CPSC 340:
Machine Learning and Data Mining

Decision Trees

Summer 2021

REMINDER TO HIT RECORD

Admin

- Lecture 3: I need extra 5 minutes to catch up
  - I will improve my time management
- Assignment 1 is due Monday: start early.
  - Gradescope setup is still in progress
- Waiting list people: Start on A1!!
- Course webpage: https://www.cs.ubc.ca/~nhgk/courses/cpsc340s21/
- Sign up for Piazza: https://piazza.com/ubc.ca/summer2021/cpsc340911
- Attend tutorials and office hours!

Waiting List Situation

Note: this section is full

Seat Summary

Total Seats Remaining: **0**

Currently Registered: **143**

General Seats Remaining: **0**

Restricted Seats Remaining*: **0**

- Select one Tutorial from sections T1A, T1B, T1C, T1D

Main section

Seat Summary

Total Seats Remaining: **944**

Currently Registered: **55**

General Seats Remaining: **944**

Restricted Seats Remaining*: **0**

Waiting list

Student Roster: Download Roster as CSV

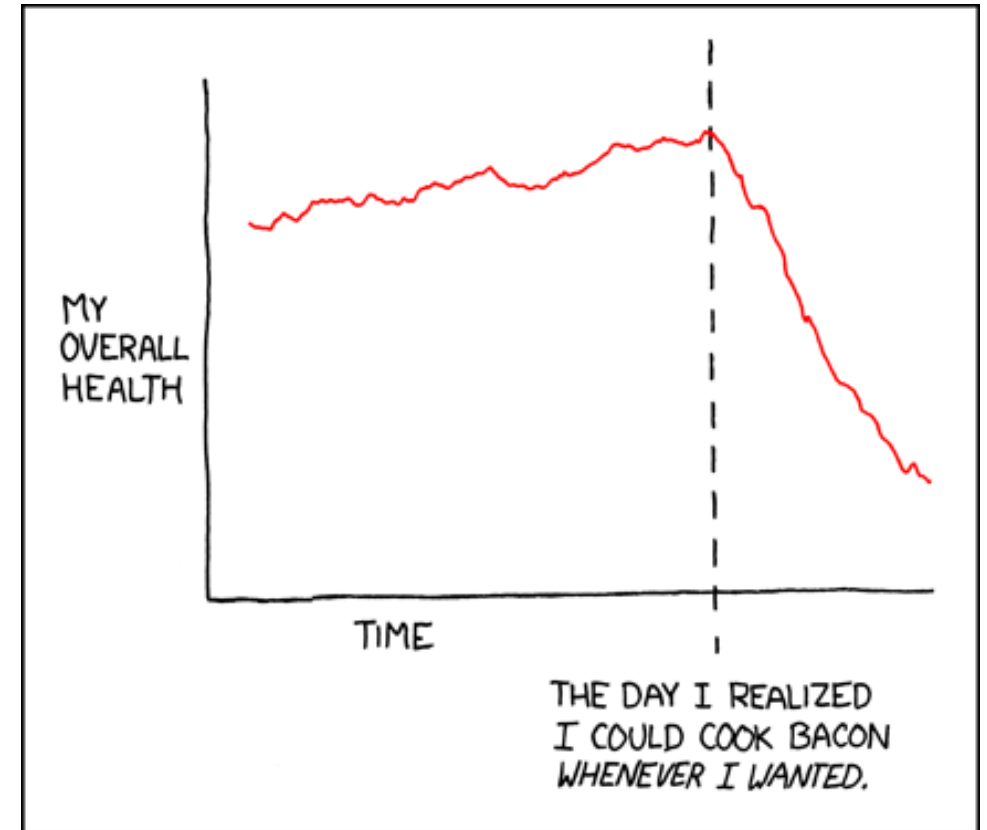...out of 176 (estimated) Edit

176 enrolled

Piazza

In this Lecture

- Visualizing Data (10 minutes)
- Supervised Learning Intro (10 minutes)
- Course Notation (10 minutes)
- Decision Trees (30 minutes)
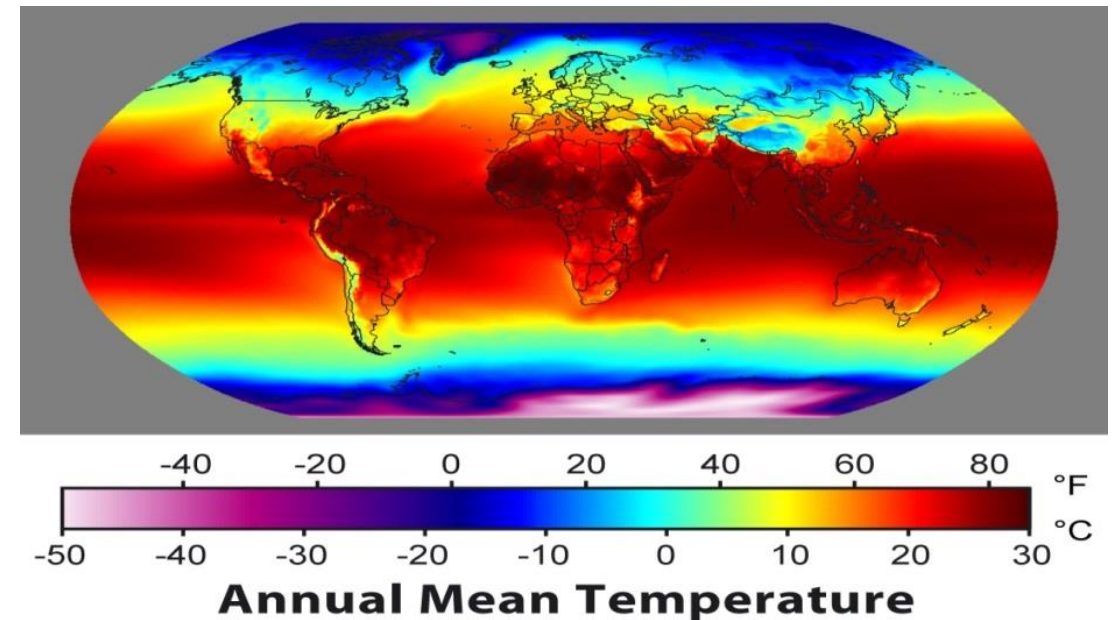
Coming Up Next
VISUALIZING DATA

Visualization

- **You can learn a lot from 2D plots of the data:**
  - Patterns, trends, outliers, unusual patterns.

Lat	Long	Temp
0	0	30.1
0	1	29.8
0	2	29.9
0	3	30.1
0	4	29.9
...	...	...

vs.
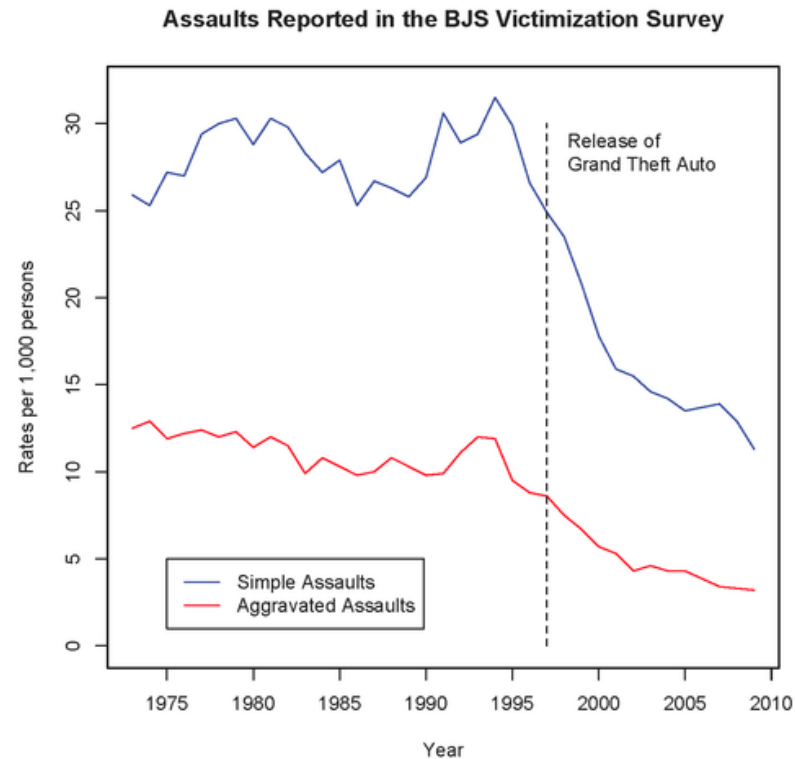

Annual Mean Temperature

"Impenetrable sea of numbers"
-Ted Kim

Intuitive visuals

Basic Plot

- Visualize one variable as a function of another.

Assaults Reported in the BJS Victimization Survey
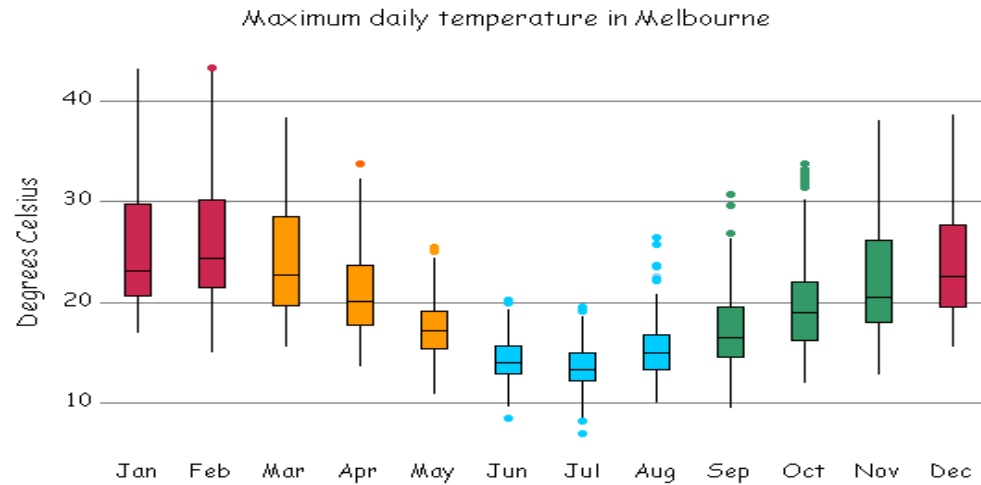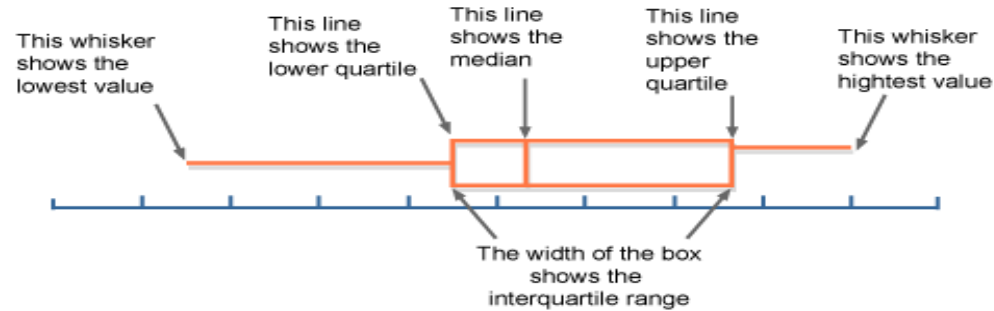
- [Fun with plots](#).

Histogram

- Histograms display counts a variable, split into "bins".
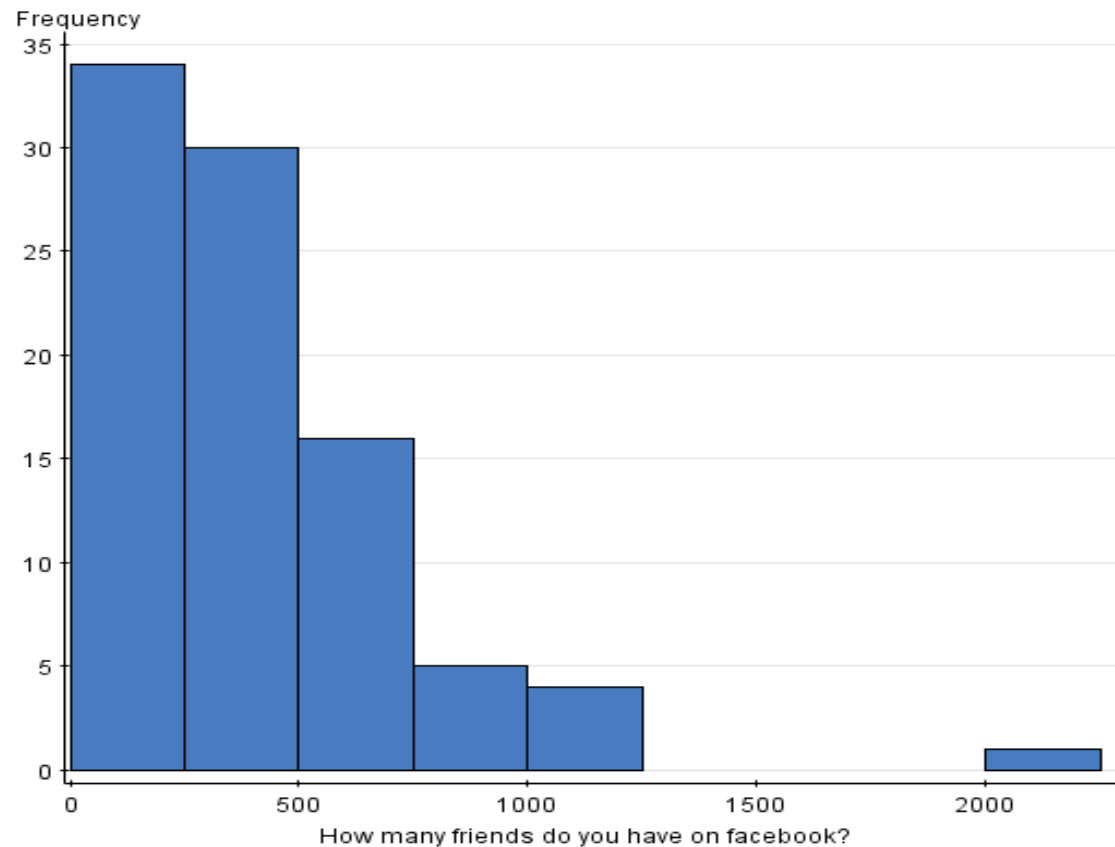
Box Plot

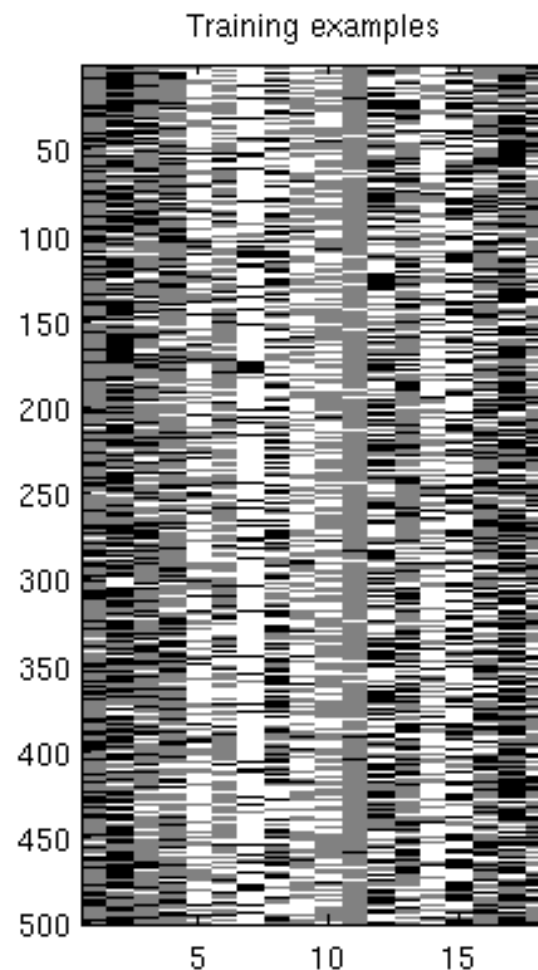Useful for visualizing summary statistics of multiple features in one view

Matrix Plot

- We can view (examples) x (features) data table as a picture:
  - "Matrix plot".
  - May be able to see trends in features.

Training examples

Correlation Plot

- A matrix plot of all similarities (or distances) between features:
  - Colour used to catch attention.

	BTC	ETH	XRP	XEM	ETC	LTC	DASH	XMR
BTC	1.00	0.61	0.36	0.51	0.60	0.56	0.55	0.66
ETH	0.61	1.00	0.28	0.49	0.68	0.43	0.70	0.64
XRP	0.36	0.28	1.00	0.48	0.08	0.35	0.40	0.44
XEM	0.51	0.49	0.48	1.00	0.40	0.43	0.47	0.52
ETC	0.60	0.68	0.08	0.40	1.00	0.47	0.56	0.53
LTC	0.56	0.43	0.35	0.43	0.47	1.00	0.59	0.67
DASH	0.55	0.70	0.40	0.47	0.56	0.59	1.00	0.74
XMR	0.66	0.64	0.44	0.52	0.53	0.67	0.74	1.00

Scatterplot

- **Look at distribution of two features:**
  - Feature 1 on x-axis.
  - Feature 2 on y-axis.

OkCupid ratings back when we let people score "looks" and "personality" separately
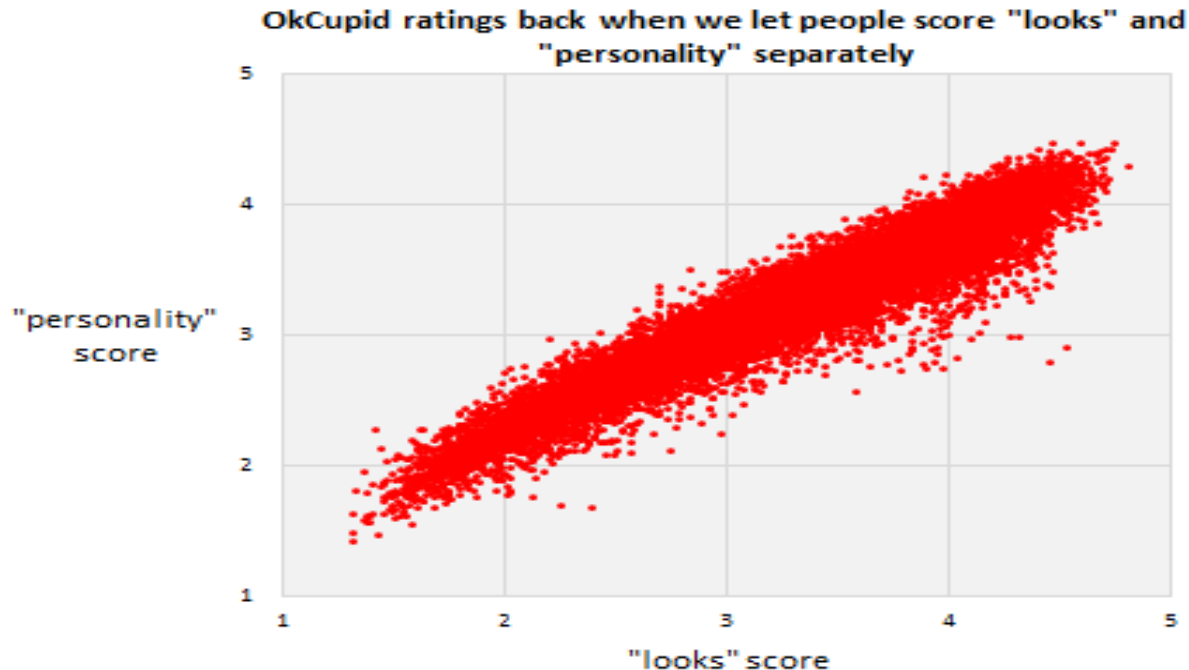
- Shows correlation between "personality" score and "looks" score.

"Why Not to Trust Plots"

- We've seen how summary statistics can be mis-leading.
- Note that plots can also be mis-leading, or can be used to mis-lead.

- Bonus slides: examples from UW's excellent course:
  - "Calling Bullshit in the Age of Big Data".
    - A course on how to recognize when people are trying to mis-lead you with data.
  - I recommend watching all the videos here:
    - https://www.youtube.com/watch?v=A2OtU5vlR0k&list=PLPnZfvKID1Sje5jWxt-4CSZD7bUl4gSPS

  - Recognizing BS not only useful for data analysis, but for daily life.

Coming Up Next
SUPERVISED LEARNING

Motivating Example: Food Allergies

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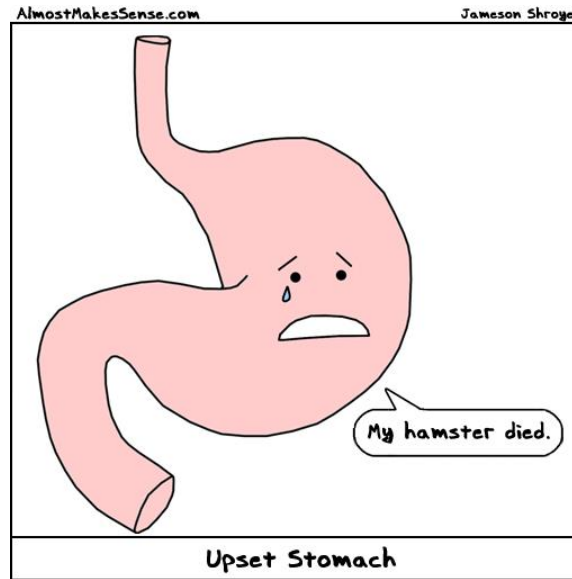
- You suspect an adult-onset food allergy.

Q: How do I know which food made me sick?

Motivating Example: Food Allergies

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

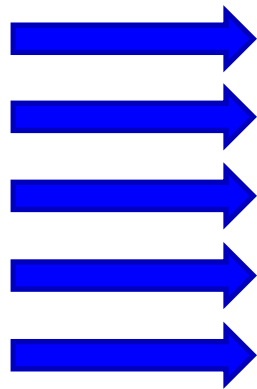| Egg	Milk	Fish	Wheat	Shellfish	Peanuts	...	Sick?
0	0.7	0	0.3	0	0		1
0.3	0.7	0	0.6	0	0.01		1
0	0	0	0.8	0	0		0
0.3	0.7	1.2	0	0.10	0.01		1
0.3	0	1.2	0.3	0.10	0.01		1

"Impenetrable sea of numbers"
-Ted Kim

- 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 a supervised learning problem:

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

- 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 as Writing A Program

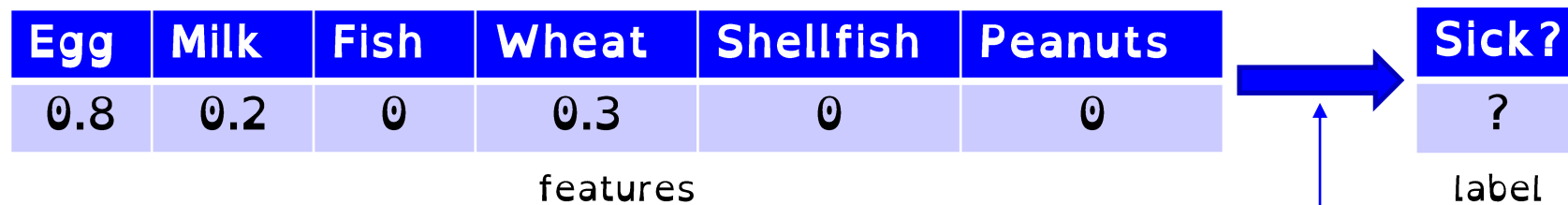**The Supervised Learning Problem**
Input: lots of labeled examples
Output: a <u>learned program</u> that returns appropriate values

The Learned Program
Input: numerical features of an example
Output: a predicted label corresponding to the features

Egg	Milk	Fish	Wheat	Shellfish	Peanuts		Sick?
0.8	0.2	0	0.3	0	0	→	?

features   label

The learned program
(aka "model" or "mapping")

Supervised Learning as 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.

Try writing a self-driving car program by hand!!!
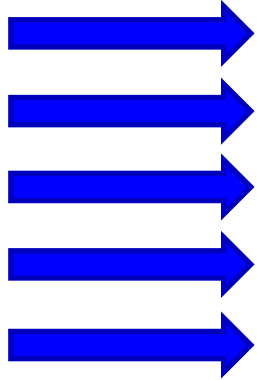(hint: it's impossible)

"Hey doctor, my FOXP3 gene looks like
ATCGAGACGCGAGCCGCGACCACCGCGAGCGA.
Does this mean I have a compromised immune system?"

Supervised Learning

- 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 classification
  - learning with categorical labels
  - E.g. {'sick', 'not sick'}, {SF, New York}
- Here, the model is a called a "classifier".

Naïve Supervised Learning: "Return-the-Mode"

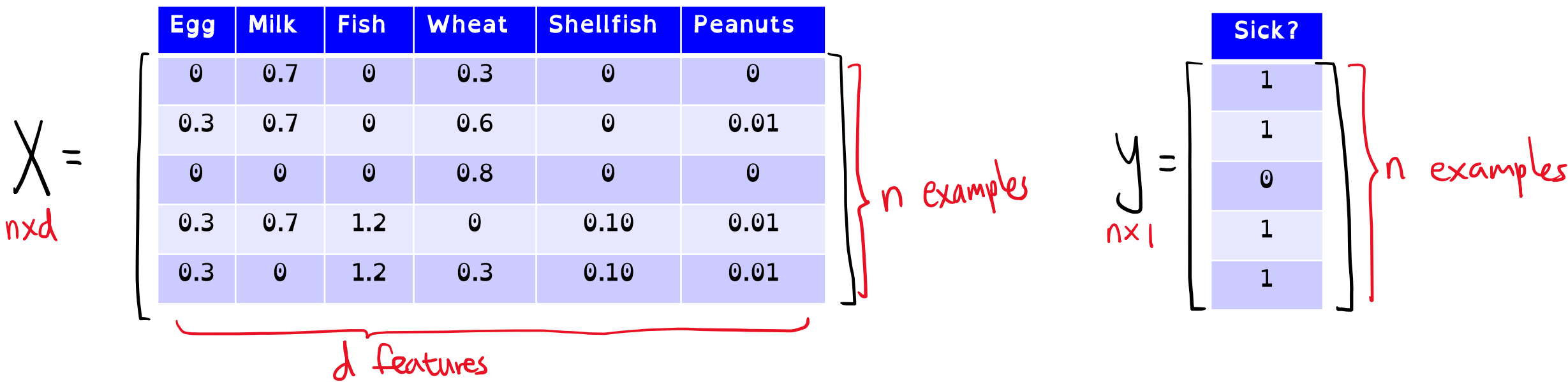| Egg	Milk	Fish	Wheat	Shellfish	Peanuts	...		Sick?
0	0.7	0	0.3	0	0		⟶	1
0.3	0.7	0	0.6	0	0.01		⟶	1
0	0	0	0.8	0	0		⟶	0
0.3	0.7	1.2	0	0.10	0.01		⟶	1
0.3	0	1.2	0.3	0.10	0.01		⟶	1

- A very naïve supervised learning method:
  - Always predict the most common label, the mode ("sick" above).
- This ignores the features!
  - cannot be more accurate than proportion of mode (80% in this case)
- We want to use the features, and there are MANY ways to do this.
  - We'll consider a classic way known as decision tree learning.

Coming Up Next

COURSE NOTATION

Supervised Learning Notation (MEMORIZE THIS)

$$X =$$

$n \times d$

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 examples

d features

$$y =$$

$n \times 1$

Sick?
1
1
0
1
1

} n examples

- **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").

$$x_{45} = \text{?} \qquad x_2 = \text{?} \qquad x^3 = \text{?} \qquad y_4 = \text{?}$$

$$X = \begin{array}{|c|c|c|c|c|c|}
\hline
\textbf{Egg} & \textbf{Milk} & \textbf{Fish} & \textbf{Wheat} & \textbf{Shellfish} & \textbf{Peanuts} \\
\hline
0 & 0.7 & 0 & 0.3 & 0 & 0 \\
0.3 & 0.7 & 0 & 0.6 & 0 & 0.01 \\
0 & 0 & 0 & 0.8 & 0 & 0 \\
0.3 & 0.7 & 1.2 & 0 & 0.10 & 0.01 \\
0.3 & 0 & 1.2 & 0.3 & 0.10 & 0.01 \\
\hline
\end{array}$$

n × d   } n examples   d features

$$y = \begin{array}{|c|}
\hline
\textbf{Sick?} \\
\hline
1 \\
1 \\
0 \\
1 \\
1 \\
\hline
\end{array}$$

n × 1   } n examples

- **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)

$$X = \begin{bmatrix} & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix}$$

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

nxd

} n examples

d features

Sick?
1
1
0
1
1

$y =$

nx1

} n examples

- **Training phase:**
  – Use 'X' and 'y' to find a 'model' (a mapping of feature → label)
- **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.

Coming Up Next
DECISION TREES

Hand-crafted decision tree at Starbucks
(Agronomy Road)

What is a Decision Tree?

- Nested if-else statements
  - E.g.

```
1    if milk > 0.5:
2        return 'sick'
3    else:
4        if egg > 1:
5            return 'sick'
6        else:
7            return 'not sick'
```
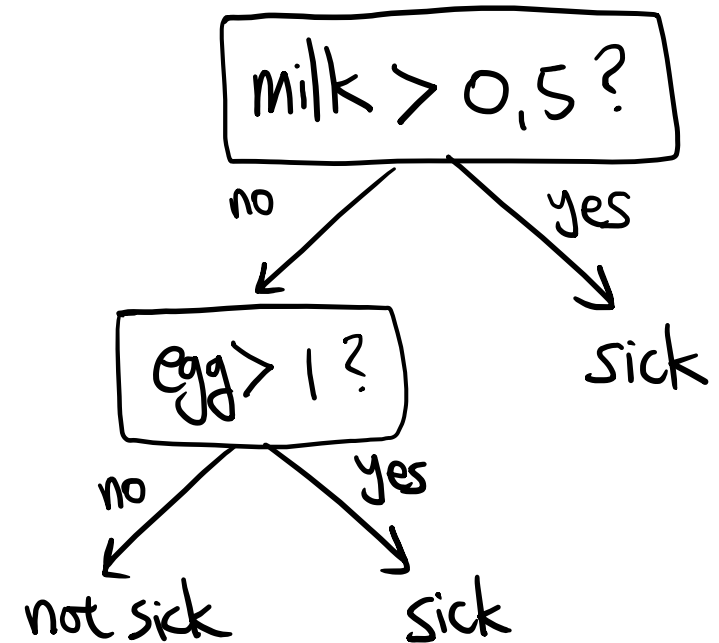
Q: How is this a "tree"?

What is a Decision Tree?

- Nested if-else statements
  - E.g.

```
1    if milk > 0.5:
2        return 'sick'
3    else:
4        if egg > 1:
5            return 'sick'
6        else:
7            return 'not sick'
```

milk > 0.5?

no   yes

egg > 1?   sick

no   yes

not sick   sick

Q: According to this decision tree,
what should we return for this example?

Egg	Milk	Fish	Wheat	Shellfish	Peanuts		Sick?
0.8	0.2	0	0.3	0	0	→	?

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.

milk > 0.5?

no → not sick

yes → sick
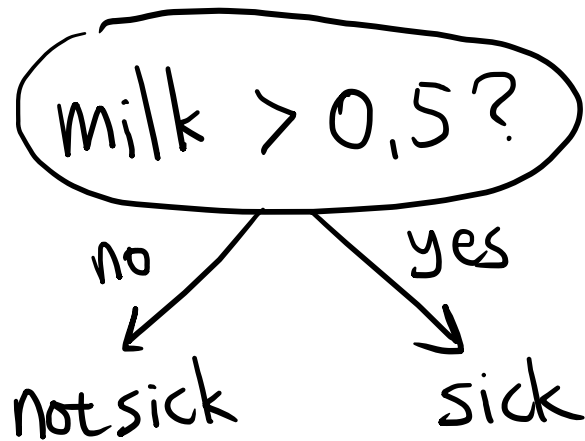
Q: What is the thresholding feature here?

Q: What is the splitting rule here?

Q: How do we find the best "rule" (feature, threshold, and leaf labels)?

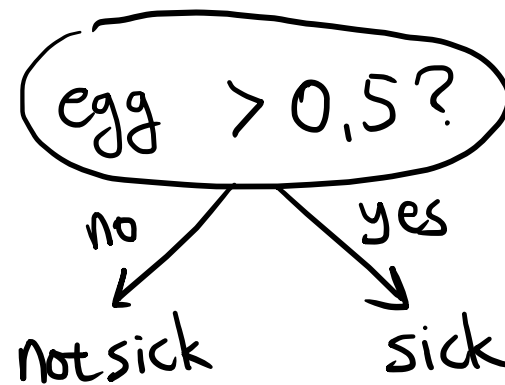# Learning A Decision Stump: "Search and Score"
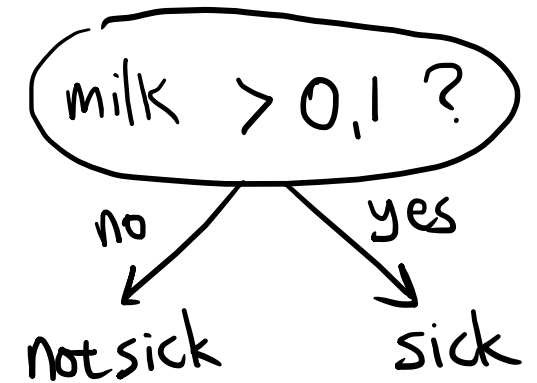
1. Define a 'score' for the rule.
2. Search for the rule with the best score.

milk > 0,5?
no → Not sick
yes → sick

Score=50

milk > 0,5?
no → Sick
yes → not sick

Score=10

egg > 0,5?
no → Not sick
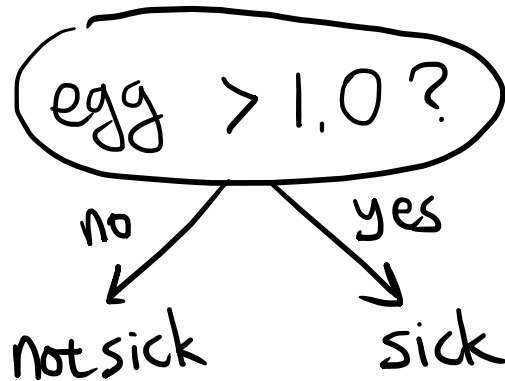yes → sick

Score=45

milk > 0,1?
no → Not sick
yes → sick
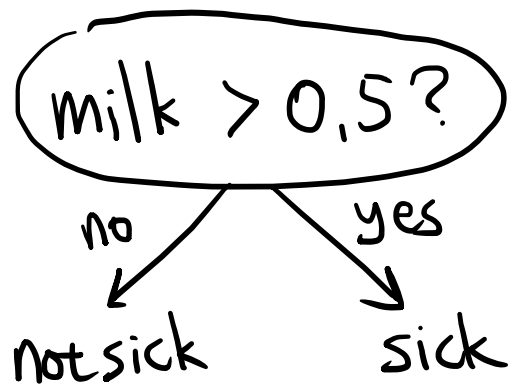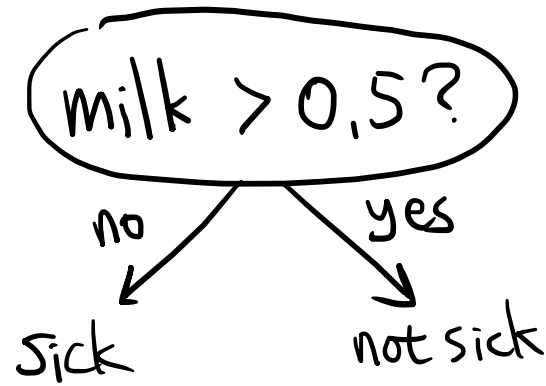
Score=65
(Best rule!)

Q: How do we "score" a decision stump?

Learning A Decision Stump: Accuracy Score

- Most intuitive score: classification accuracy.
  - "If we use this rule, how many examples do we label correctly?"

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

egg > 1.0 ?

no → Not sick
yes → sick

- Computing classification accuracy for this stump:
  - Find most common labels after applying splitting rule:
    - When (egg > 1), we were "sick" _ times out of _.
    - When (egg ≤ 1), we were "not sick" _ times out of _.
  - Compute accuracy:
    - The accuracy ("score") of the rule (egg > 1) is _ times out of _.
- This "score" evaluates quality of a rule.
  - We "learn" a decision stump by finding the rule with the best score.
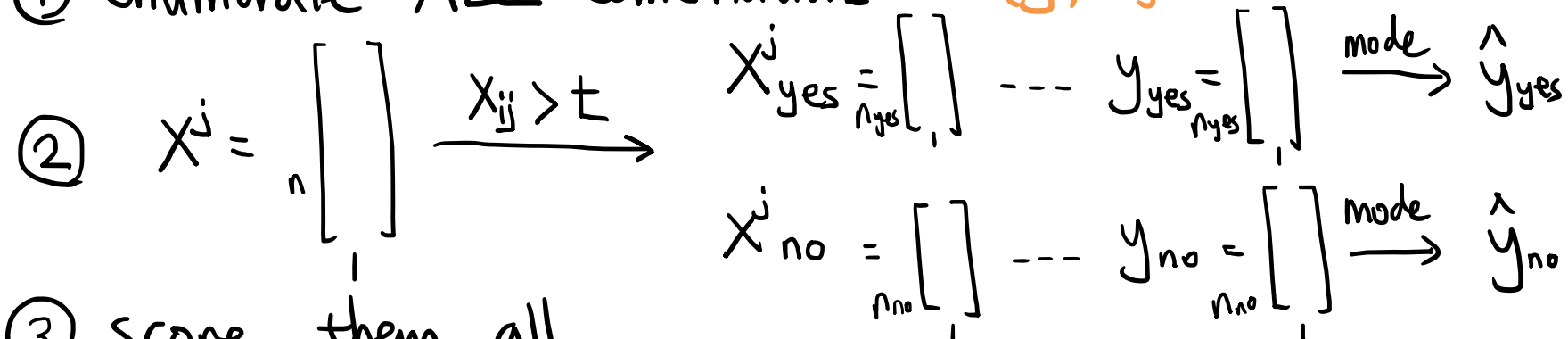
Decision Stump Learning Pseudo-Code

Input: $\quad X = {}_n\begin{bmatrix} & \\ & \end{bmatrix}_d \qquad Y = {}_n\begin{bmatrix} & \\ & \end{bmatrix}_1 \qquad$ Output:

$$\boxed{X_{ij} > t}$$

no $\qquad$ yes

$\hat{y}_{no}$ $\qquad\qquad$ $\hat{y}_{yes}$
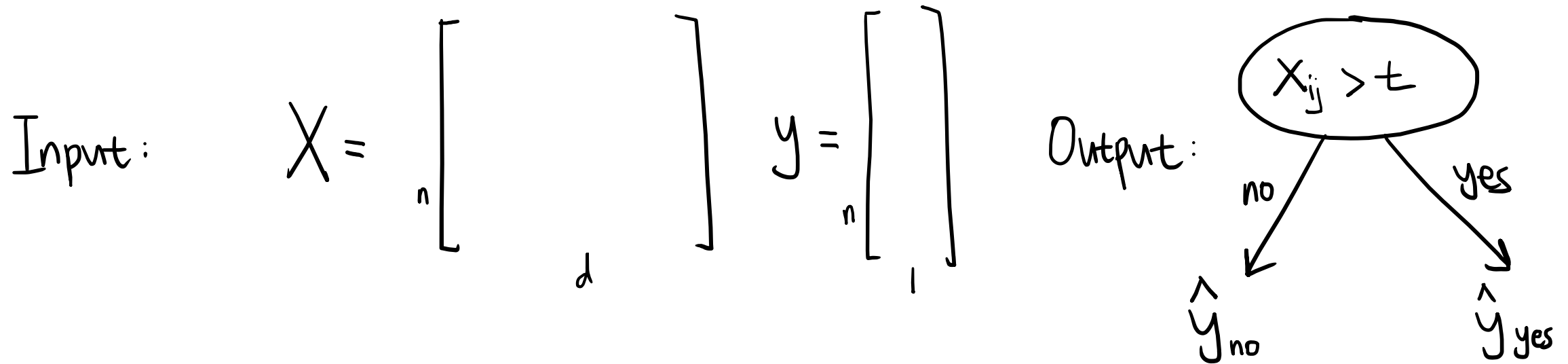
Q: Which of these components are learned?

Decision Stump Learning Pseudo-Code

Input:

$$X = \quad n\begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix}$$

$$d \qquad \uparrow \atop j$$

$$y = \quad n\begin{bmatrix} \\ \\ \end{bmatrix} \atop 1$$

Output:

$$x_{ij} > t$$

no $\qquad$ yes

$\hat{y}_{no}$ $\qquad$ $\hat{y}_{yes}$
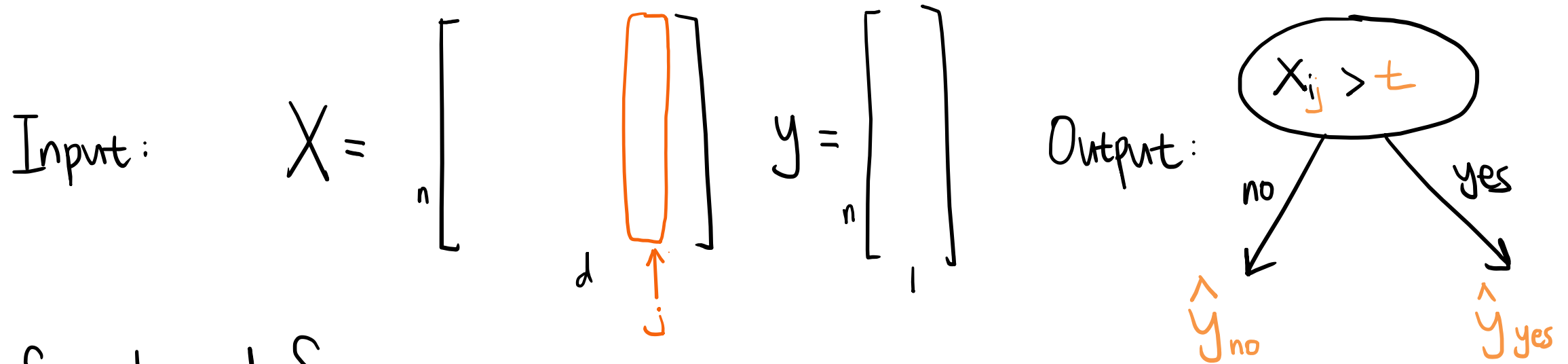
Search—and—Score

① enumerate ALL combinations of $\{j, t\}$

② $X^j = \quad n\begin{bmatrix} \\ \\ \end{bmatrix} \atop 1 \xrightarrow{\; X_{ij} > t \;}$

$X^j_{yes} = n_{yes}\begin{bmatrix} \\ \\ \end{bmatrix} \;-\!-\!-\; y_{yes} = n_{yes}\begin{bmatrix} \\ \\ \end{bmatrix}_1 \xrightarrow{\text{mode}} \hat{y}_{yes}$

$X^j_{no} = n_{no}\begin{bmatrix} \\ \\ \end{bmatrix}_1 \;-\!-\!-\; y_{no} = n_{no}\begin{bmatrix} \\ \\ \end{bmatrix}_1 \xrightarrow{\text{mode}} \hat{y}_{no}$

③ score them all

④ pick the one with best score! $\longrightarrow$ $\{j^*, t^*, \hat{y}^*_{yes}, \hat{y}^*_{no}\}$

See post-lecture
slides for more
formal pseudo-code

33

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(_):
  - We need to go through all 'n' examples to find modes for y_{yes} and y_{no}.
  - 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(___).

Cost of Decision Stumps

(k := number of thresholds)

- Is a cost of O(ndk) good?
- Size of the input data is O(__):
  - 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(__), so we can fit huge datasets.
- Example: if all our features are numerical with unique values then k=_.
  - Cost of fitting decision stump is O(__).
    - Bad if 'n' is large!
  - Bonus slides: fitting decision stump in O(nd log n).
    - Basic idea: sort features and track labels.
      Allows us to fit decision stumps to huge datasets.

Coming Up Next

DECISION STUMP TO DECISION TREE

How Do I Make a Tree from Stumps?
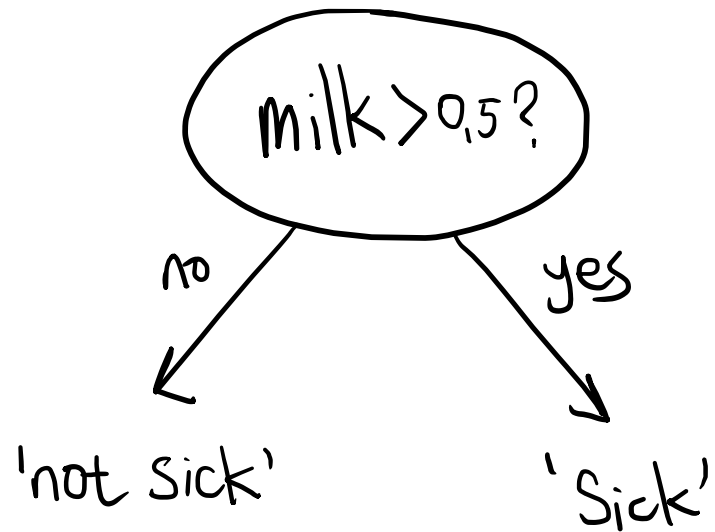

```
1    def decision_stump():
2        if milk > 0.5:
3            return 'sick'
4        else:
5            return 'not sick'
```

How Do I Make a Tree from Stumps?
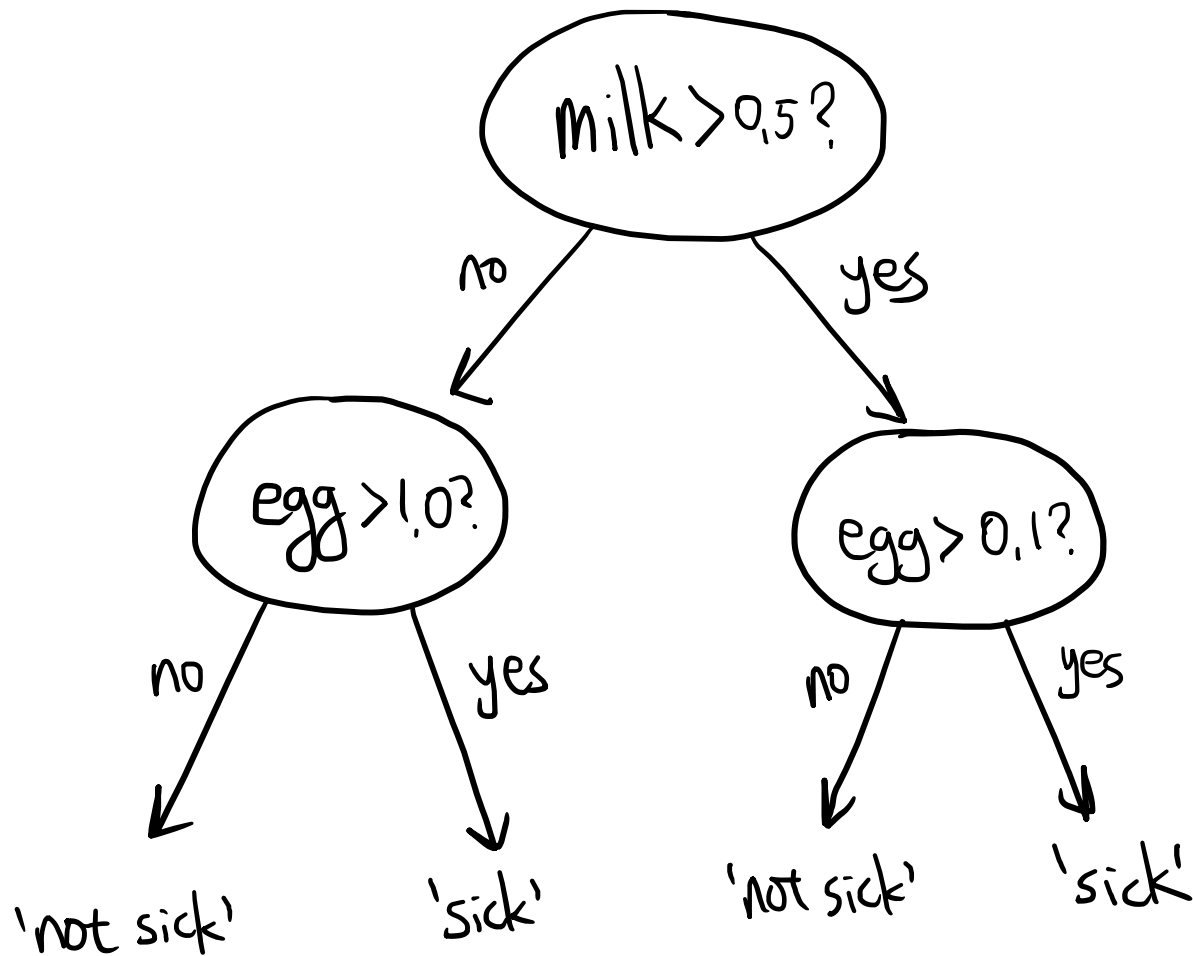
```
 7    def decision_tree():
 8        if milk > 0.5:
 9            if egg > 0.1:
10                return 'sick'
11            else:
12                return 'not sick'
13        else:
14            if egg > 1.0:
15                return 'sick'
16            else:
17                return 'not sick'
```

milk > 0.5?

no → egg > 1.0?

yes → egg > 0.1?

egg > 1.0?
no → 'not sick'
yes → 'sick'

egg > 0.1?
no → 'not sick'
yes → 'sick'

these are decision stumps!

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.

Greedy Recursive Splitting

Original data   First splitting rule   Second splitting rules   Third splitting rules
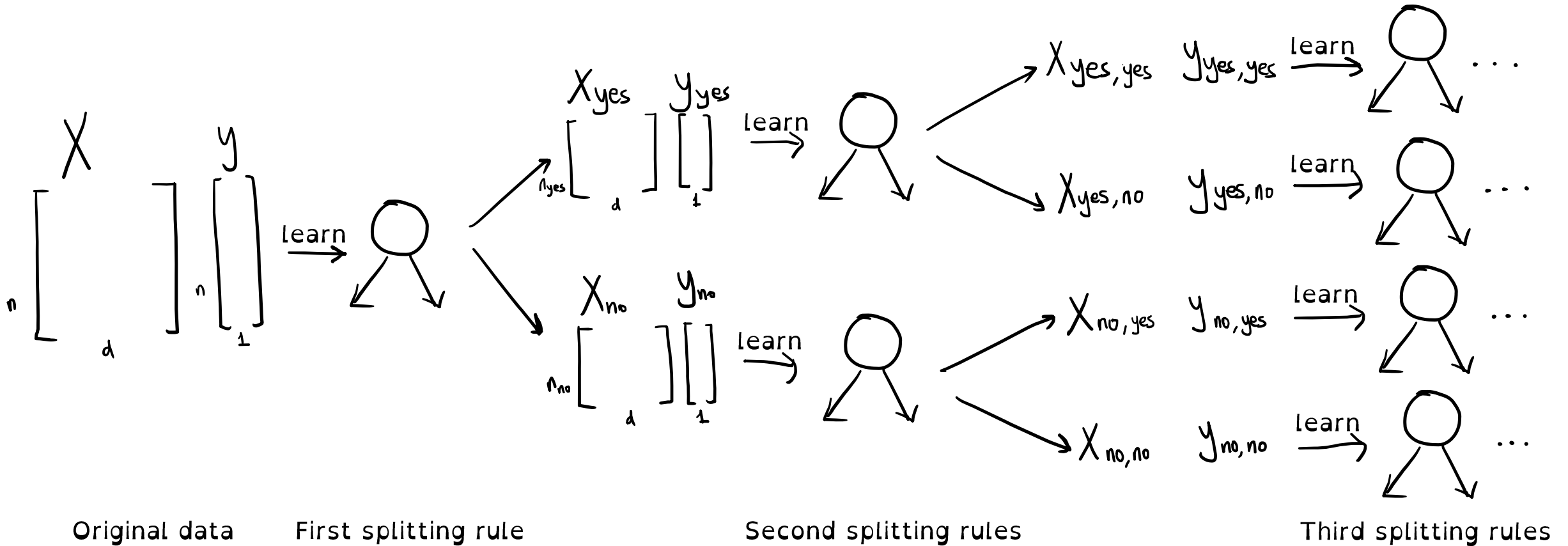
Q: What is the base case for this recursion?

Stopping Criteria

X

y

$$n \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \begin{bmatrix} \text{'sick'} \\ \text{'sick'} \\ \text{'sick'} \\ \vdots \\ \text{'sick'} \end{bmatrix}$$

d 1
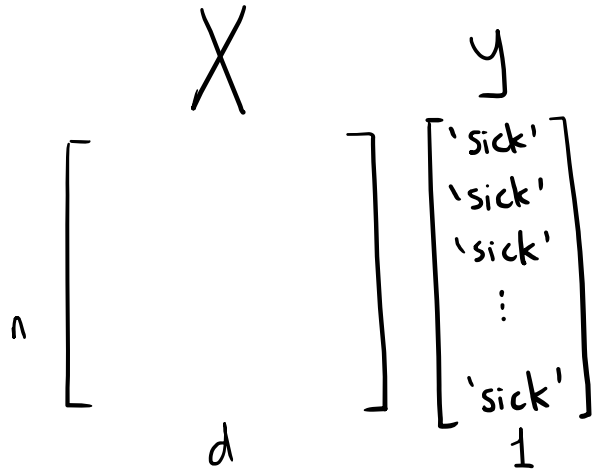
Only 'sick' in y.
Q: Does it make sense to make a stump here?
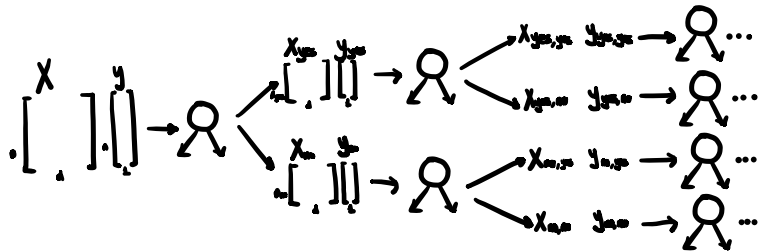
Q: Does it make sense to keep splitting?
(suppose n is very big)

When To Stop Greedy Recursive Splitting:
1. Only a single class remains in the labels
2. Maximum recursion depth reached

A1 Skeleton Code

```python
def fit(self, X, y):
    N, D = X.shape

    # Get an array with the number of 0's, number of 1's, etc.
    count = np.bincount(y)

    # Get the index of the largest value in count.
    # Thus, y_mode is the mode (most popular value) of y
    y_mode = np.argmax(count)

    self.splitSat = y_mode
    self.splitNot = None
    self.splitVariable = None
    self.splitValue = None

    # If all the labels are the same, no need to split further
    if np.unique(y).size <= 1:
        return

    minError = np.sum(y != y_mode)

    # Loop over features looking for the best split
    X = np.round(X)

    for d in range(D):
        for n in range(N):
            # Choose value to equate to
            value = X[n, d]

            # Find most likely class for each split
            y_sat = utils.mode(y[X[:,d] == value])
            y_not = utils.mode(y[X[:,d] != value])

            # Make predictions
            y_pred = y_sat * np.ones(N)
            y_pred[X[:, d] != value] = y_not

            # Compute error
            errors = np.sum(y_pred != y)

            # Compare to minimum error so far
            if errors < minError:
                # This is the lowest error, store this value
                minError = errors
                self.splitVariable = d
                self.splitValue = value
                self.splitSat = y_sat
                self.splitNot = y_not
```

decision_stump.py

```python
def fit(self, X, y):
    # Fits a decision tree using greedy recursive splitting
    N, D = X.shape

    # Learn a decision stump
    splitModel = self.stump_class()
    splitModel.fit(X, y)

    if self.max_depth <= 1 or splitModel.splitVariable is None:
        # If we have reached the maximum depth or the decision stump does
        # nothing, use the decision stump

        self.splitModel = splitModel
        self.subModel1 = None
        self.subModel0 = None
        return

    # Fit a decision tree to each split, decreasing maximum depth by 1
    j = splitModel.splitVariable
    value = splitModel.splitValue

    # Find indices of examples in each split
    splitIndex1 = X[:,j] > value
    splitIndex0 = X[:,j] <= value

    # Fit decision tree to each split
    self.splitModel = splitModel
    self.subModel1 = DecisionTree(self.max_depth-1, stump_class=self.stump_class)
    self.subModel1.fit(X[splitIndex1], y[splitIndex1])
    self.subModel0 = DecisionTree(self.max_depth-1, stump_class=self.stump_class)
    self.subModel0.fit(X[splitIndex0], y[splitIndex0])
```

decision_tree.py

Coming Up Next

DISCUSSION ON DECISION TREES

Which score function should we use?

- What's wrong with using accuracy score with GRS?
  - Improving accuracy at level 1 doesn't mean you get best accuracy overall

- Sometimes we can't improve accuracy with a stump at all.
  - But this doesn't 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'.

$$X = \begin{bmatrix} 1.2 & 2.1 \\ 3.3 & 1.4 \\ 2.0 & 2.1 \\ 7.2 & 2.1 \\ 4.0 & 3.4 \\ \vdots & \vdots \end{bmatrix} \quad y = \begin{bmatrix} 'o' \\ 'o' \\ 'o' \\ 'x' \\ 'x' \\ 'o' \\ \vdots \end{bmatrix}$$
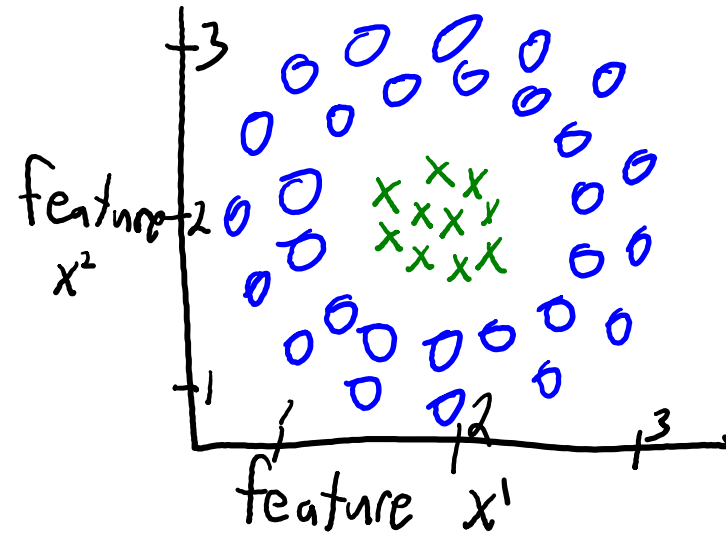
Q: How do we visualize a decision stump here?

Example Where Accuracy Fails

- A decision stump would divide space by a horizontal or vertical line.
  - E.g. $x_{i1} > t$
- On this dataset, a horizontal/vertical line cannot do better than return-the-mode.
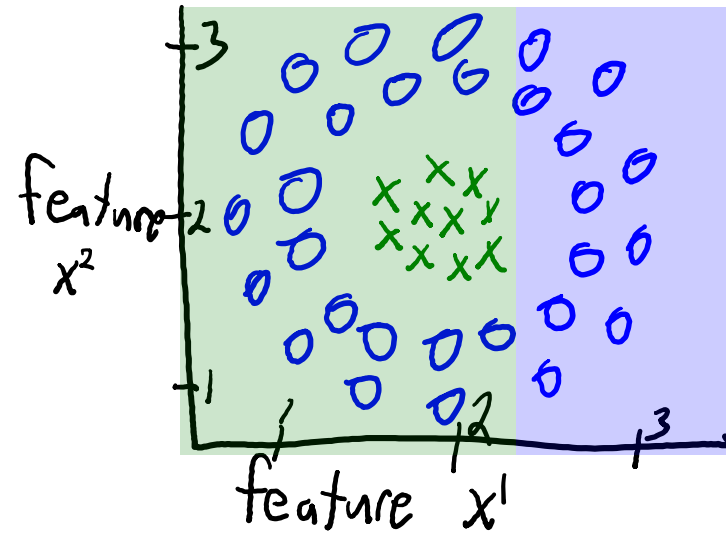
Which score function should we use?

- Most common score in practice is "information gain".
  - Information gain := [entropy before split] – [entropy after split]

<div style="border:1px solid black; text-align:center">

Q: When is information gain large?

</div>

$$\text{information gain} = \text{entropy}(y) - \frac{n_{yes}}{n}\text{entropy}(y_{yes}) - \frac{n_{no}}{n}\text{entropy}(y_{no})$$

entropy of labels before split

number of examples satisfying rule

entropy of labels for examples satisfying rule.

- Information gain for baseline rule ("do nothing") is __.
- Infogain is largest if labels are_____.

- **Less greedy** than classification accuracy GRS
  - Even if accuracy isn't better with current stump, accuracy may increase with more depth.

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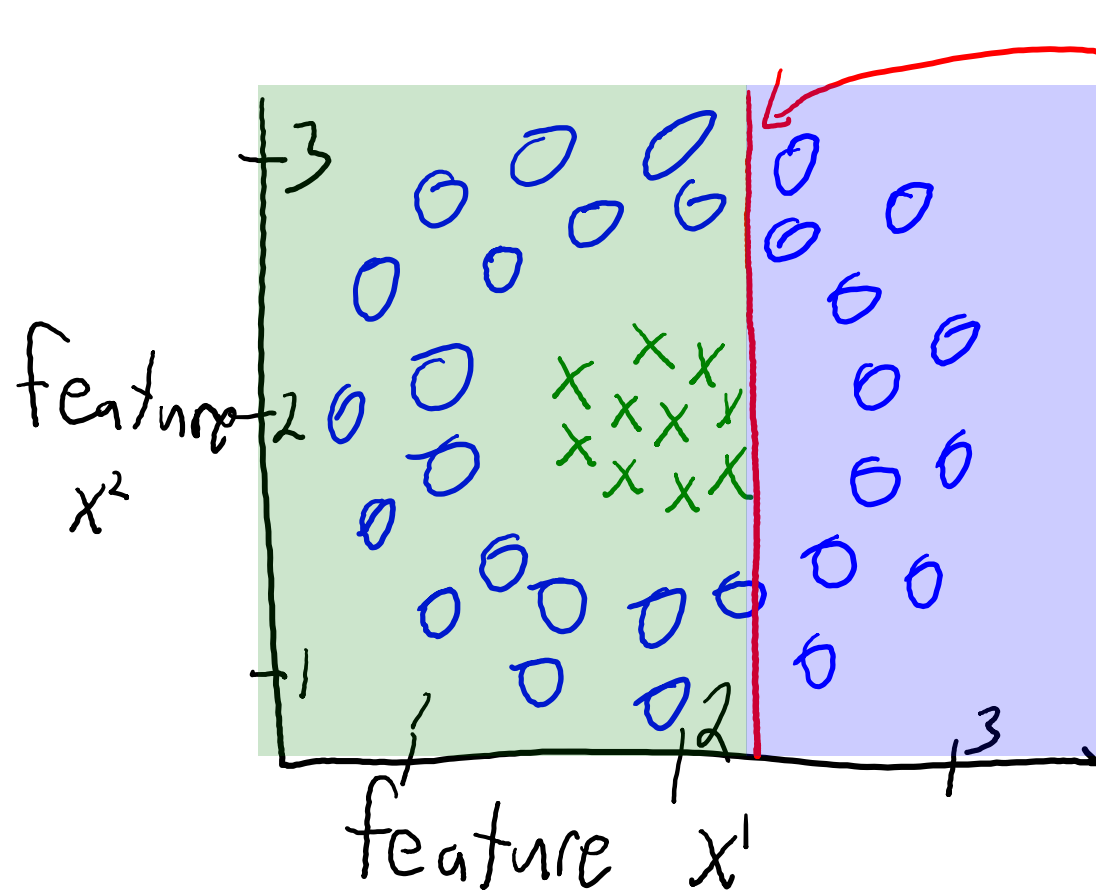
This split makes labels less random. (Everything on the right is a '0')

It did not improve accuracy.

feature x^2

feature x^1

Example Where Accuracy Fails

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

It did not improve accuracy.

But three more splits maximizing infogain lead to perfect accuracy.

feature x^2

feature x^1

Discussion of Decision Tree Learning

- Advantages:
  - Easy to implement.
  - Interpretable.
  - Learning is fast. Prediction is very fast.
- Disadvantages:
  - Hard to find optimal set of rules.
  - Greedy splitting often not accurate, requires very deep trees

Discussion of Decision Tree Learning

- 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 scoring function?
    - Infogain is the most popular and often works well, but is not always the best.
  - What if you get new data?
    - 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?
    - Some implementations stop at a maximum depth, some stop if too few
      examples in leaf, some stop if infogain is too small.

Summary

- **Exploring data**:
  - Data visualization
- **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.

Review Questions

- Q1: Is "return-the-mode" always a bad idea? Why?

- Q2: What is the output of a supervised learning algorithm?

- Q3: What are the 4 parameters of a decision stump?

- Q4: In course notation, what are n, d, i, and j?

- Q5: Why is finding an optimal decision tree difficult?

- Q6: How many thresholds do we need to look at for a decision stump?

You can do A1 Q5-Q6 now!

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

Entropy Function

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

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
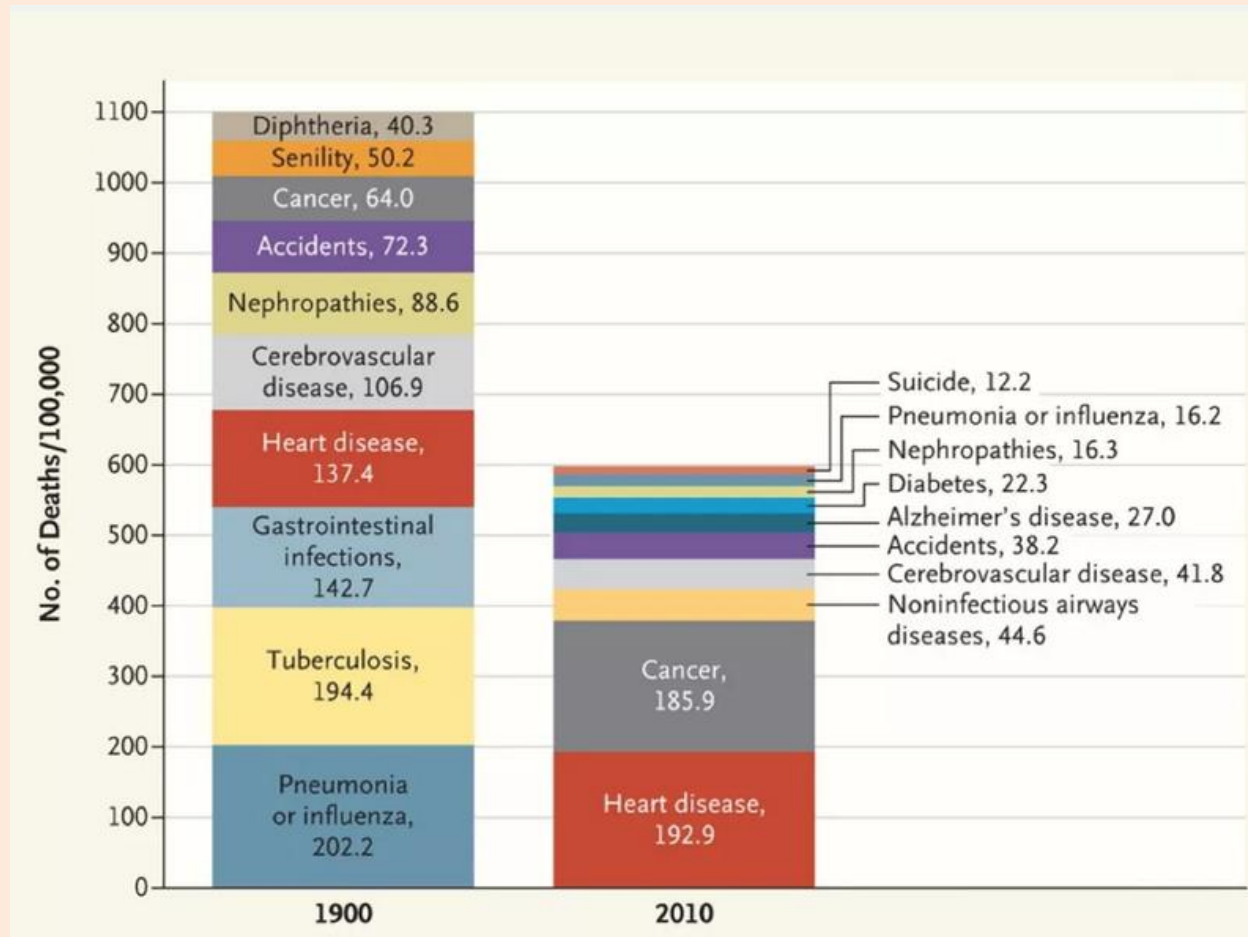
Histogram

- **"[Four Basic Data Science Lessons Illustrated by COVID-19 Data](#)"**
  - First two lessons come from just plotting:
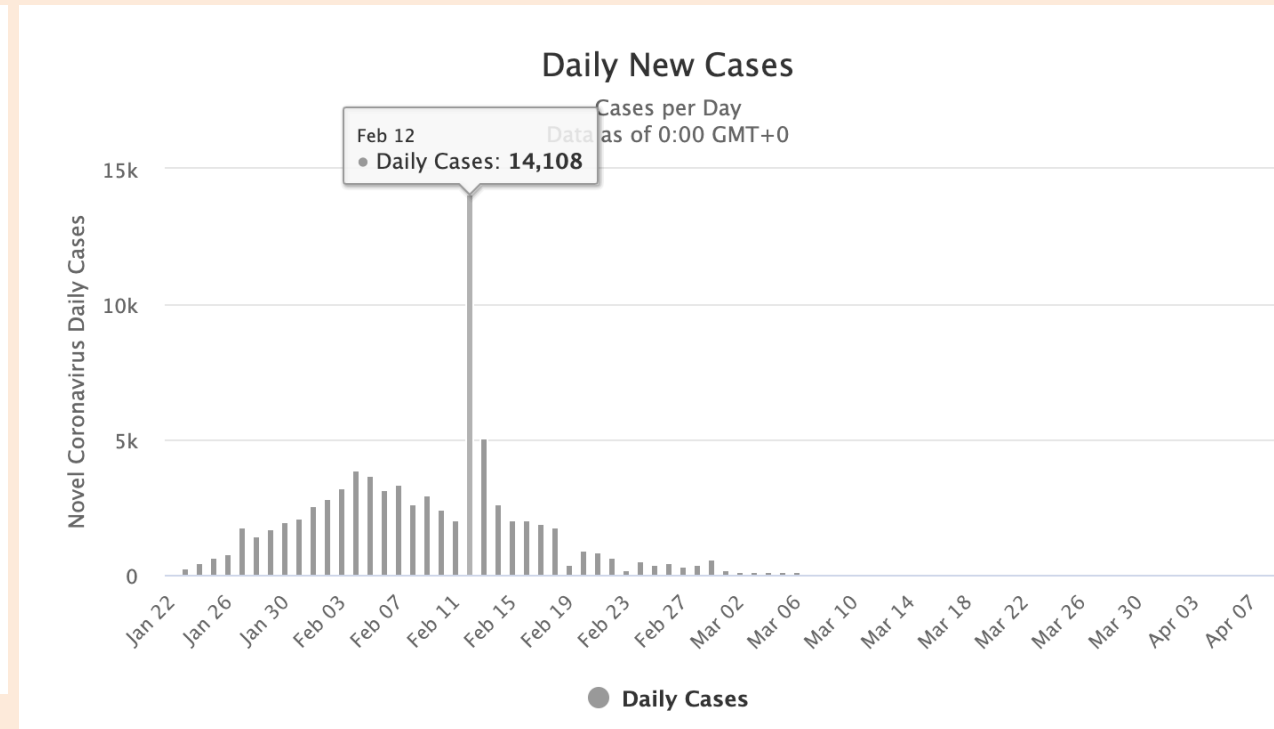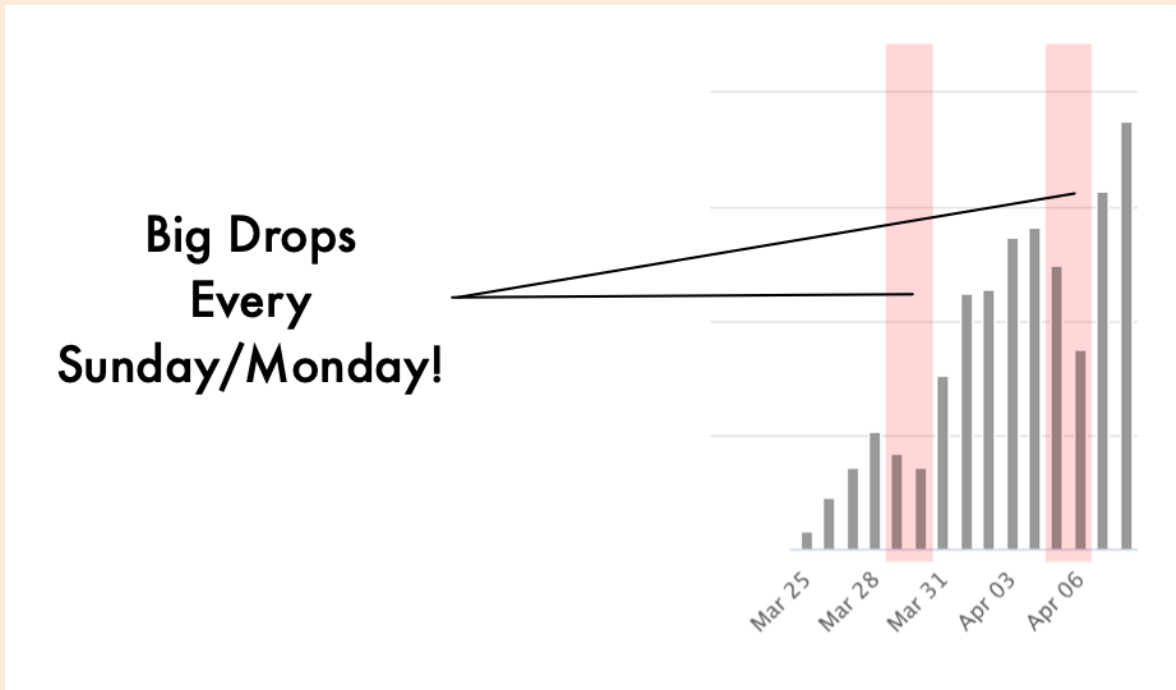
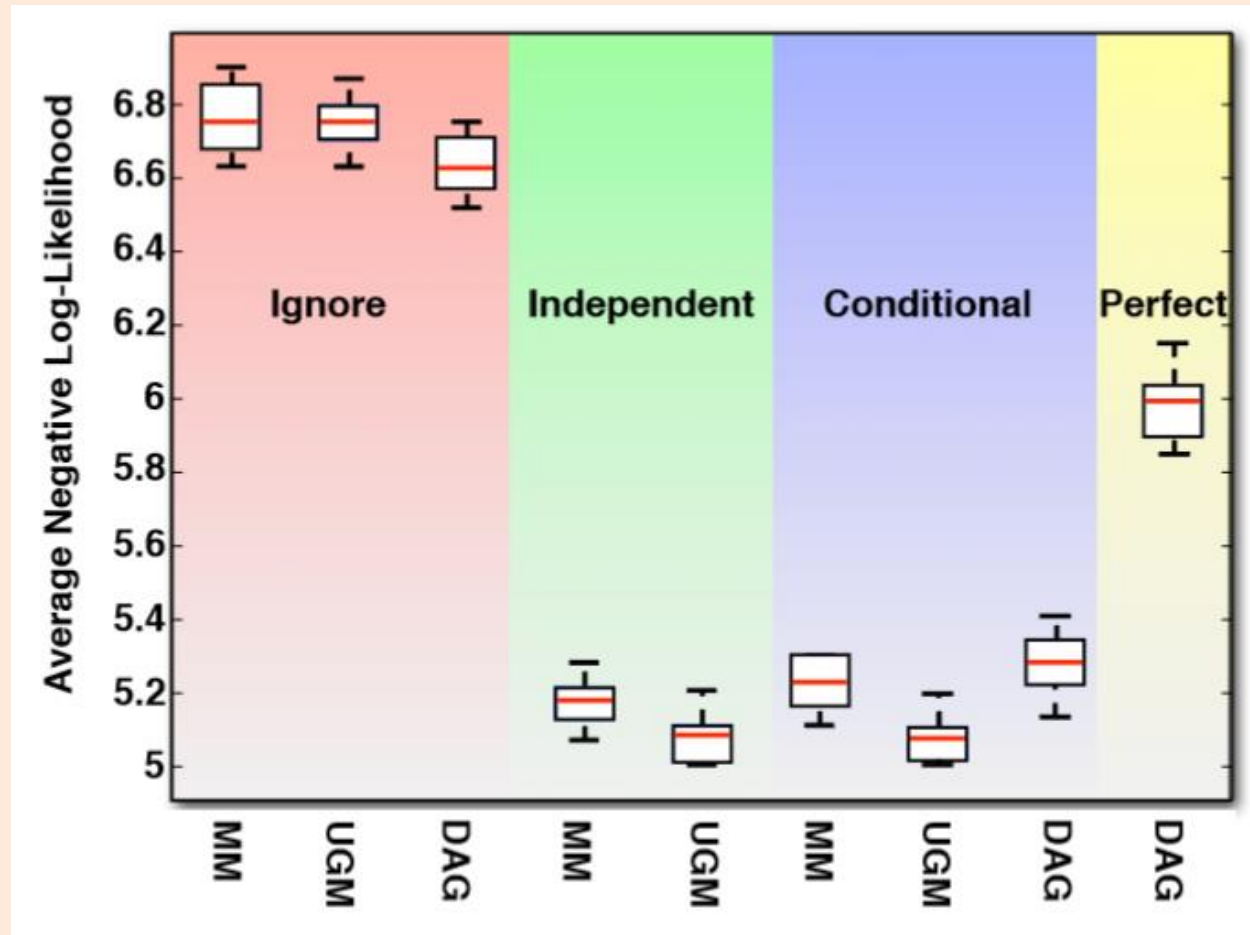 - These oddities had to do with how data was recorded.
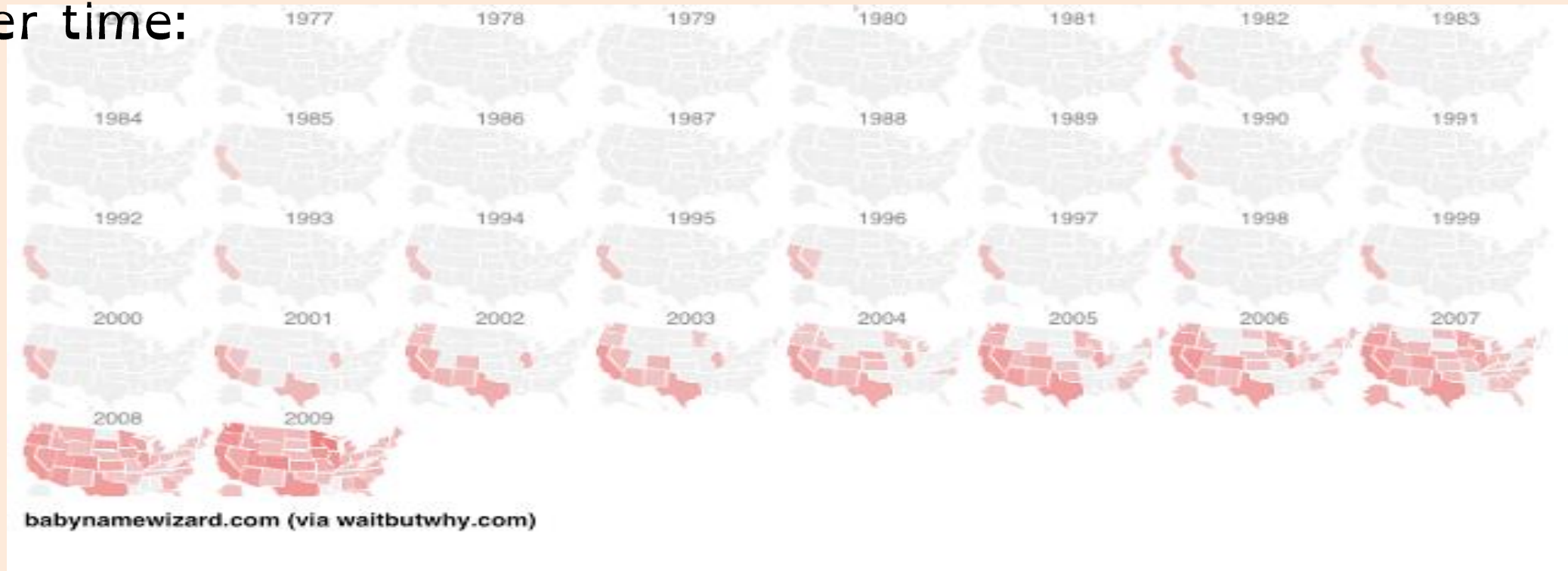
Histogram

- Histogram with grouping:

http://www.vox.com/2016/5/10/11608064/americans-

Box Plots

- Box plot with grouping:

Map Coloring

- Color/intensity can represent feature of region.
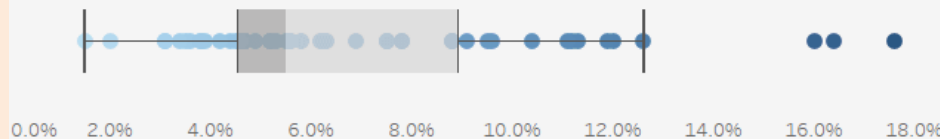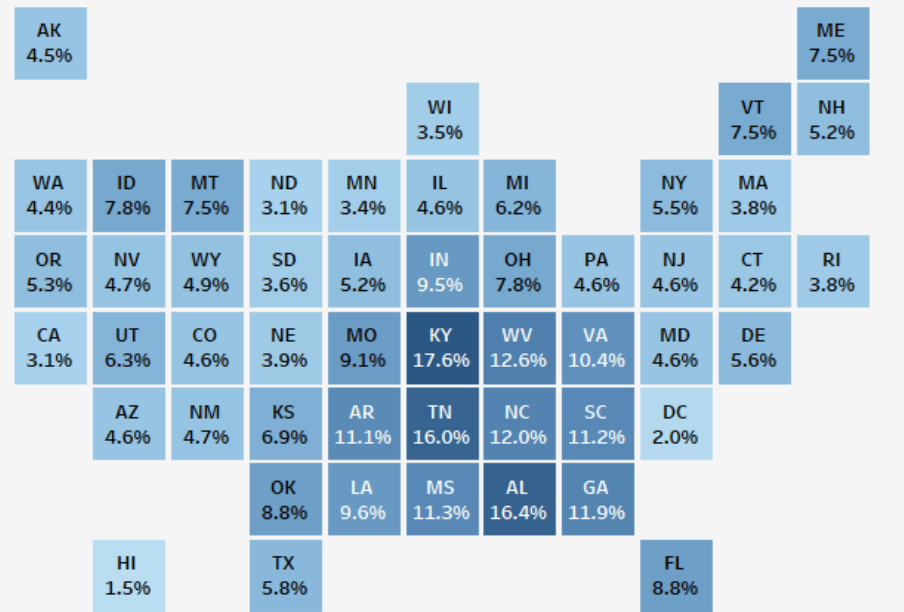
Popularity of naming baby "Evelyn"
over time:

baby namewizard.com (via waitbutwhy.com)

But not very good if some regions are very small.

http://waitbutwhy.com/2013/12/how-to-name-baby.html
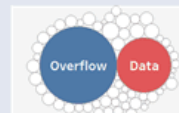
Canadian Income Mobility

Map Coloring

- Variation just uses fixed-size blocks and tries to arrange geog...

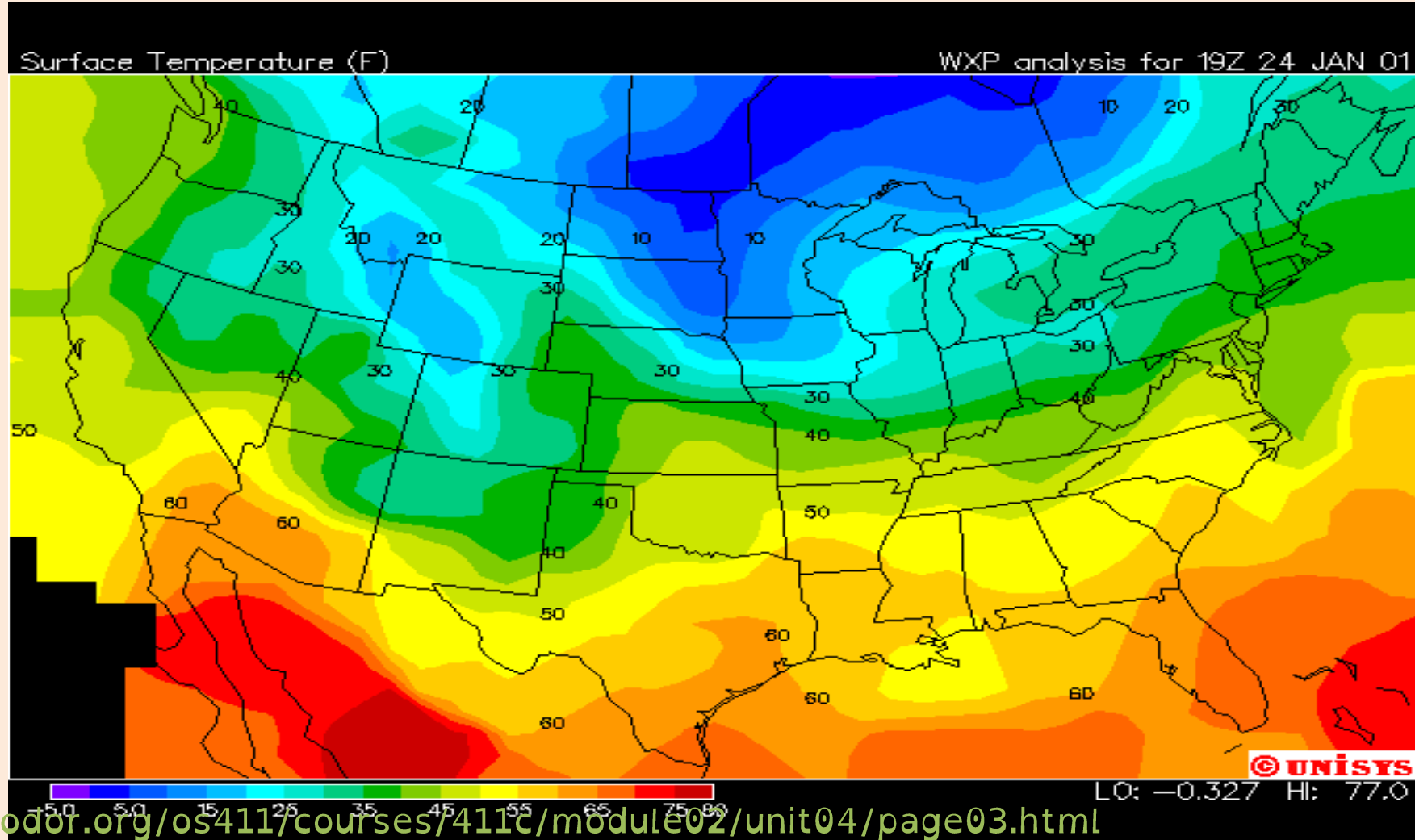What % of the population claims American ancestry in each state?

Source: U.S. Census Bureau, 2015 ACS 1 Year Estimates
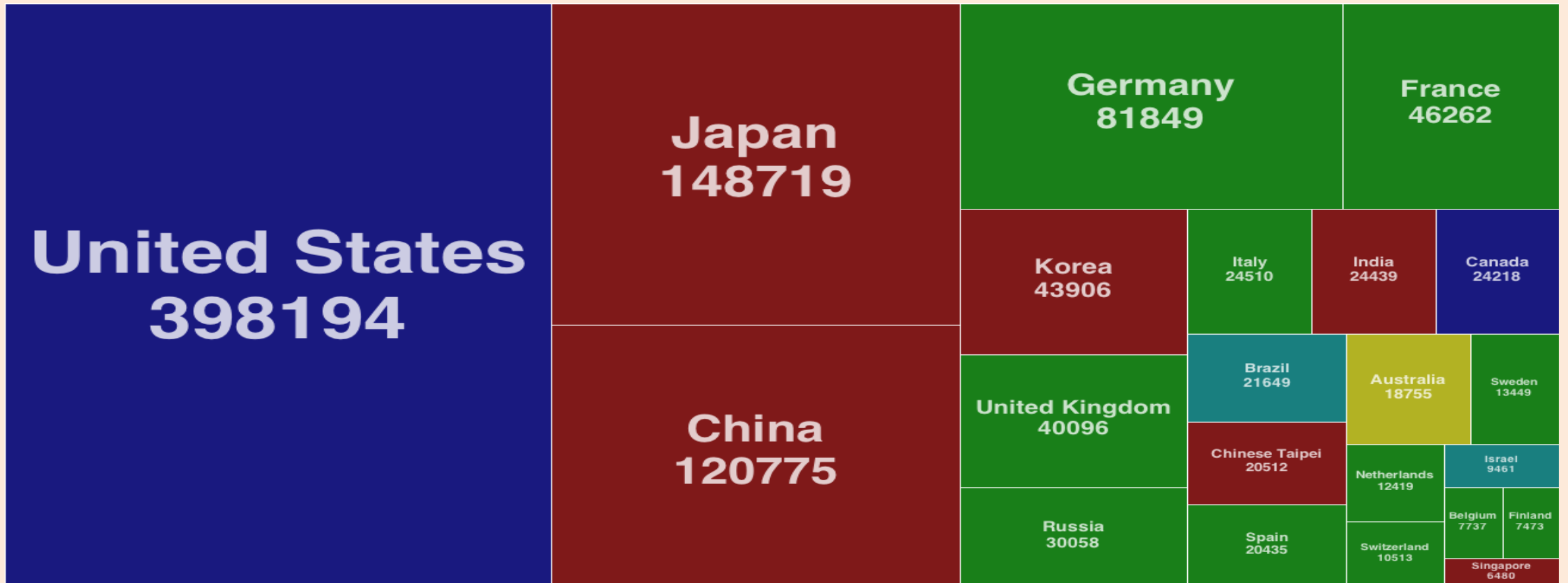Click the logo to see more data visualizations

http://www.visualcapitalist.com/america-immigration-1820/

Contour Plot

- Colour visualizes 'z' as we vary 'x' and 'y'.

Surface Temperature (F) — WXP analysis for 19Z 24 JAN 01

http://www.shodor.org/os411/courses/411c/module02/unit04/page03.html

Treemaps

- Area represents attribute value:

http://mjwest.org/vis/treemaps/

Cartogram

- Fancier version of treemaps:

http://www-personal.umich.edu/~mejn/cartograms/

Stream Graph

Baby Name >   ⦿ Both ⦾ Boys ⦾ Girls

Current rank: boys [1000] [500] [100] [25] [1]
girls [1000] [500] [100] [25] [1]

This is every name in the Top 1,000 for each decade since 1880, all stacked on top of each other.

In 1950, 95% of all babies were given a Top 1,000 name.

In 2012, only 73% of baby names fell into the Top 1,000.

900,000
800,000
700,000
600,000
500,000
400,000
300,000
200,000
100,000

1880s   1890s   1900s   1910s   1920s   1930s   1940s   1950s   1960s   1970s   1980s   1990s   2000s   2012

babynamewizard.com (via waitbutwhy.com)

http://waitbutwhy.com/2013/12/how-to-name-baby.html

Stream Graph

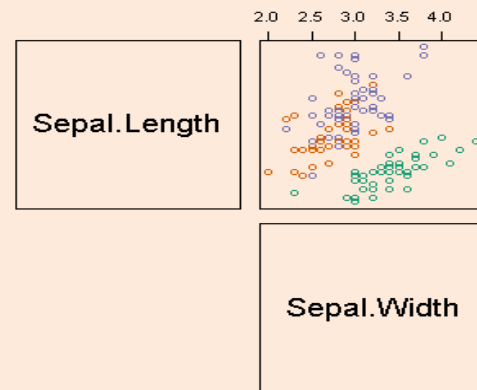http://www.babynamewizard.com/d3js-

Stream Graph

Videos and Interactive Visualizations

- For data recorded over time, videos can be useful:
  - [Map colouring over time](.).

- There are also lots of neat interactive visualization methods:
  - [Sale date for most expensive paintings](.).
  - [Global map of wind, weather, and oceans](.).
  - [Many examples here](.).

Scatterplot Arrays

- For multiple variables, can use scatterplot array.

- Colors can indicate a third categorical variable.

https://en.wikipedia.org/wiki/Iris_flower_data_set
http://www.ats.ucla.edu/stat/r/pages/layout.htm

Scatterplot Arrays

- For multiple variables, can use scatterplot array.

Fisher's *Iris* Data [hide]

Sepal length	Sepal width	Petal length	Petal width	Species
5.0	2.0	3.5	1.0	I. versicolor
6.0	2.2	4.0	1.0	I. versicolor
6.2	2.2	4.5	1.5	I. versicolor
6.0	2.2	5.0	1.5	I. virginica
4.5	2.3	1.3	0.3	I. setosa
5.0	2.3	3.3	1.0	I. versicolor
5.5	2.3	4.0	1.3	I. versicolor
6.3	2.3	4.4	1.3	I. versicolor
4.9	2.4	3.3	1.0	I. versicolor
5.5	2.4	3.7	1.0	I. versicolor
5.5	2.4	3.8	1.1	I. versicolor
5.1	2.5	3.0	1.1	I. versicolor

- Colors can indicate a third categorical variable.

Scatterplot Arrays

- For multiple variables, can use scatterplot array.

- Colors can indicate a third categorical variable.

"Why Not to Trust Plots"

- We've seen how summary statistics can be mis-leading.
- Note that plots can also be mis-leading, or can be used to mis-lead.

- Next slide: first example from UW's excellent course:
  - "Calling Bullshit in the Age of Big Data".
    - A course on how to recognize when people are trying to mis-lead you with data.
  - I recommend watching all the videos here:
    - https://www.youtube.com/watch?v=A2OtU5vlR0k&list=PLPnZfvKID1Sje5jWxt-4CSZD7bUl4gSPS

  - Recognizing BS not only useful for data analysis, but for daily life.

Mis-Leading Axes

- This plot seems to show amazing recent growth:

Explosive growth at callingbullshit.org
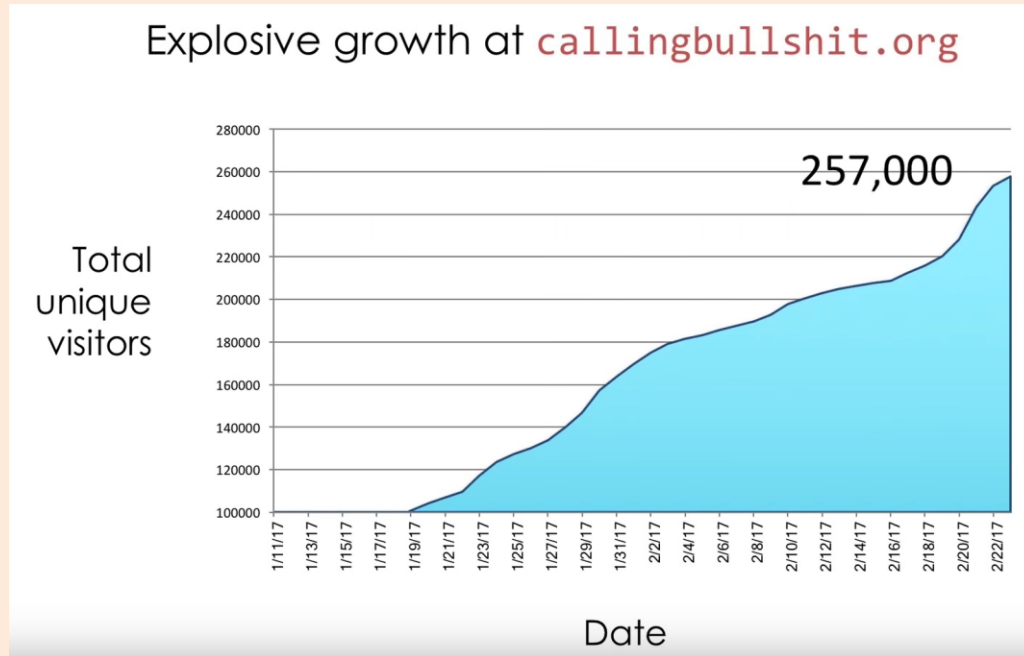
- But notice y-axis starts at 100,000 (so ~40% of growth was earlier).
- And it plots "total" users (which necessarily goes up).

https://www.youtube.com/watch?v=q94VJ3KToK8

Mis-Leading Axes

- Plot of actual daily users (starting from 0) looks totally different:

Explosive growth at callingbullshit.org ?

Cumulative iPhone sales
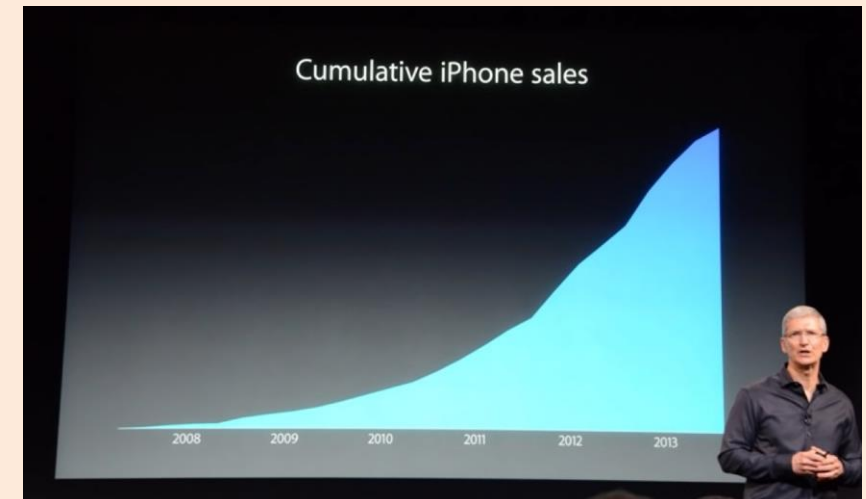
- People can mis-lead to push agendas/stories:

https://www.youtube.com/watch?v=q94VJ3KToK8

Mis-Leading Axes

- Watch out for the starting point of the axes too:

https://www.boredpanda.com/average-women-height-data-chart-latvian-
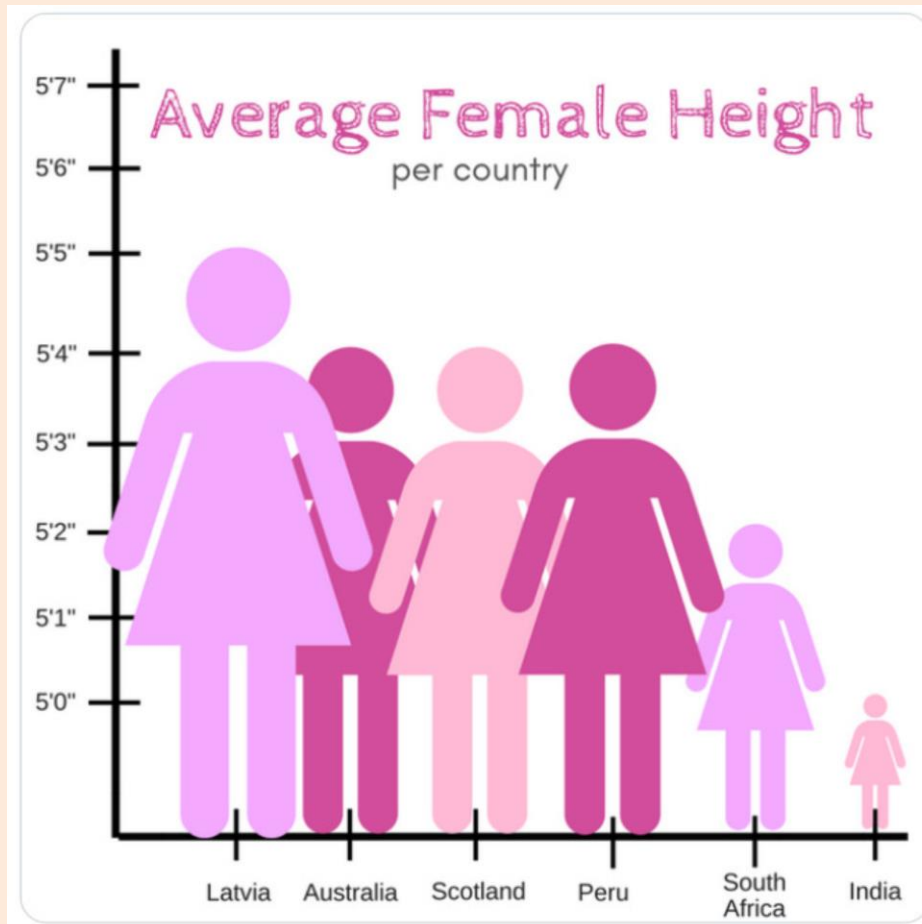
Mis-Leading Axes

- We see "lack of appropriate axes" ALL THE TIME in the news:
  - "British research revealed that patients taking ibuprofen to treat arthritis face a 24% increased risk of suffering a heart attack"
    - What is probability of heart attack if I you don't take it? Is that big or small?
    - Actual numbers: less than 1 in 1000 "extra" heart attacks vs. baseline frequency.
      - There is a risk, but "24%" is an exaggeration.
    - "Health-scare stories often arise because their authors simply don't understand numbers."
      - Or it could be that they do understand, but media wants to "sensationalize" mundane news.

  - Bonus slides: more "Calling Bullshit" course examples on "political" issues:
    - Global warming, vaccines, gun violence, taxes.

More Mis-Leading Axes from "Calling Bullshit"

https://www.youtube.com/watch?v=9pNWVMxaFuM
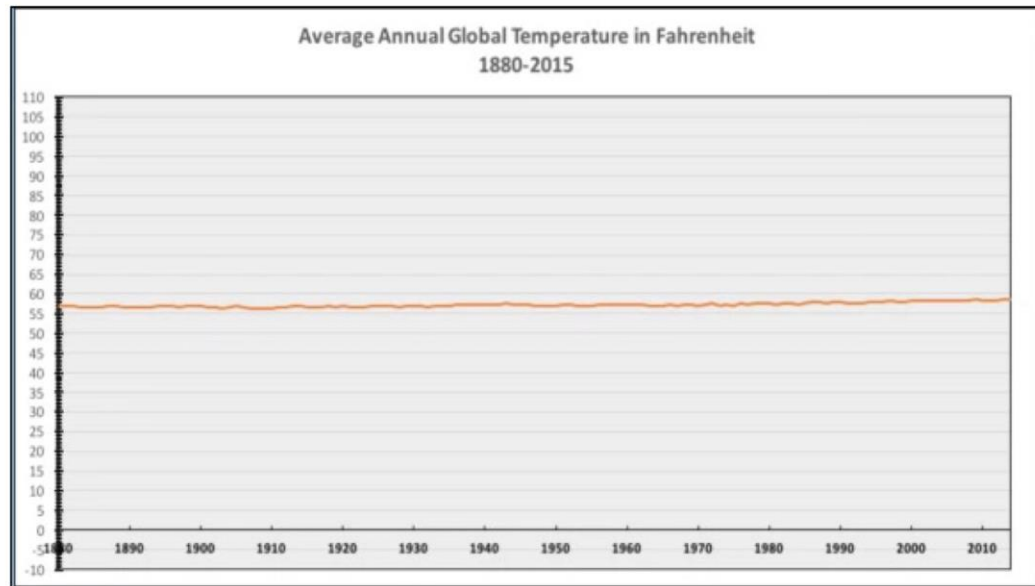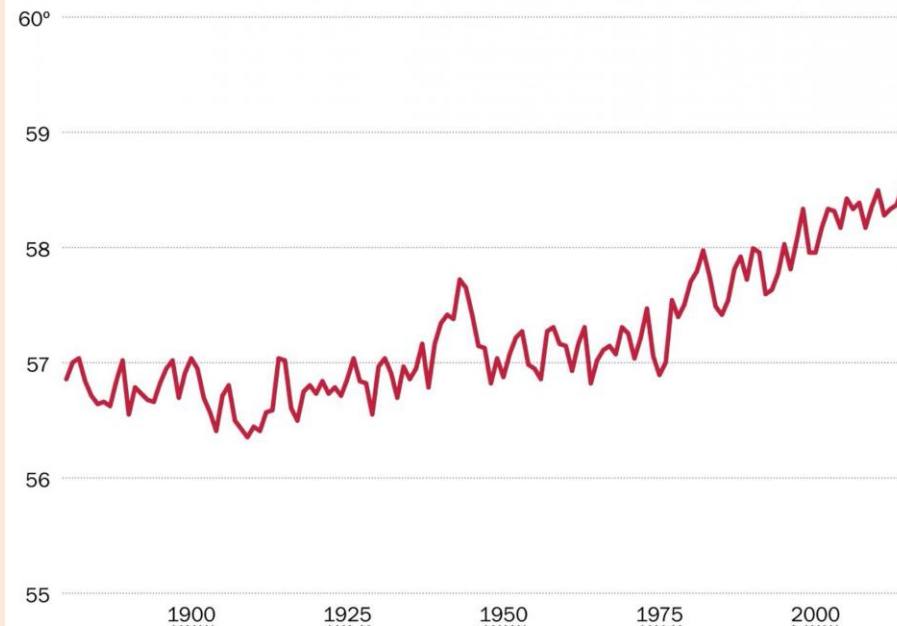
More Mis-Leading Axes from "Calling Bullshit"
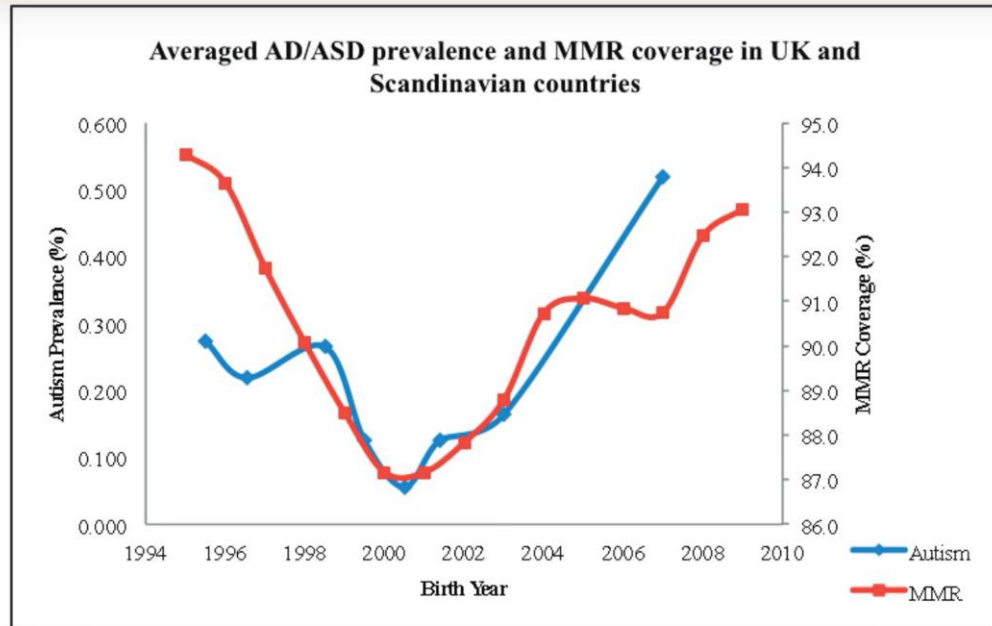
Figure 1-Averaged AD/ASD prevalence and MMR coverage in UK, Norway and Sweden. Both MMR and AD/ASD data are normalized to the maximum coverage/prevalence during the time period of this analysis.

Diesher *et al.* 2015 *Issues in Law and Medicine*
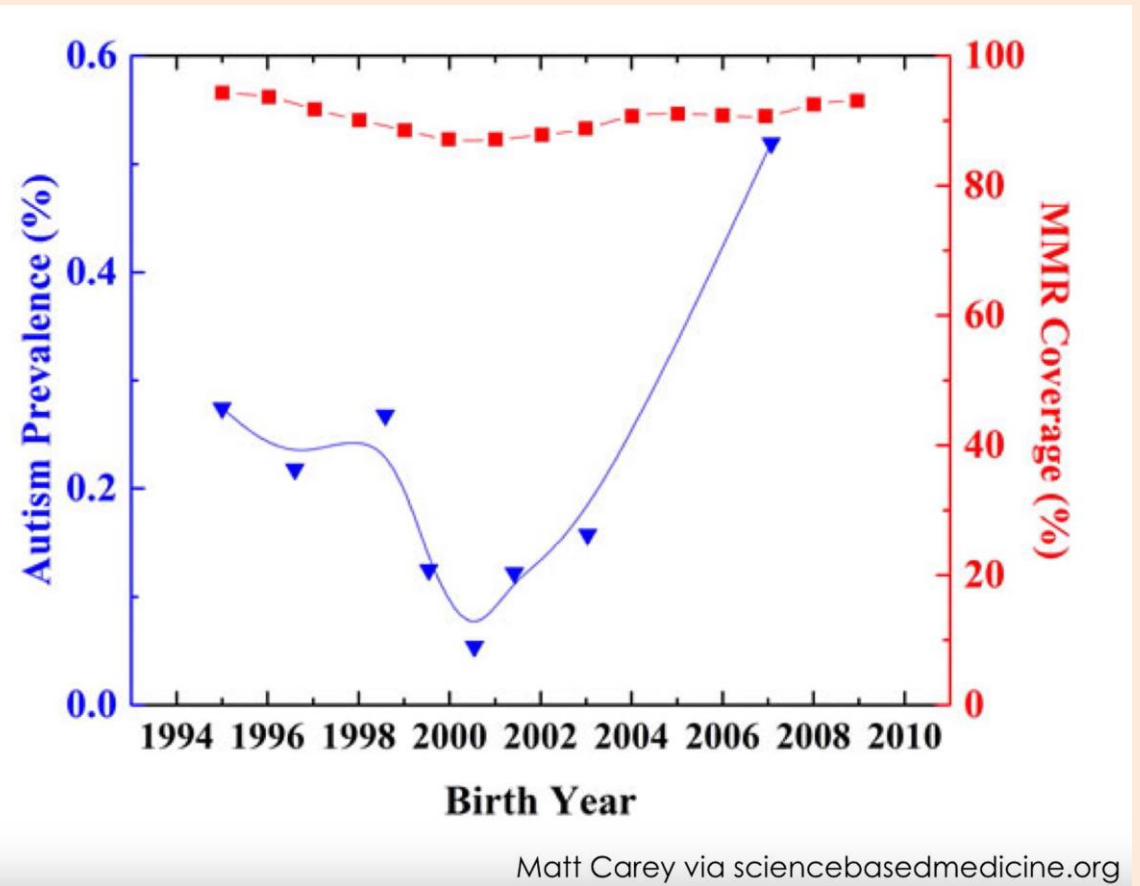
Matt Carey via sciencebasedmedicine.org

https://www.youtube.com/watch?v=9pNWVMxaFuM

More Mis-Leading Axes from "Calling Bullshit"

The Middle Class Tax Target

The amount of total taxable income (left scale) for all filers by adjusted gross income level for 2008
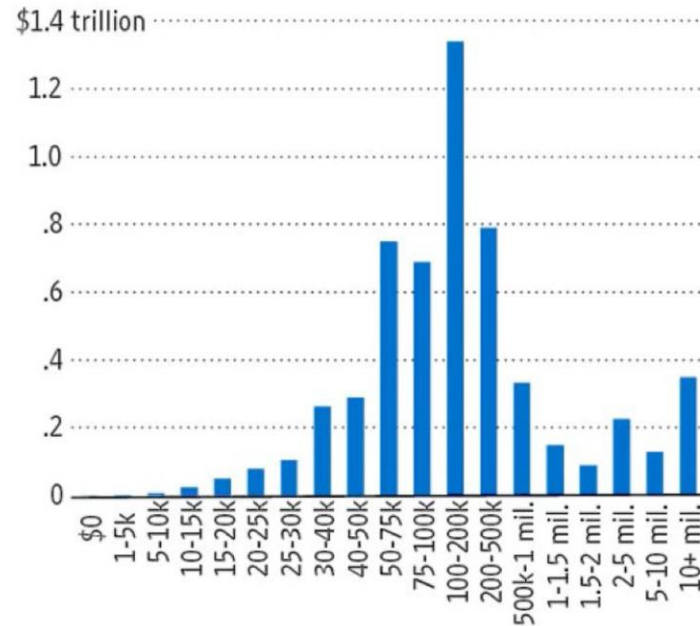
"The rich, in short, aren't nearly rich enough to finance Mr. Obama's entitlement state ambitions—even before his health-care plan kicks in.

So who else is there to tax? Well, in 2008, there was about $5.65 trillion in total taxable income from all individual taxpayers, and most of that came from middle income earners. The nearby chart shows the distribution, and the big hump in the center is where Democrats are inevitably headed for the same reason that Willie Sutton robbed banks."

Source: IRS

-The Wall Street Journal
April 17, 2011

- Look at the histogram bin widths.

More Mis-Leading Axes from "Calling Bullshit"

- **Axis is upside down.**
- Looks like law makes murder go down, but number of murders go up!
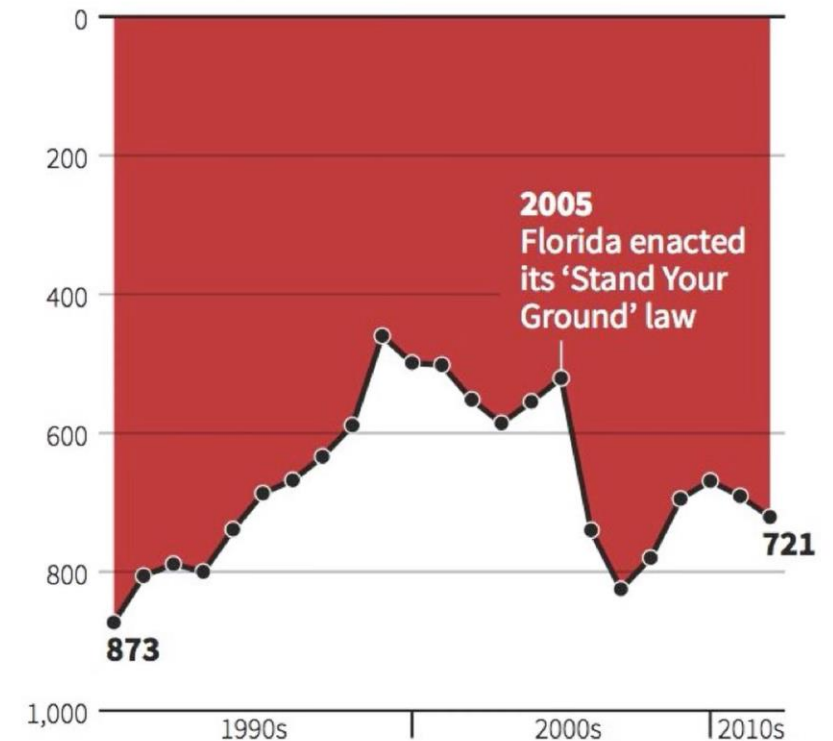
More Mis-Leading Axes from "Calling Bullshit"

- Calling BS gives this as another example:

- **Actual numbers don't say much of anything:**

  > Key findings of the survey include:
  >
  > -- 39% of responding institutions reported a decline in international applications, 35% reported an increase, and 26% reported no change in applicant numbers.

- 39% vs. 35% (without sizes) doesn't mean "down nearly 40 percent".
  - Data can be used in mis-leading ways to "push agendas".
  - Even by reputed sources.
  - Even if you agree with the message.

https://www.youtube.com/watch?v=-Mtmi8smpfo

More Mis-Leading Axes from Vancouver Housing

- A local example:
  - Are almost all Vancouver homes becoming empty?

  - Or did they use different y-axis scales for each line?

Total Private Dwellings and Unoccupied or Occupied by Temporary Resident

https://openhousing.ca/2020/12/30/vancouver-empty-homes-crossed-
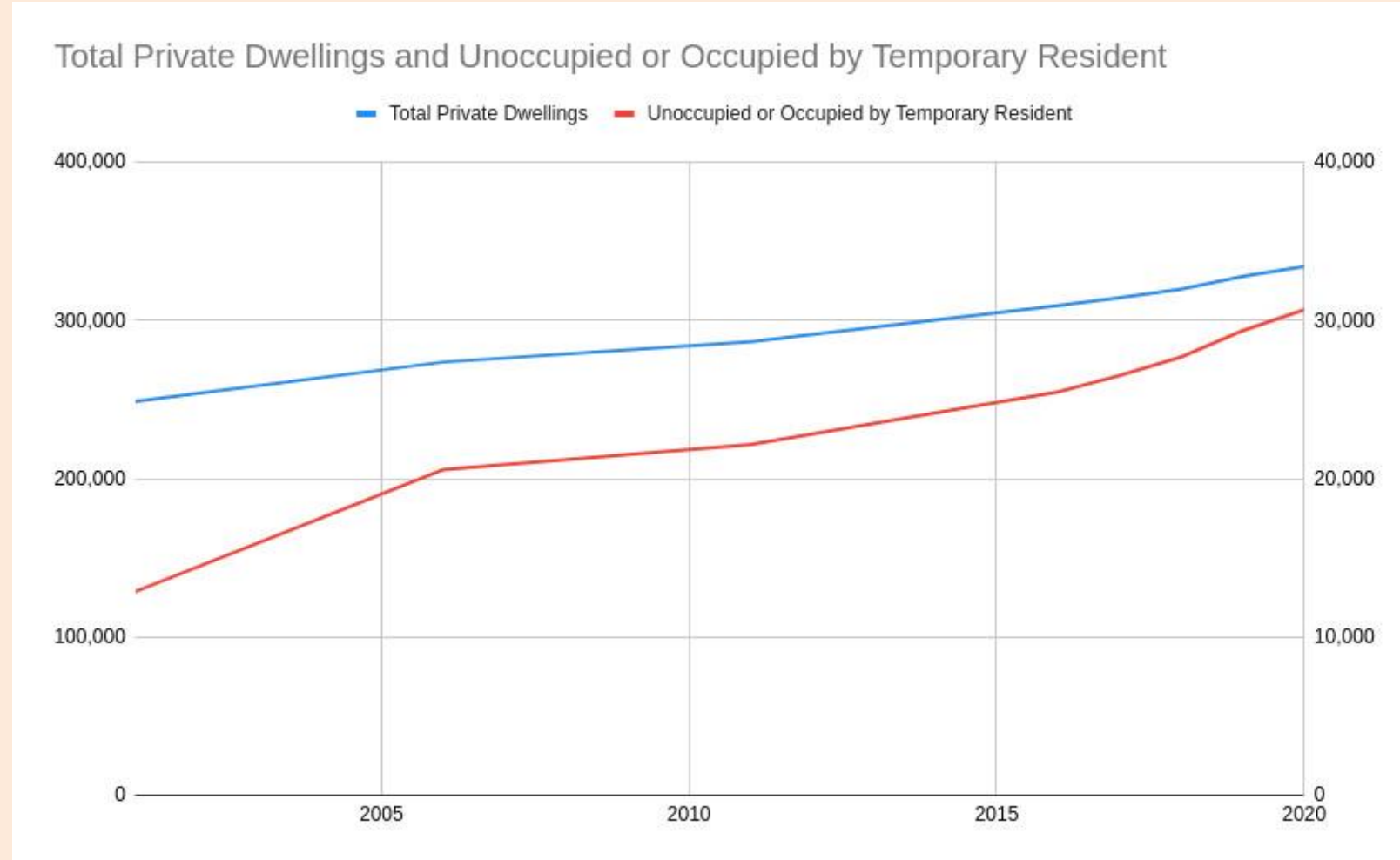
Hamming Distance vs. Jaccard Coefficient

A	B
1	0
1	0
1	0
0	1
0	1
1	0
0	0
0	0
0	1

- These vectors agree in 2 positions.
  - Normalizing Hamming distance by vector length, similarity is 2/9.

- If we're really interested in predicting 1s, we could find set of 1s in both and compute Jaccard:
  - A -> {1,2,3,6}, B -> {4,5,9}
  - No intersection so Jaccard similarity is actually 0.

Hamming Distance vs. Jaccard Coefficient
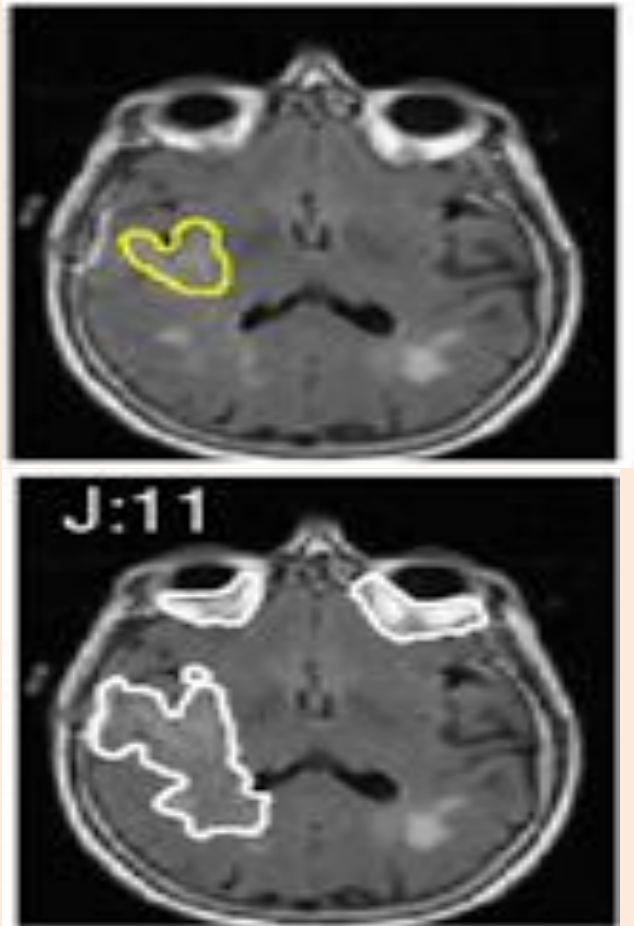
- Let's say we want to find the tumour in an MR image.
- We have an expert label (top) and a prediction from our ML system (bottom).

- The normalized Hamming distance between the predictions at each pixel is 0.91. This sounds good, but since there are so many non-tumour pixels this is misleading.
- The ML system predicts a much bigger tumour so hasn't done well. The Jaccard coefficient between the two sets of tumour pixels is only 0.11 so reflects this.

Coupon Collecting

- Consider trying to collect 50 uniformly-distributed states, drawing at random.
- The probability of getting a new state if there 'x' states left: p=x/50.
- So expected number of samples before next "success" (getting a new state) is 50/x.
   (mean of geometric random variable with p=x/50)
- So the expected number of draws is the sum of 50/x for x=1:50.
- For 'n' states instead of 50, summing until you have all 'n' gives:

$$\sum_{i=1}^{n} \frac{n}{i} = n \sum_{i=1}^{n} \frac{1}{i} \leq n\left(1 + \log(n)\right) = O(n \log n)$$

Huge Datasets and Parallel/Distributed Computation

- Most sufficient statistics can be computed in linear time.
- For example, the mean of 'n' numbers is computed as:

$$\text{mean}(x_1, x_2, x_3, \ldots, x_n) = \frac{x_1 + x_2 + x_3 + \cdots + x_n}{n}$$

- This costs O(n), which is great.

- But if 'n' is really big, we can go even faster with parallel computing...

Huge Datasets and Parallel/Distributed Computation

- Computing the mean with multiple cores:
  - Each of the 'c' cores computes the sum of O(n/c) of the data:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n/c} \end{bmatrix} \longrightarrow \text{sum1}$$

$$\rightarrow \begin{bmatrix} x_{n/c+1} \\ x_{n/c+2} \\ \vdots \\ x_{2n/c} \end{bmatrix} \longrightarrow \text{sum2}$$

$$\rightarrow \begin{bmatrix} x_{(c-1)n/c+1} \\ x_{(c-1)n/c+2} \\ \vdots \\ x_n \end{bmatrix} \longrightarrow \text{sumc}$$

Huge Datasets and Parallel/Distributed Computation

- **Computing the mean with** multiple cores**:**
  - Each of the 'c' cores computes the sum of $O(n/c)$ of the data:
  - Add up the 'c' results from each core to get the mean.

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n/c} \end{bmatrix} \rightarrow sum1$$

$$\begin{bmatrix} x_{n/c+1} \\ x_{n/c+2} \\ \vdots \\ x_{2n/c} \end{bmatrix} \rightarrow sum2$$

$$mean(x) = \frac{sum1 + sum2 + \cdots sumc}{n}$$

$$\begin{bmatrix} x_{(c-1)n/c+1} \\ x_{(c-1)n/c+2} \\ \vdots \\ x_n \end{bmatrix} \rightarrow sumc$$
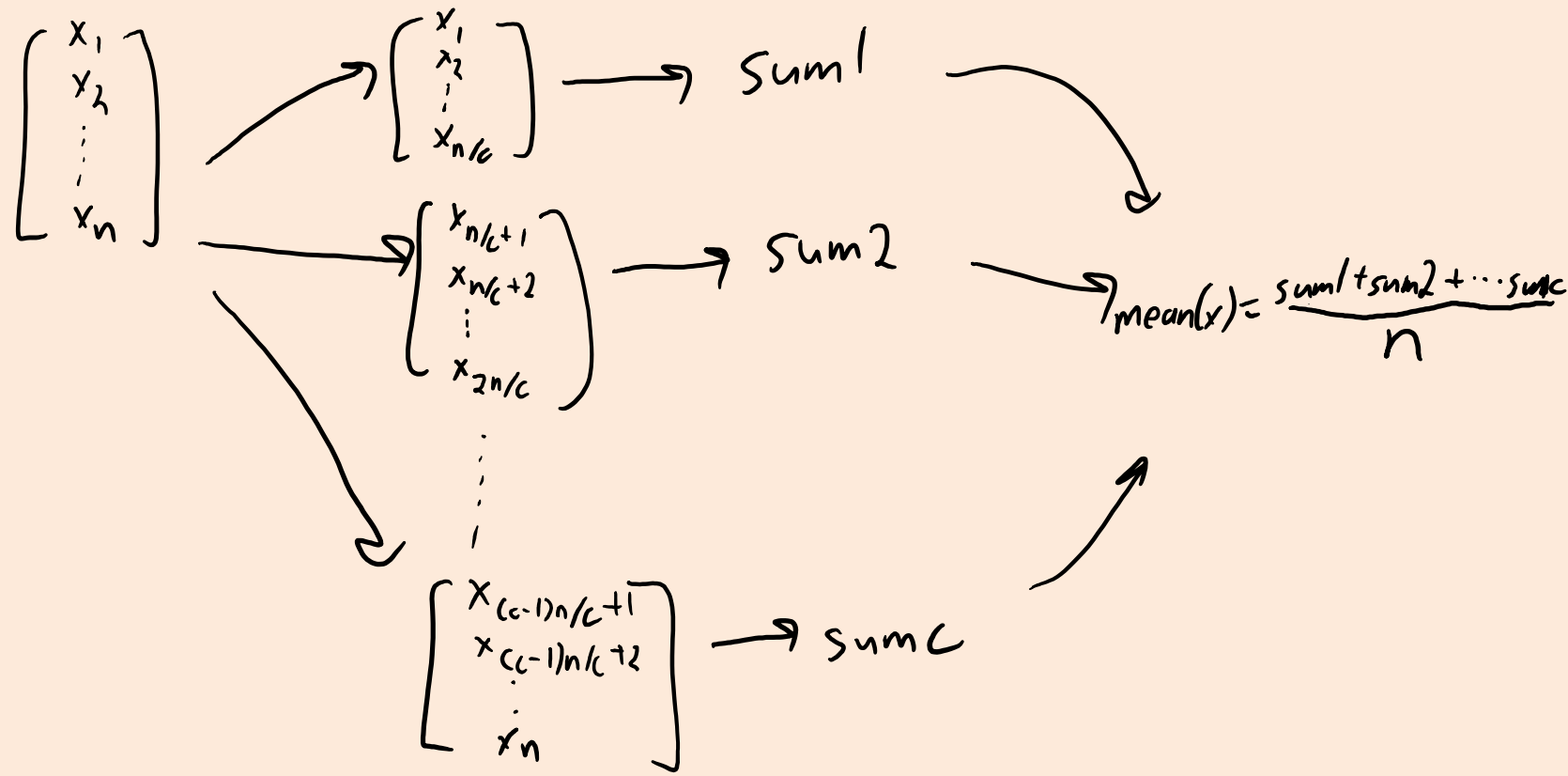
Huge Datasets and Parallel/Distributed Computation

- Computing the mean with multiple cores:
  - Each of the 'c' cores computes the sum of $O(n/c)$ of the data.
  - Add up the 'c' results from each core to get the mean.
  - Cost is only $O(n/c + c)$, which can be much faster for large 'n'.

- This assumes cores can access data in parallel (not always true).
- Can reduce cost to $O(n/c)$ by having cores write to same register.
  - But need to "lock" the register and might effectively cost $O(n)$.

Huge Datasets and Parallel/Distributed Computation

- Sometimes 'n' is so big that data can't fit on one computer.

- In this case the data might be distributed across 'c' machines:
  - Hopefully, each machine has $O(n/c)$ of the data.

- We can solve the problem similar to the multi-core case:
  - "Map" step: each machine computes the sum of its data.
  - "Reduce" step: each machine communicates sum to a "master" computer, which adds them together and divides by 'n'.

Huge Datasets and Parallel/Distributed Computation

- **Many problems in DM and ML** have this flavour:
  - "Map" computes an operation on the data on each machine (in parallel).
  - "Reduce" combines the results across machines.

Input Data → map() / map() / map() → reduce() / reduce() → Output Data

Split [k1, v1]   Sort by k1   Merge [k1, [v1, v2, v3 …]]
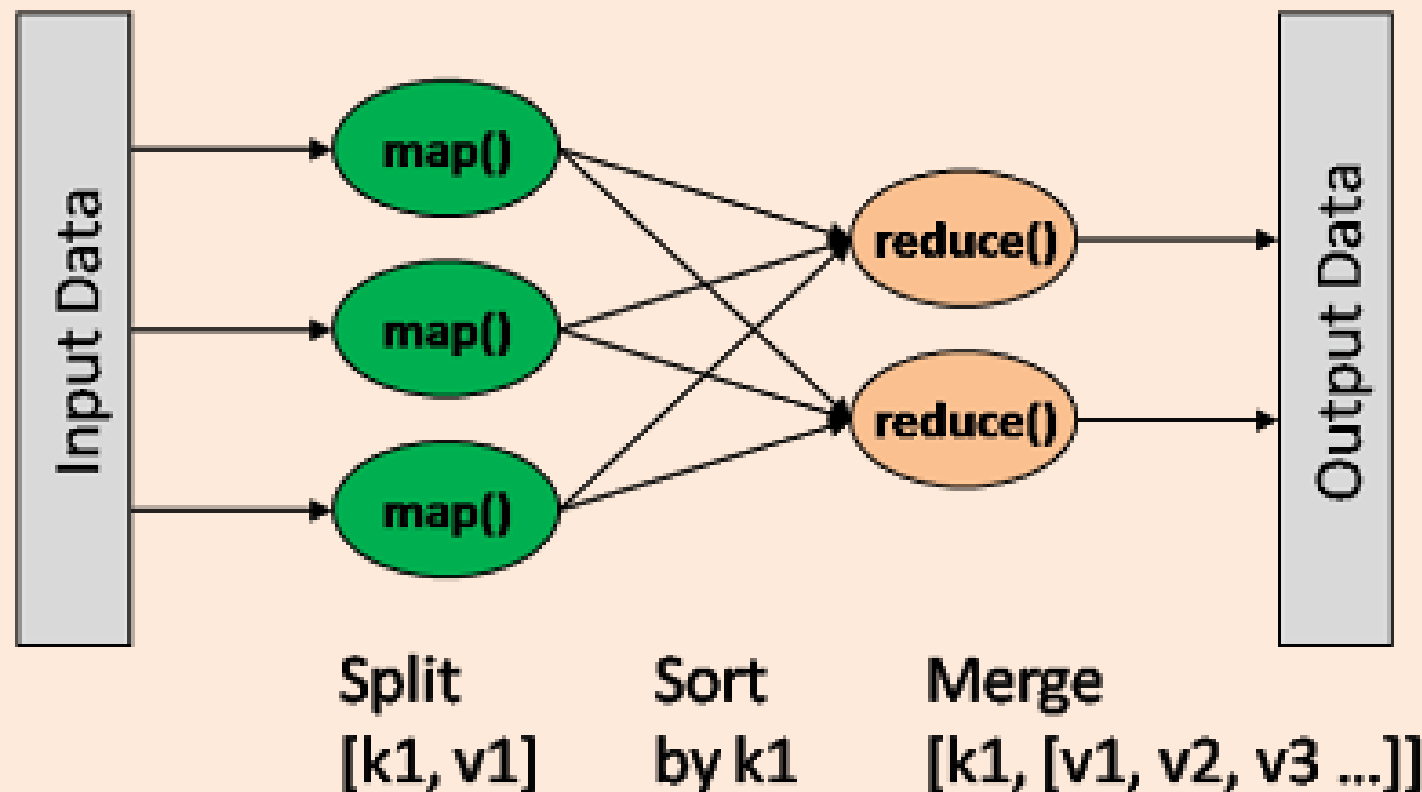
Huge Datasets and Parallel/Distributed Computation

- Many problems in DM and ML have this flavour:
  - "Map" computes an operation on the data on each machine (in parallel).
  - "Reduce" combines the results across machines.
  - These are standard operations in parallel libraries like MPI.

- Can solve many problems almost 'c' times faster with 'c' computers.

- To make it up for the high cost communicating across machines:
  - Assumes that most of the computation is in the "map" step.
  - Often need to assume data is already on the computers at the start.

Huge Datasets and Parallel/Distributed Computation

- Another challenge with "Google-sized" datasets:
  - You may need so many computers to store the data,
    that it's inevitable that some computers are going to fail.
- Solution to this is a distributed file system.

- Two popular examples are Google's MapReduce and Hadoop DFS:
  - Store data with redundancy (same data is stored in many places).
    - And assume data isn't changing too quickly.
  - Have a strategy for restarting "map" operations on computers that fail.
  - Allows fast calculation of more-fancy things than sufficient statistics:
    - Database queries and matrix multiplications.

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

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

Mil k
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

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)?

Mil k
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).

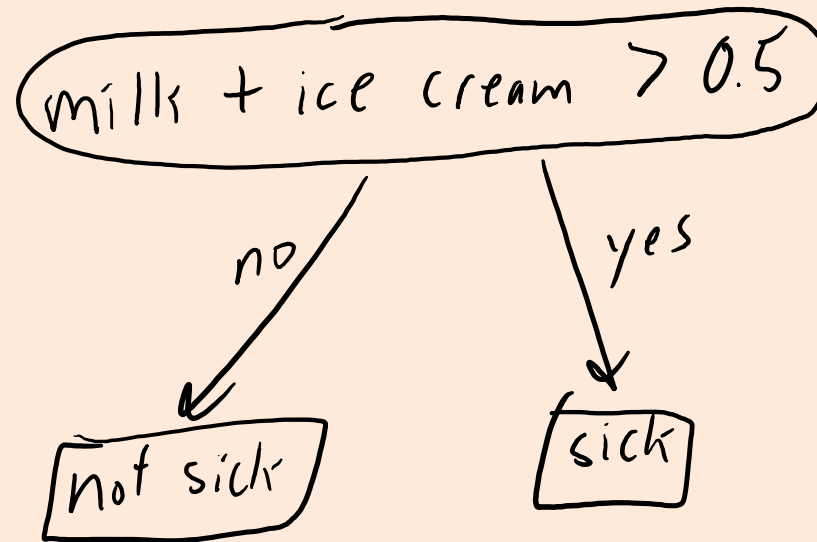# Can decision trees re-visit a feature?

- Yes.
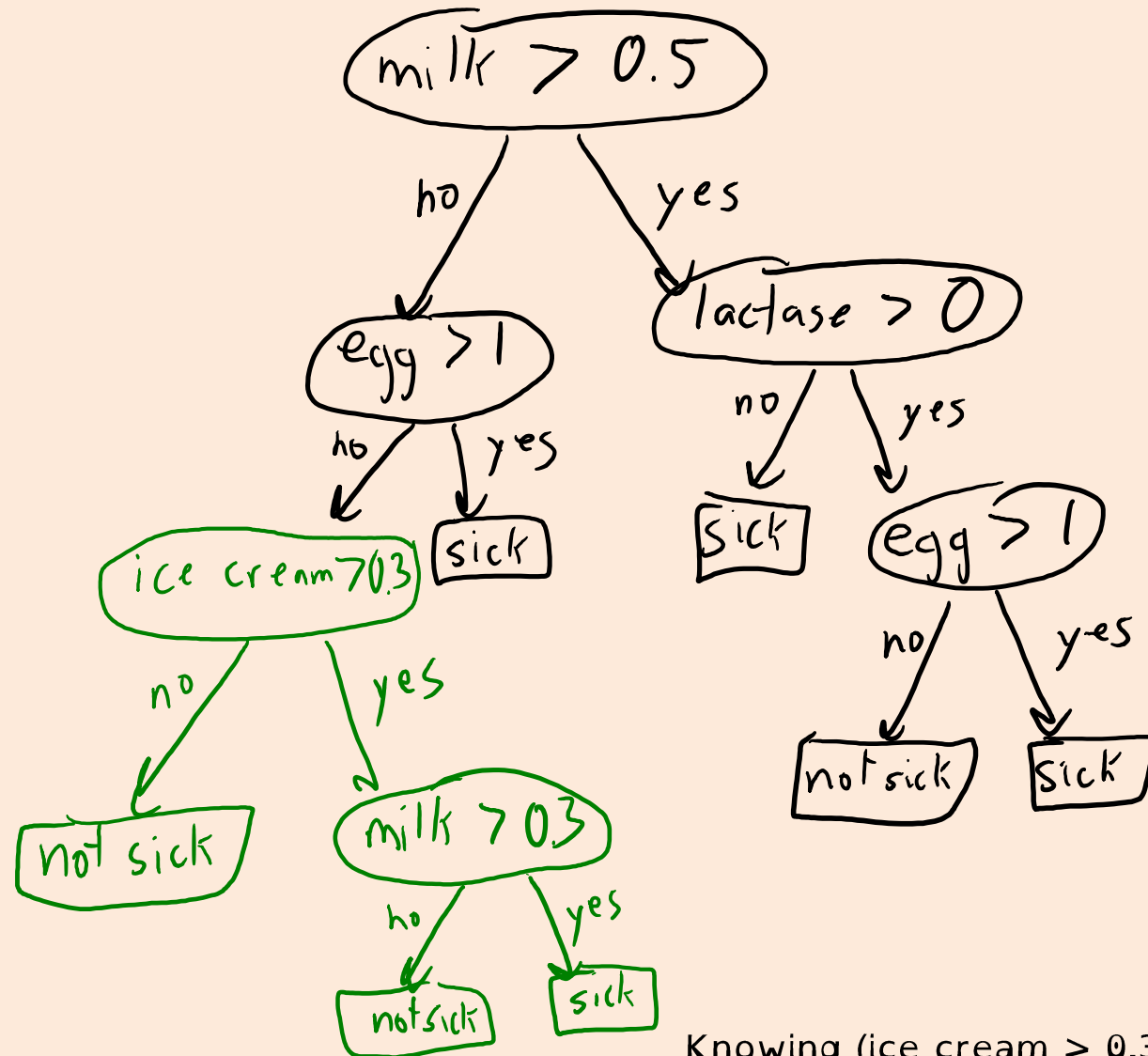
Knowing (ice cream > 0.3) makes small milk quantities relevan

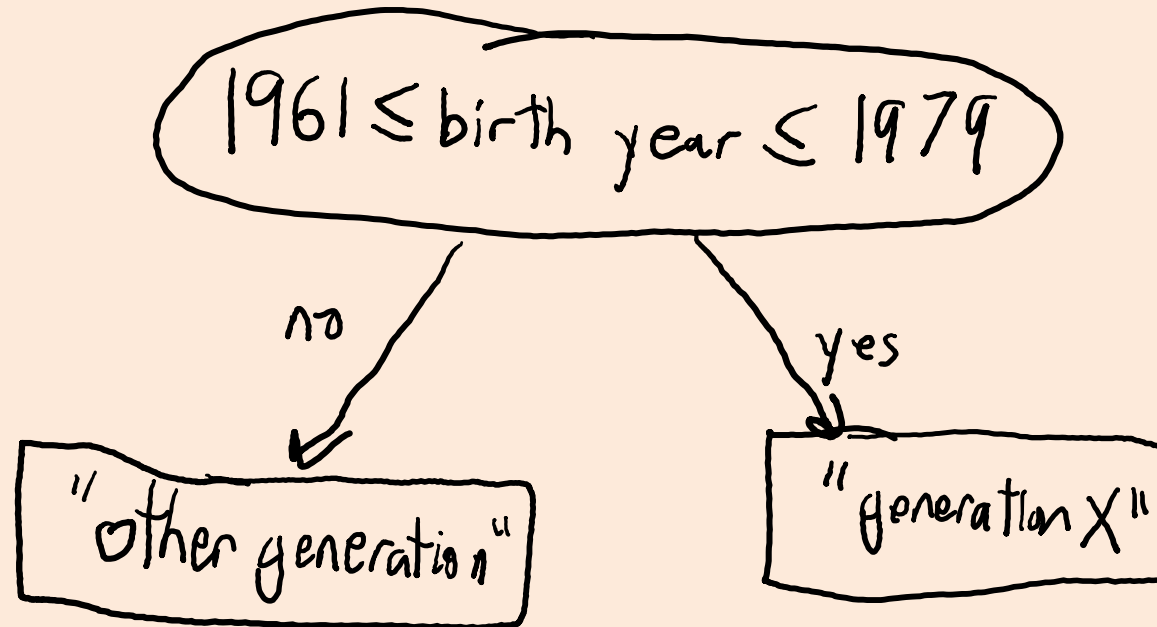# Can decision trees have more complicated rules?

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

milk + ice cream > 0.5

no → not sick

yes → sick

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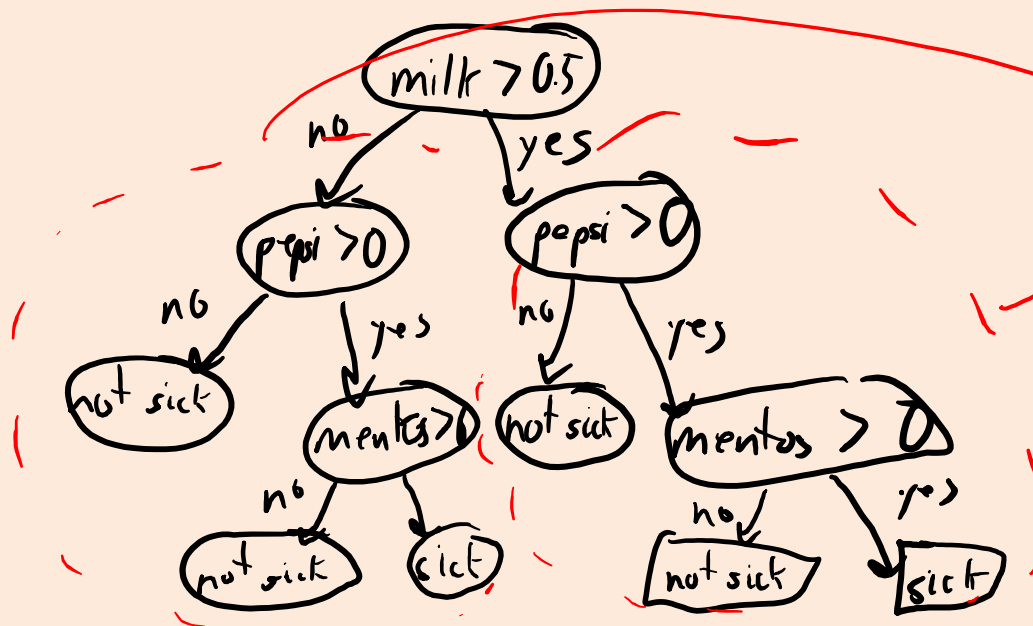
Get same sub-tree, you need learn it twice and with less data

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):
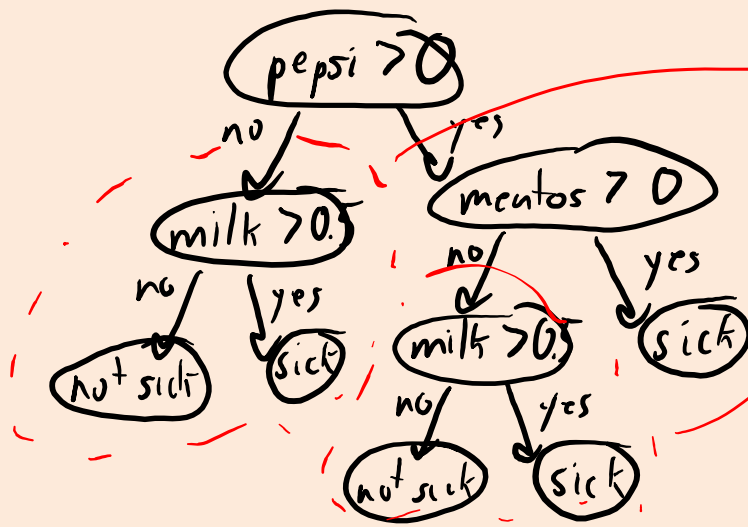
Decision Trees with Probabilistic Predictions

- 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.