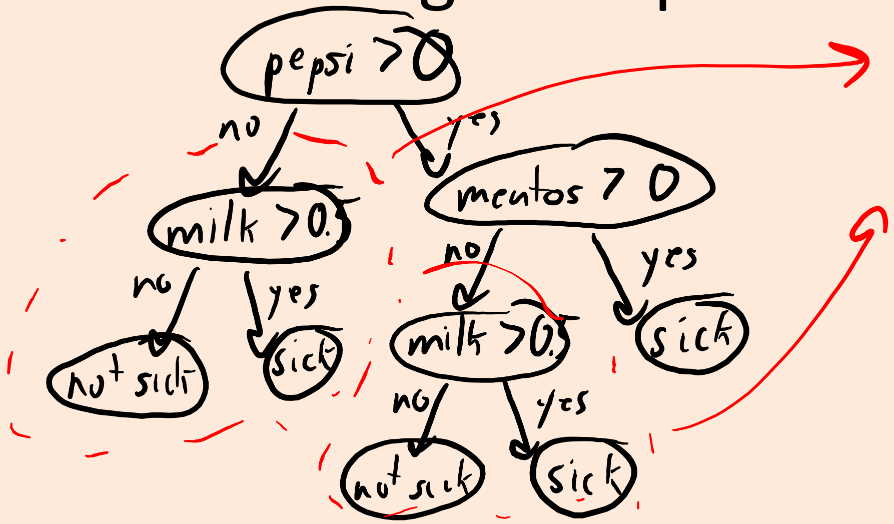
- Can't you just go deeper to correct greedy decisions?
  - Yes, but you need to "re-discover" rules with less data.
- Consider that you are allergic to milk (and drink this often), and also get sick when you (rarely) combine diet coke with mentos.
- Greedy method should first split on milk (helps accuracy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



Still has <u>repeated</u> structure, but you should have <u>more data</u> to estimate these splits.

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" examples 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 examples that arrive to leaf node 'k'.
  - Let 'n_{kc}' be the number of times (y == c) in the examples 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 examples is less than some ‘m’.
  - Don’t split any nodes that create children with less than ‘m’ examples.
    - 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.