

# CPSC 340: Machine Learning and Data Mining

Decision Trees

Fall 2019

# Admin

- **Assignment 1** is due Friday: start early.
  - Getting you signed up for Gradescope is still in progress, it will get worked out.
- Waiting list people: you should be registered soon-ish.
  - Start on the assignment now, everybody currently on the waiting list will get in.
- Course webpage: <https://www.cs.ubc.ca/~schmidtm/Courses/340-F19/>
  - Sign up for Piazza.
- Tutorials and office hours start this week (see webpage for time).
- Auditors:
  1. Sign up for the class.
  2. Show me you are enrolled.
  3. After everyone is off the waiting list,  
I will sign your form switching you to audit status, and tell you the requirements.

# Last Time: Data Representation and Exploration

- We discussed **example-feature representation**:

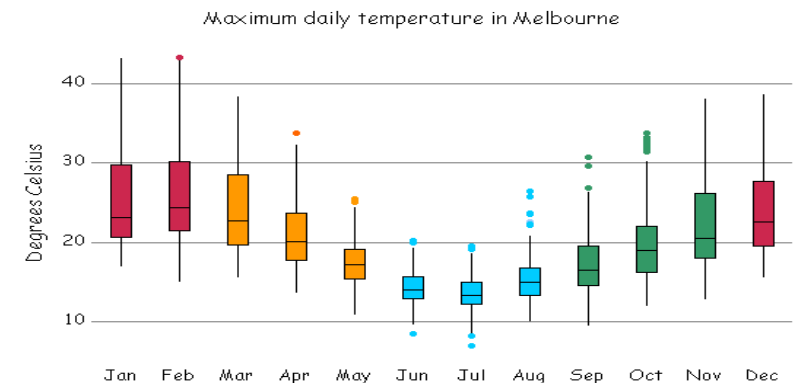
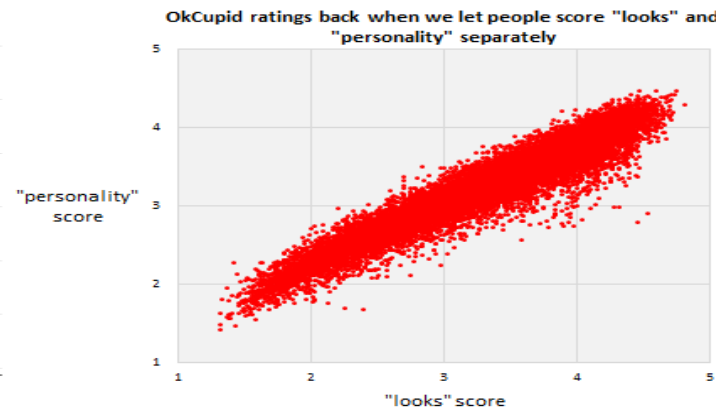
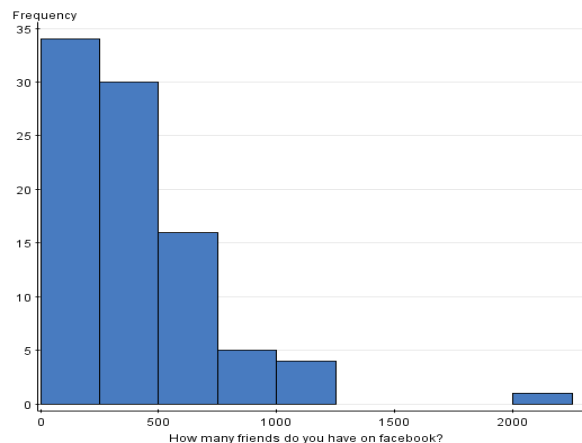
- **Samples**: another name we'll use for examples.

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

"feature"

"example"

- We discussed **summary statistics** and **visualizing data**.



# Last Time: Supervised Learning

- We discussed 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 example (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.
    - Above, model predicts whether foods will make you sick (even with new combinations).
  - This framework can be applied any problem where we have input/output examples.

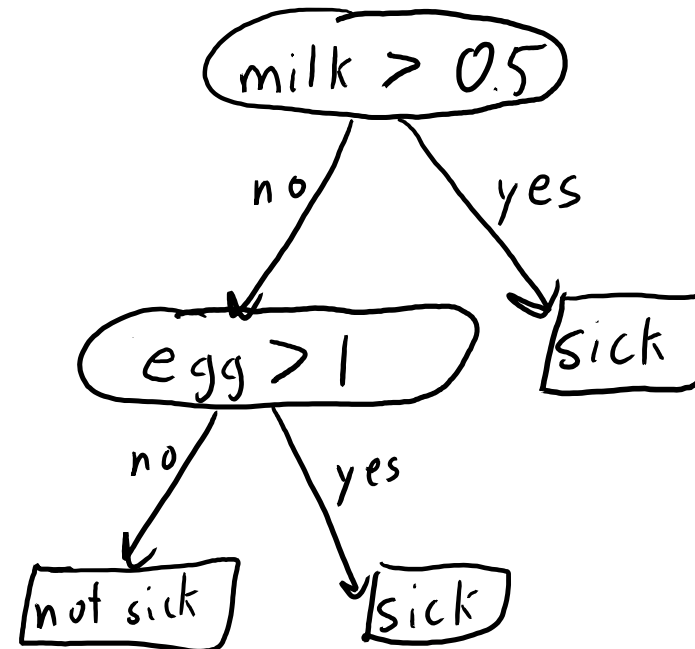
# Decision Trees

- **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**:

Can draw sequences of decisions as a tree:

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



# Supervised Learning as Writing A Program

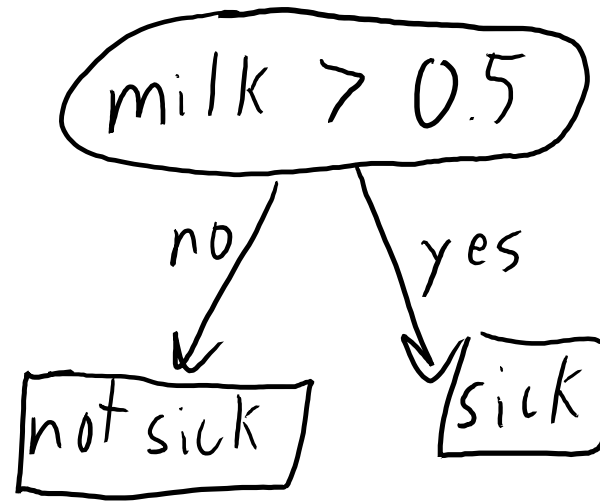
- There are many possible decision trees.
  - We're going to search for one that is good at our supervised learning problem.
- So our input is data and the output will be a program.
  - This is called “training” the supervised learning model.
  - Different than usual input/output specification for writing a program.
- Supervised learning is useful when you have lots of labeled data BUT:
  1. Problem is too complicated to write a program ourselves.
  2. Human expert can't explain why you assign certain labels.

OR

  2. We don't have a human expert for the problem.

# Learning A Decision Stump: “Search and Score”

- 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.

# Learning A Decision Stump: Accuracy Score

- Most intuitive score: **classification accuracy**.
  - “If we use this rule, how many examples 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” 2 times out of 2.
  - When (egg ≤ 1), we were “not sick” 3 times out of 4.

- Compute **accuracy**:

- The accuracy (“score”) of the rule (egg > 1) is **5 times out of 6**.

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

- This “**score**” evaluates quality of a rule.
  - We “learn” a decision stump by **finding the rule with the best score**.



# Learning A Decision Stump: By Hand

- Let's **search** for the decision stump maximizing classification **score**:

Milk	Fish	Egg	Sick?
0.7	0	1	1
0.7	0	2	1
0	1.2	0	0
0.7	1.2	0	0
0	1.3	2	1
0	0	0	0

First we check “**baseline rule**” of predicting mode (no split): this gets 3/6 accuracy.

If (milk > 0) predict “sick” (2/3) else predict “not sick” (2/3): 4/6 accuracy

If (fish > 0) predict “not sick” (2/3) else predict “sick” (2/3): 4/6 accuracy

If (fish > 1.2) predict “sick” (1/1) else predict “not sick” (3/5): 5/6 accuracy

If (egg > 0) predict “sick” (3/3) else predict “not sick” (3/3): 6/6 accuracy

If (egg > 1) predict “sick” (2/2) else predict “not sick” (3/4): 5/6 accuracy

- Highest-scoring rule**: (egg > 0) with leaves “sick” and “not sick”.
- Notice we **only need to test feature thresholds that happen** in the data:
  - There is no point in testing the rule (egg > 3), it gets the “baseline” score.
  - There is no point in testing the rule (egg > 0.5), it gets the (egg > 0) score.
  - Also note that we don't need to test “<”, since it would give equivalent rules.

# 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

$y =$

Sick?
1
1
0
1
1

Handwritten red annotations: A bracket labeled 'n' spans the rows of matrix X. A bracket labeled 'n' spans the rows of vector y. A bracket labeled 'd' spans the columns of matrix X.

- Feature matrix 'X' has rows as examples, columns as features.
  - $x_{ij}$  is feature 'j' for example 'i' (quantity of food 'j' on day 'i').
  - $x_i$  is the list of all features for example 'i' (all the quantities on day 'i').
  - $x^j$  is column 'j' of the matrix (the value of feature 'j' across all examples).
- Label vector 'y' contains the labels of the examples.
  - $y_i$  is the label of example '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

$y =$

Sick?
1
1
0
1
1

$x_2 = (0.3, 0.7, 0, 0.6, 0, 0.01)$

$x^3 = (0, 0, 0, 1.2, 1.2)$

$x_{45} = 0.10$

$y_4 = 1$

# 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

$y =$

Sick?
1
1
0
1
1

Handwritten red annotations: A bracket labeled 'n' groups the rows of both tables. A bracket labeled 'd' groups the columns of the X table.

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

# Decision Stump Learning Pseudo-Code

Input: feature matrix  $X$  and label vector  $y$

Compute error if using "baseline" rule: number of times  $y_i$  does not equal most common value.  
for each feature ' $j$ ' (column of ' $X$ ')  
for each threshold ' $t$ '

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

set ' $y_{\text{no}}$ ' 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  $\hat{y}_i \neq y_i$  ( $\hat{y}_i = y_{\text{yes}}$  if satisfied,  
 $\hat{y}_i = y_{\text{no}}$  if not satisfied)

store the rule ( $j, t, y_{\text{yes}}, y_{\text{no}}$ ) if it has the lowest error so far.

Output: a "best" decision stump based on score. (the "model")

# Cost of Decision Stumps

- How much does this cost?
- Assume we have:
  - ‘n’ examples (days that we measured).
  - ‘d’ features (foods that we measured).
  - ‘k’ thresholds ( $>0$ ,  $>1$ ,  $>2$ , ...) for each feature.
- Computing the score of one rule costs  $O(n)$ :
  - We need to go through all ‘n’ examples to find most common labels.
  - We need to go through all ‘n’ examples again to compute the accuracy.
  - See notes on webpage for review of “ $O(n)$ ” notation.
- We compute score for up to  $k*d$  rules (‘k’ thresholds for each of ‘d’ features):
  - So we need to do an  $O(n)$  operation  $k*d$  times, giving total cost of  $O(ndk)$ .

# Cost of Decision Stumps

- Is a cost of  $O(ndk)$  good?
- Size of the input data is  $O(nd)$ :
  - If 'k' is small then the cost is roughly the same cost as loading the data.
    - We should be happy about this, you can learn on any dataset you can load!
  - If 'k' is large then this could be too slow for large datasets.
- Example: if all our features are **binary** then  $k=1$ , just test (feature > 0):
  - Cost of fitting decision stump is  $O(nd)$ , so we can fit huge datasets.
- Example: if all our features are **numerical** with unique values then  $k=n$ .
  - Cost of fitting decision stump is  $O(n^2d)$ .
    - We don't like having  $n^2$  because we want to fit datasets where 'n' is large!
  - Bonus slides: how to reduce the cost in this case down to  $O(nd \log n)$ .
    - Basic idea: sort features and track labels. Allows us to fit decision stumps to huge datasets.

(pause)



# Decision Tree Learning

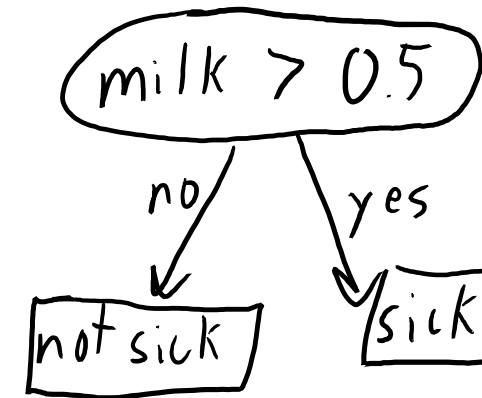
- **Decision stumps** have only 1 rule based on only 1 feature.
  - Very limited class of models: usually not very accurate for most tasks.
- **Decision trees** allow **sequences of splits** based on multiple features.
  - Very general class of models: can get very high accuracy.
  - However, it's **computationally 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

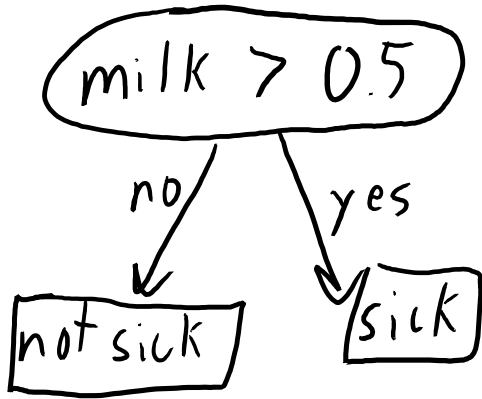
$milk \leq 0.5$

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

$milk > 0.5$

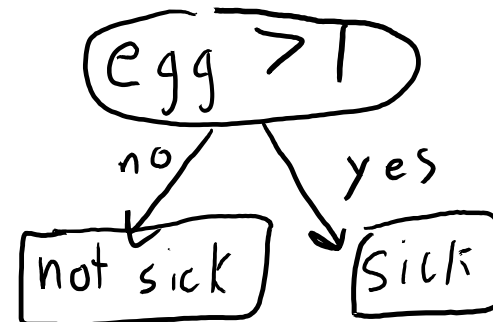
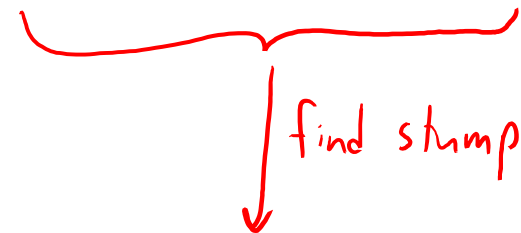
# Greedy Recursive Splitting

We now have a decision stump and two datasets:

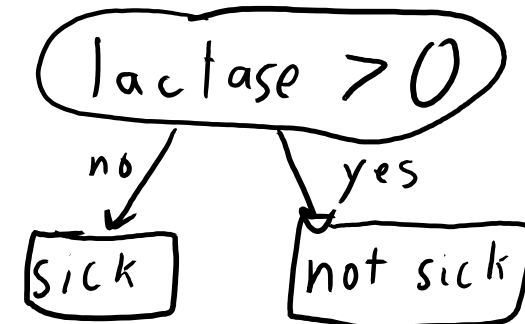
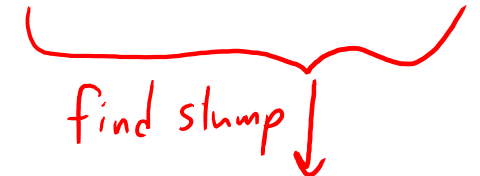


Fit a **decision stump** to each leaf's data.

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

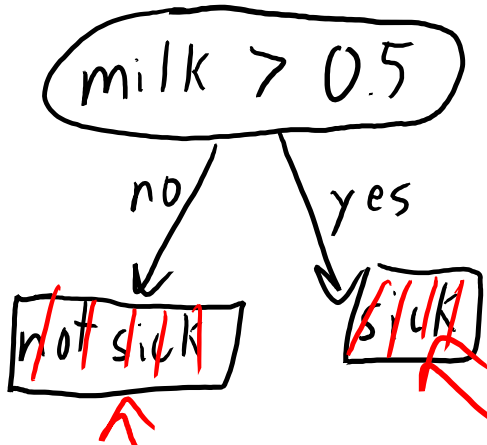


Egg	Milk	...	Sick?
0	0.7		1
1	0.7		1
1	0.6		1
2	0.6		1
0	1		1
1	0.6		0



# Greedy Recursive Splitting

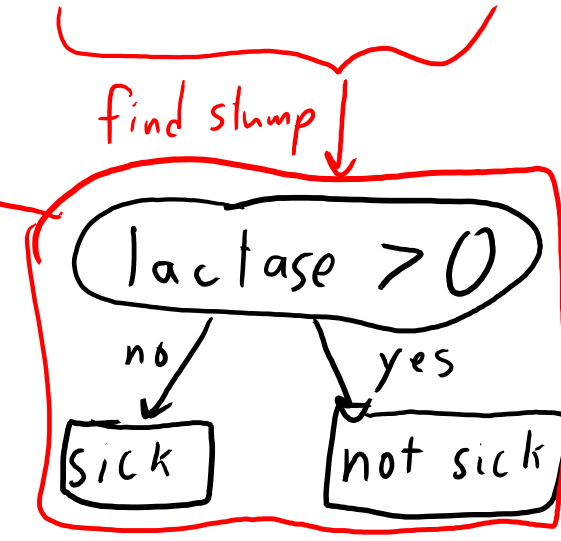
We now have a decision stump and two datasets:



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

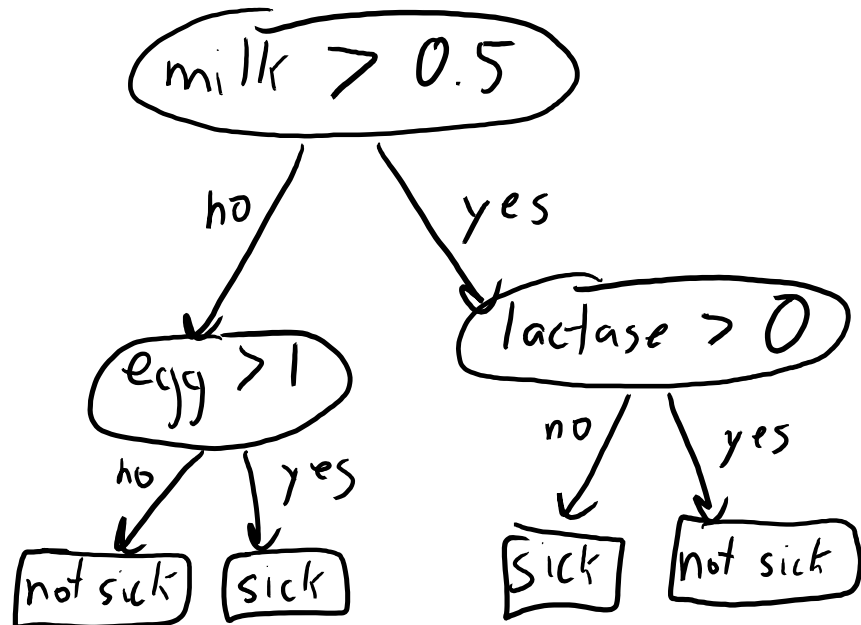
Egg	Milk	...	Sick?
0	0.7		1
1	0.7		1
1	0.6		1
2	0.6		1
0	1		1
1	0.6		0

Fit a **decision stump** to each leaf's data.  
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

*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

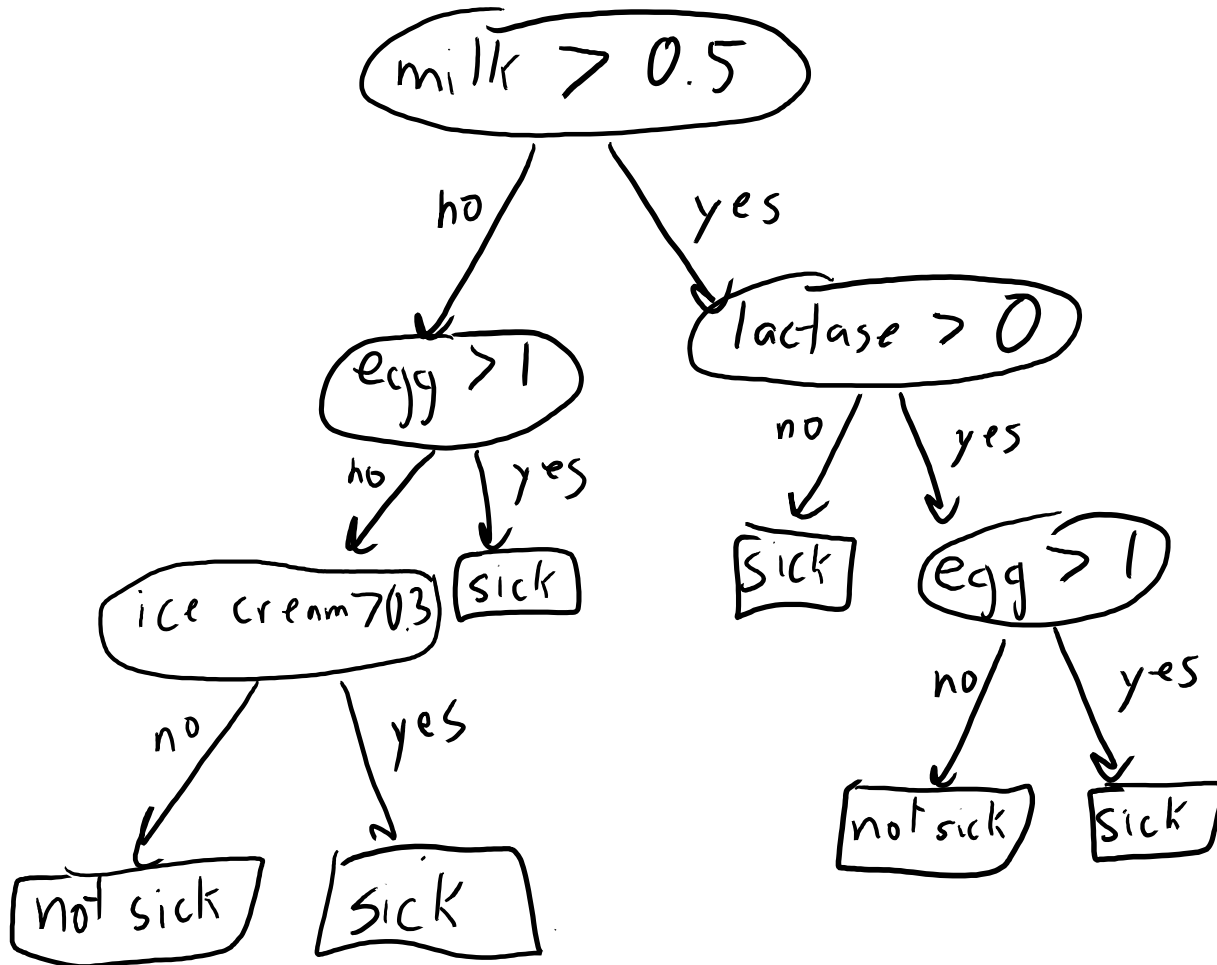
*milk > 0.5, lactase > 0*

Egg	Milk	...	Sick?
1	0.6		0

*Much more accurate!*

# 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.

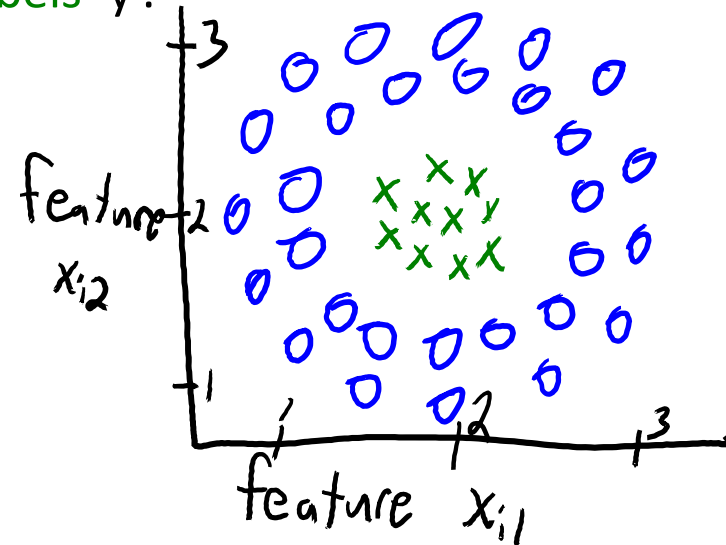
# Which score function should a decision tree use?

- Shouldn't we just use accuracy score?
  - For leafs: yes, just maximize accuracy.
  - For internal nodes: not necessarily.
- Maybe no simple rule like  $(\text{egg} > 0.5)$  improves accuracy.
  - But this doesn't necessarily mean we should stop!

# Example Where Accuracy Fails

- Consider a dataset with 2 features and 2 classes ('x' and 'o').
  - Because there are 2 features, we can draw 'X' as a scatterplot.
    - Colours and shapes denote the class labels 'y'.

$$X = \begin{bmatrix} 1.2 & 2.1 \\ 3.3 & 1.4 \\ 2.0 & 2.1 \\ 2.2 & 2.1 \\ 4.0 & 3.4 \\ \vdots & \vdots \end{bmatrix} \quad Y = \begin{bmatrix} 'o' \\ 'o' \\ 'x' \\ 'x' \\ 'o' \\ \vdots \end{bmatrix}$$



- A decision stump would divide space by a horizontal or vertical line.
  - Testing whether  $x_{i1} > t$  or whether  $x_{i2} > t$ .
- On this dataset no horizontal/vertical line improves accuracy.
  - Baseline is 'o', but need to get many 'o' wrong to get one 'x' right.



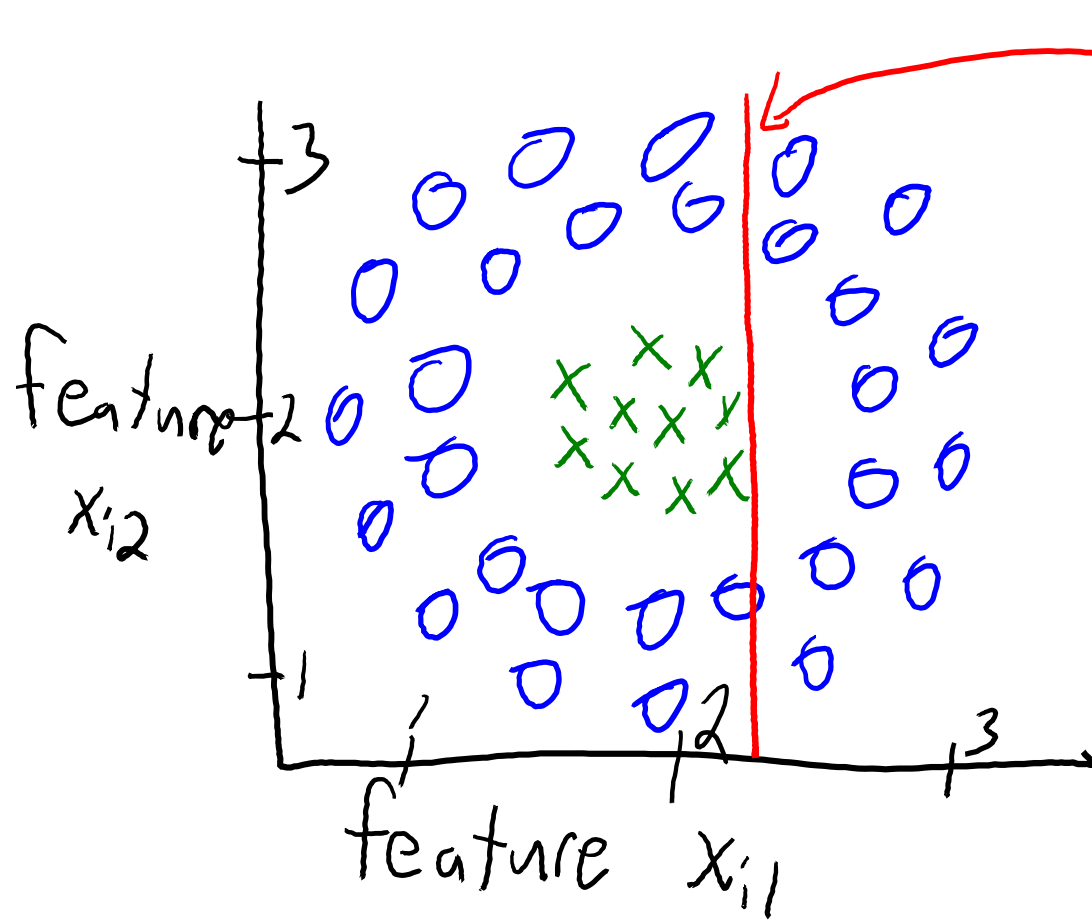
# Which score function should a decision tree use?

- Most common score in practice is “information gain”.
  - “Choose split that decreases entropy of labels the most”.

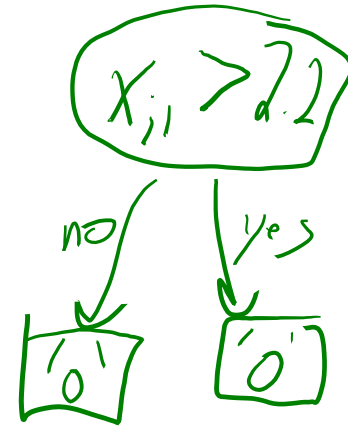
$$\text{information gain} = \underbrace{\text{entropy}(y)}_{\text{entropy of labels before split}} - \underbrace{\frac{n_{\text{yes}}}{n} \text{entropy}(y_{\text{yes}})}_{\text{number of examples satisfying rule}} - \underbrace{\frac{n_{\text{no}}}{n} \text{entropy}(y_{\text{no}})}_{\text{entropy of labels for example satisfying rule.}}$$

- Information gain for baseline rule (“do nothing”) is 0.
  - Infogain is large if labels are “more predictable” (“less random”) in next layer.
- Even if it does not increase classification accuracy at one depth, we hope that it makes classification easier at the next depth.

# Example Where Accuracy Fails

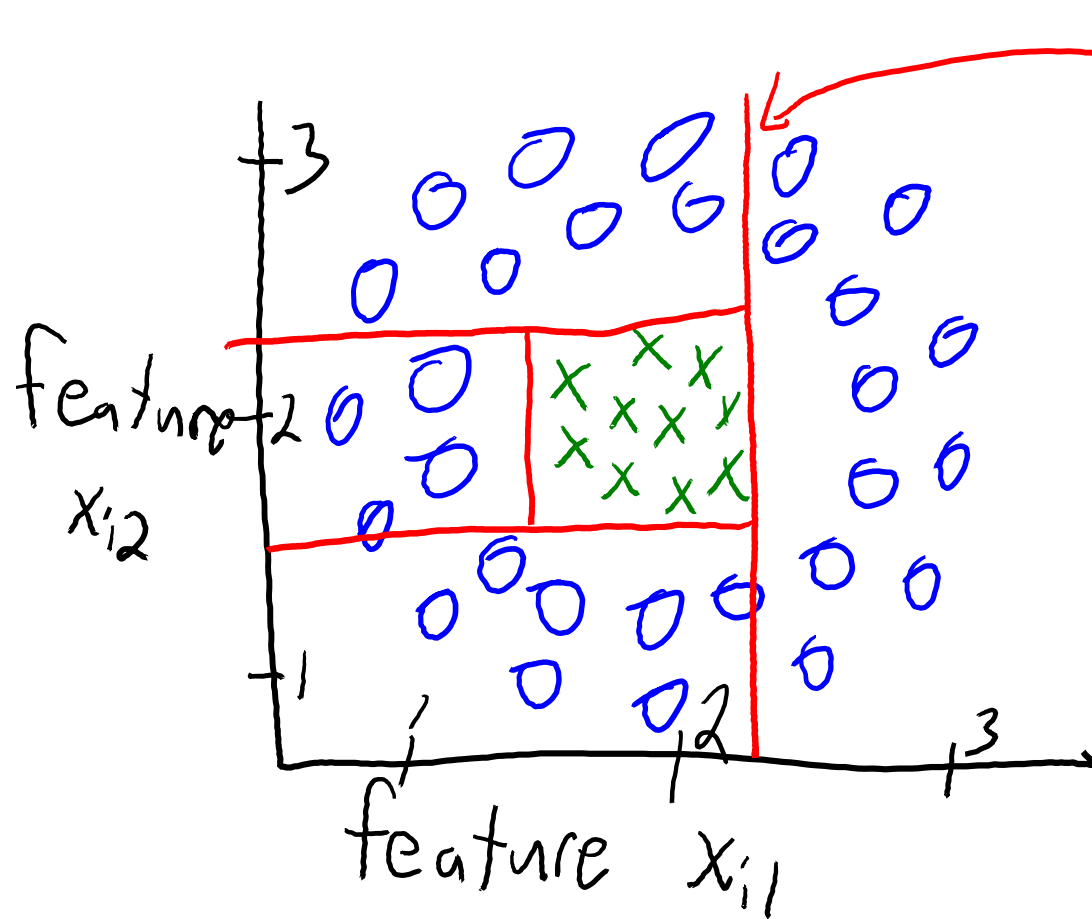


This split makes labels less random.  
(Everything on the right is a '0')

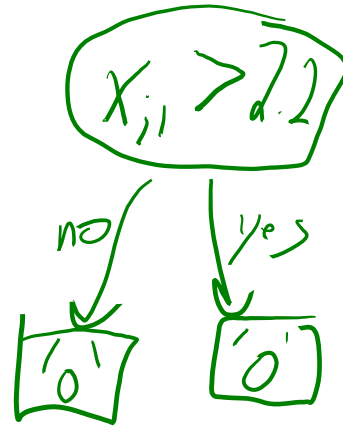


It did not improve accuracy.  
(still classifies everything  
as '0')

# Example Where Accuracy Fails



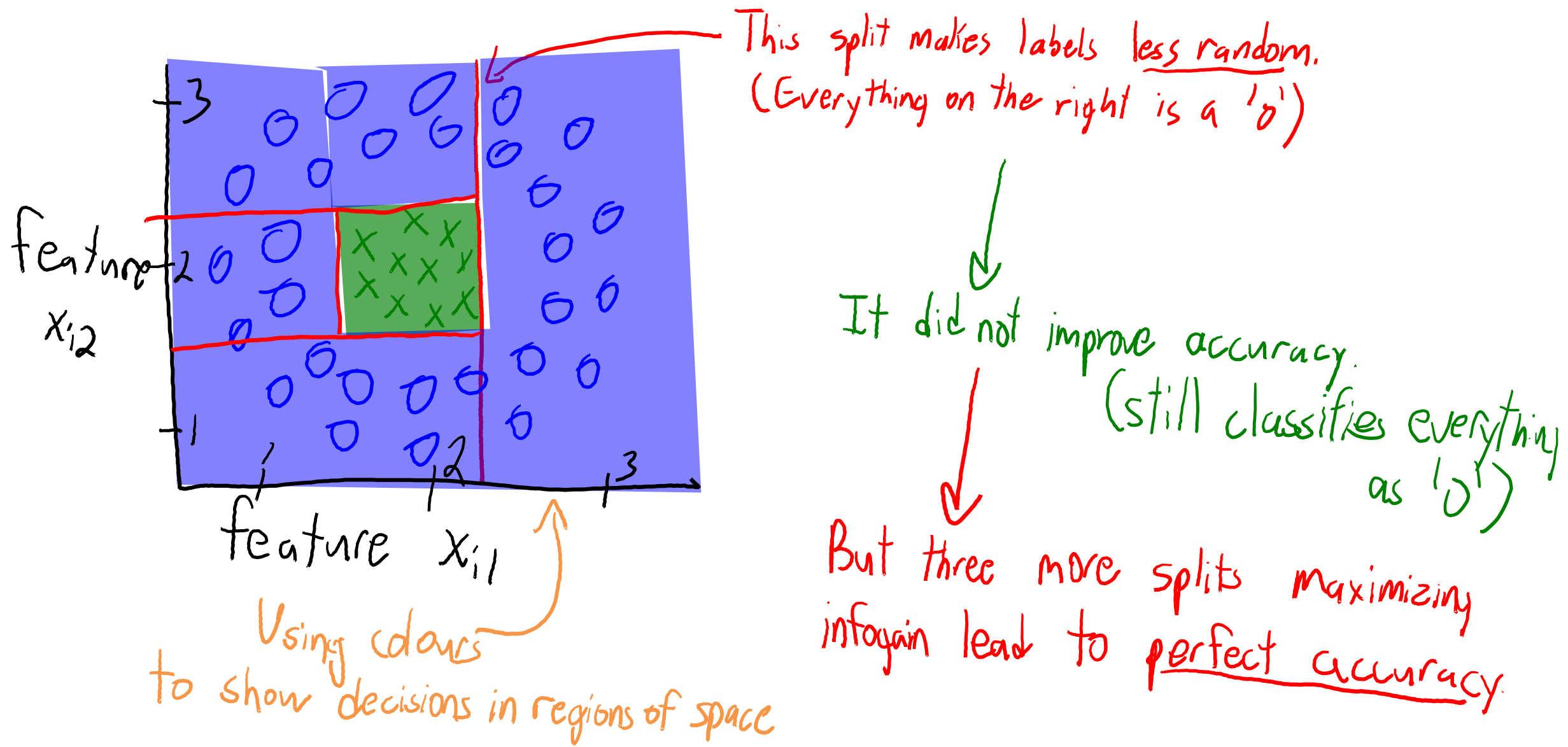
This split makes labels less random.  
(Everything on the right is a '0')



It did not improve accuracy.  
(still classifies everything as '0')

But three more splits maximizing  
info gain lead to perfect accuracy.

# Example Where Accuracy Fails



# Discussion of Decision Tree Learning

- Advantages:
  - Easy to implement.
  - Interpretable.
  - Learning is fast prediction is very fast.
  - Can elegantly handle a small number missing values during training.
- 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.
  - What is best score?
    - Infogain is the most popular and often works well, but is not always the best.
  - What is you get new data?
    - You could consider splitting if there is enough data at the leaves, but occasionally might want to re-learn the whole tree or sub-trees.
  - 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.
- Information gain: splitting score based on decreasing entropy.
- Next time: the most important ideas in machine learning.

# Entropy Function

Input: vector 'y' of length 'n' with numbers  $\{1, 2, \dots, k\}$

```
counts = zeros(k)
```

```
for i in 1:n
```

```
    counts[y[i]] += 1
```

```
entropy = 0
```

```
for c in 1:k
```

```
    prob = counts[c]/n
```

```
    entropy -= prob * log(prob)
```

```
return entropy
```

# 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”  $\Leftrightarrow$  “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  $y$

$(n, d) = \text{size}(X)$

$\text{minError} = \text{sum}(y \neq \text{mode}(y))$  compute error if you don't split (user-defined function "mode")

$\text{minRule} = []$

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'!

$y_{\text{above}} = \text{mode}(y[X[:, j]. > t])$

find mode of label vector when feature 'j' is above threshold!

$y_{\text{below}} = \text{mode}(y[X[:, j]. \leq t])$

find mode of label vector when feature 'j' is below threshold!

$\hat{y} = \text{fill}(y_{\text{above}}, n)$

Classify all examples based on threshold

$\hat{y}[X[:, j]. \leq t] = y_{\text{below}}$

$\text{error} = \text{sum}(\hat{y} \neq y)$

count the number of errors.

if  $\text{error} < \text{minError}$   
 $\text{minError} = \text{error}$   
 $\text{minRule} = [j \quad t]$

store this rule if it has the lowest error so far.

# 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  $(\text{egg} > 1)$  and  $(\text{egg} > 2)$  have same decisions, except when  $(\text{egg} == 2)$ .
- 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  $O(kn)$ .

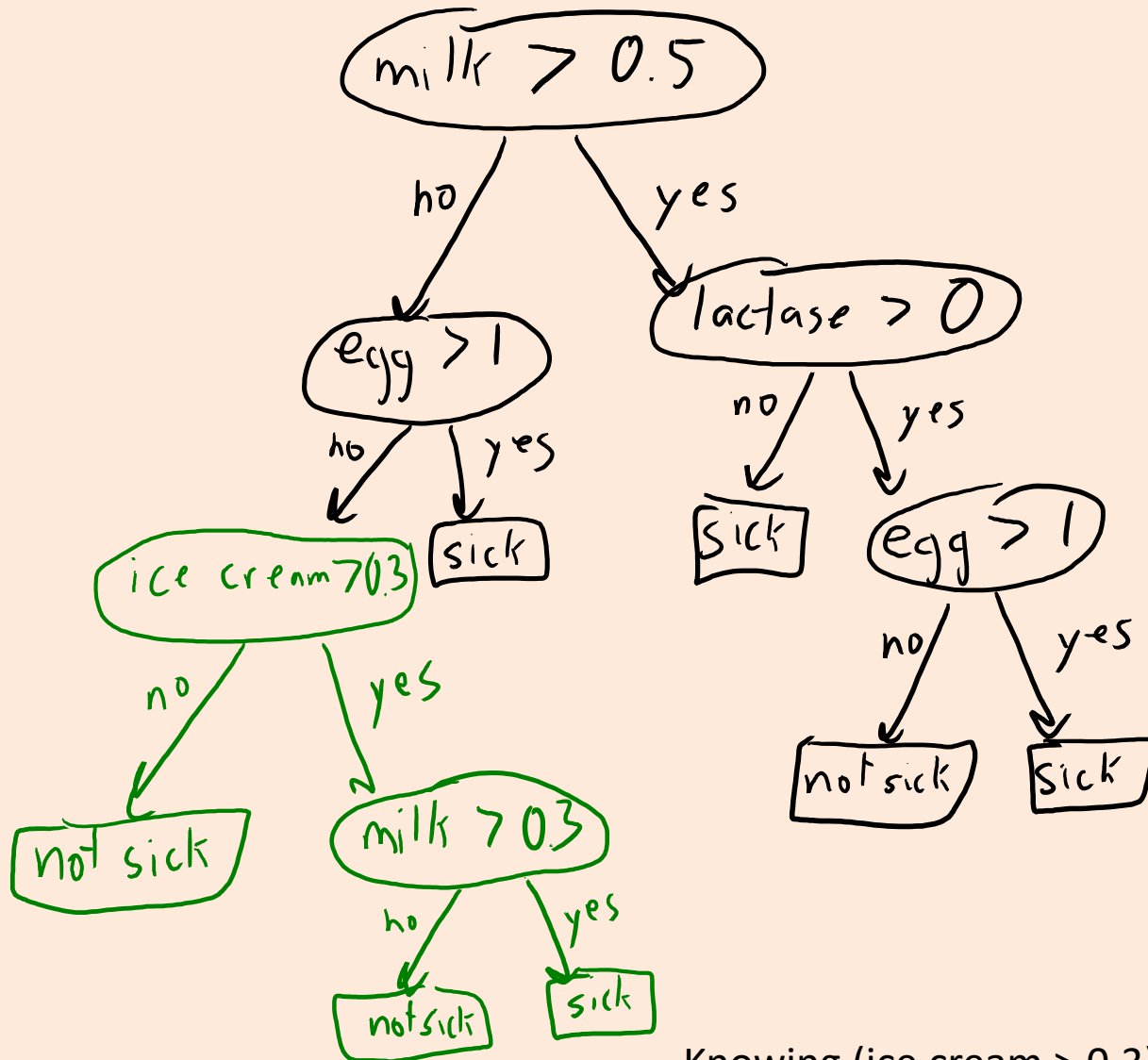
With 'k' labels, you can decrease cost using a "max-heap" data structure:

- Cost of getting max is  $O(1)$ , cost of updating heap for a row is  $O(\log k)$ .
- But  $k \leq n$  (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?

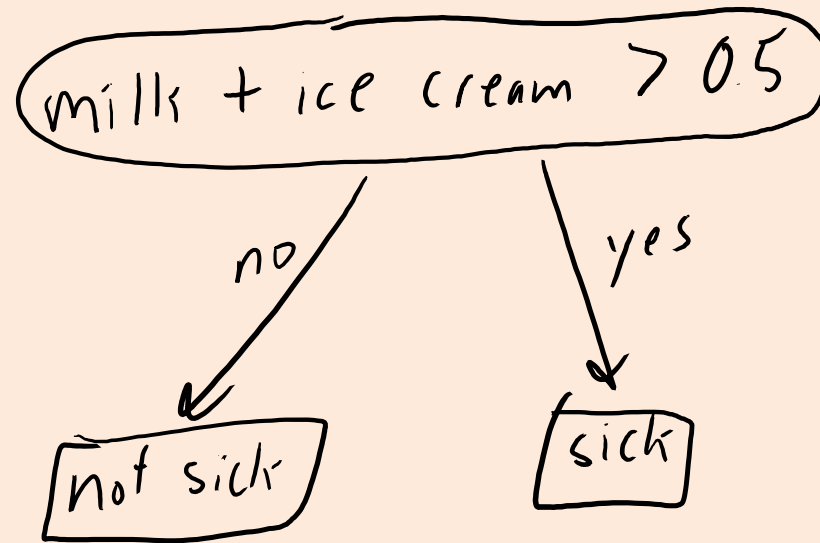
- Yes.



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

# Can decision trees have more complicated rules?

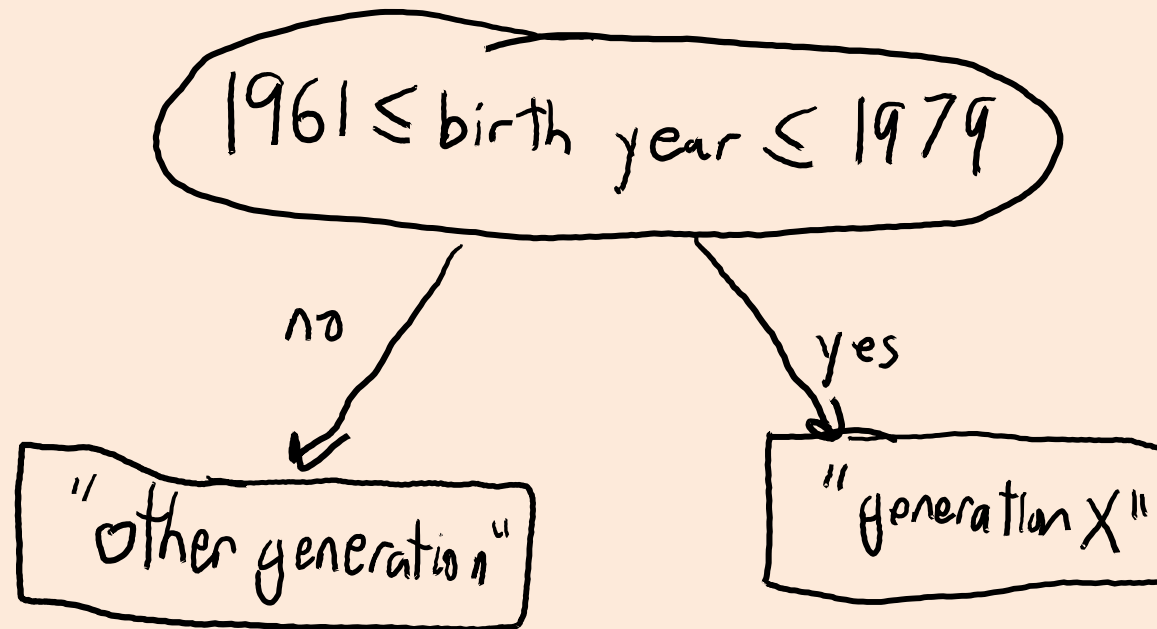
- Yes!
- Rules that depend on **more than one feature**:



- 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**:

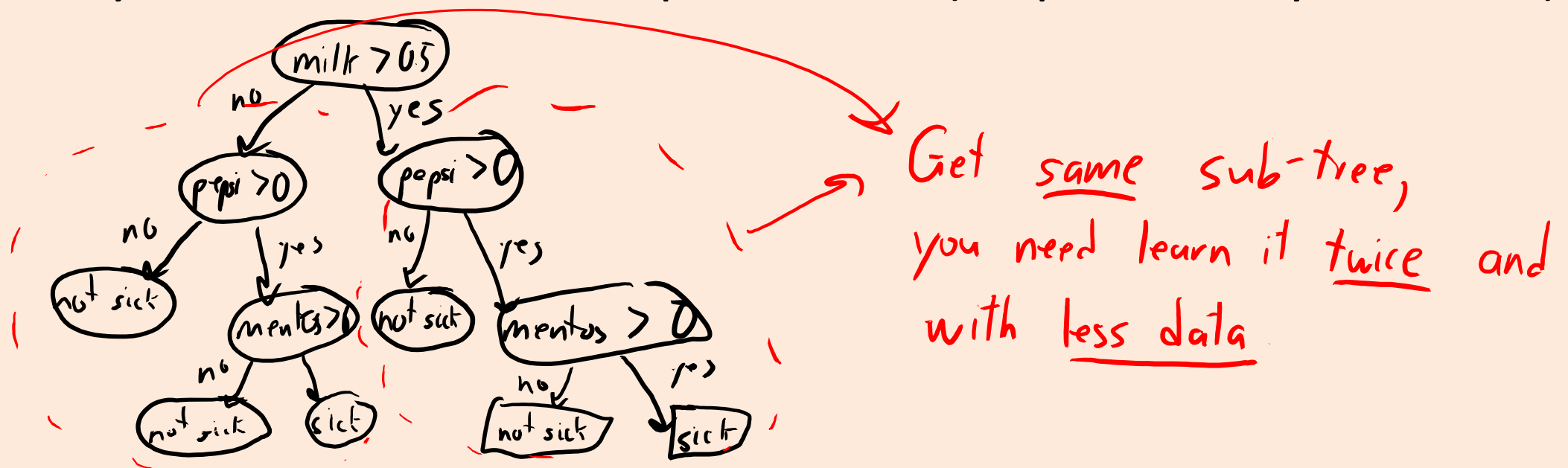


- [“Very Simple Classification Rules Perform Well on Most Commonly Used Datasets”](#)
  - Consider decision stumps based on multiple splits of 1 attribute.
  - Showed that this gives comparable performance to more-fancy methods on many datasets.



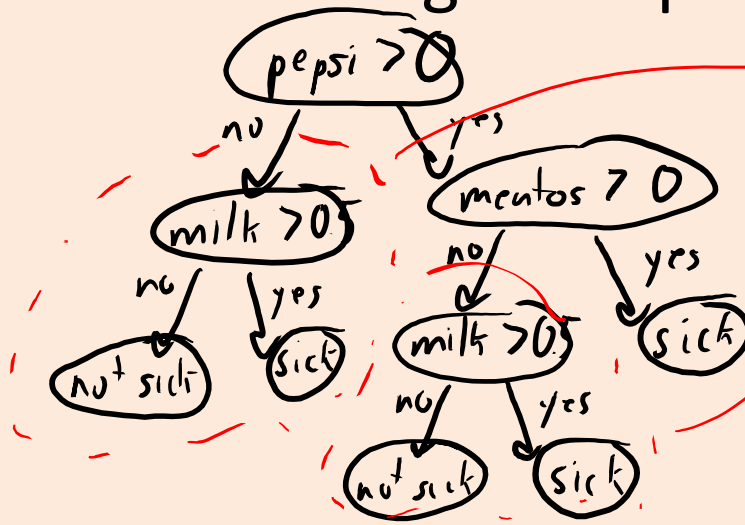
# Does being greedy actually hurt?

- 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):



# 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 repeated structure, but you should have more data to estimate those 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.