100 Lectures on Machine Learning

This is a collection of course material from various courses that I've taught on machine learning at UBC, including material from over 100 lectures covering a large number of topics related to machine learning. The notation is fairly consistent across the topics which makes it easier to see relationships, and the topics are meant to be gone through *in order* (with the difficulty slowly increasing and concepts being defined at their first occurrence). I'm putting this in one place in case people find it useful for educational purposes.

Part 1: Computer Science 340

The first set of notes is mainly from the Fall 2019 version of CPSC 340, a course on machine learning and data mining. Related readings and assignments are available from the Fall 2019 course homepage. In the relevant places, I've also included some lectures from previous terms in cases where I covered different topics.

I've given a "title" to each lecture, but the length of time I spent on each topic usually did not exactly equal 50 minutes. This means that most of the topics are spread across more or less than one lecture. This does not matter if you go through the lectures in order, but if you "skip" to a certain topic you may need to look at the lecture before/after and there may be material from the previous topic included.

I made the first version of these notes in 2015, but Mike Gelbart has also been teaching the course since 2016 and has made numerous improvements. Although I've never had my lectures for this course recorded, videos of the lectures from the Winter 2018 section of this course taught by Mike Gelbart are available here (the material is largely the same).

Bonus Slides: Many lectures include "bonus material", and *these slides have a different background colour* (orange in the case of the 340 slides). These slides cover tangential or more-advanced topics, and should probably be skipped if this is the first time you are seeing this material. Also note that the lecture end on the slide titled "Summary", and typically all slides after this one only contain "bonus material" slides.

1. Supervised Learning

- Overview
- Exploratory Data Analysis
- Decision Trees (Notes on Big-O Notation)
- Fundamentals of Learning (Notation Guide)
- Probabilistic Classifiers (Probability Slides, Notes on Probability)
- Non-Parametric Models
- Ensemble Methods

2. Unsupervised Learning

- Clustering
- More Clustering
- Outlier Detection
- Finding Similar Items

3. Linear Models

- Least Squares (Notes on Calculus, Notes on Linear Algebra, Notes on Linear/Quadratic Gradients)
- Nonlinear Regression
- Gradient Descent
- Robust Regression
- Feature Selection
- Regularization
- More Regularization
- Linear Classifiers
- More Linear Classifiers
- Feature Engineering
- Convolutions
- Kernel Methods
- Stochastic Gradient
- Boosting
- MLE and MAP (Notes on Max and Argmax)

4. Latent-Factor Models

- Principal Component Analysis
- More PCA
- Sparse Matrix Factorization
- Recommender Systems
- Nonlinear Dimensionality Reduction

5. Deep Learning

- Deep Learning
- More Deep Learning
- Convolutional Neural Networks
- More CNNs

Part 2: Data Science 573 and 575

The second set of notes are from an assortment of other places where I've given lectures, mainly from courses in the Master of Data Science program, aimed at a target audience that is familiar with the above material.

- Structure Learning
- Sequence Mining
- Tensor Basics
- Semi-Supervised Learning
- PageRank

Part 3: Computer Science 440

The third set of notes is mainly from the January-April 2022 of CPSC 440, a course on machine learning that builds upon the material in CPSC 340. The notation in this course is similar to CPSC 340, except that we switch to using superscripts to refer to training examples (so that subscripts can refer to individual variables). Related readings and assignments are available from the course homepage.

A. Binary Random Variables

- Binary Density Estimation
- Bernoulli Distribution
- MAP Estimation
- Generative Classifiers
- Discriminative Classifiers
- Neural Networks
- Double Descent Curves
- Automatic Differentiation
- Convolutional Neural Networks
- Autoencoders
- Fully-Convolutional Networks

B. Categorical Random Variables

- Monte Carlo Approximation
- Conjugate Priors
- Bayesian Learning
- Empirical Bayes
- Multi-Class Classification
- What do we learn?
- Recurrent Neural Networks
- Long Short Term Memory
- Attention and Transformers

C. Gaussian Random Variables

- Univariate Gaussian
- Multivariate Gaussian (Motivation)
- Multivairate Gaussian (Definition)
- Learning Gaussians
- Bayesian Linear Regression
- End to End Learning
- Exponential Family

D. Markov Models

- Markov Chains
- Learning Markov Chains
- Message Passing
- Markov Chain Monte Carlo
- Directed Acyclic Graphical Models
- Learning Graphical Models
- Log-Linear Models

E. Latent-Variable Models

- Mixture Models
- EM and KDE (Notes on EM)
- HMMs and RBMs (Forward-Backward for HMMs)
- Topic Models and Variational Inference
- VAEs and GANs

Part 4: Computer Science 540

The fourth set of notes are from an assortment of other places where I have given lectures, mainly previous offerings of CPSC 540, aimed at a target audience that is familiar with the above material.

- Fundamentals of Learning
- More Fundamentals of Learning
- Convexity (Notes on Norms)
- More Convexity
- How Much Data?
- Faster Algorithms for Deep Learning?
- Probabilistic PCA, Factor Analysis, Independent Component Analysis
- Inference in Graphical Models
- Structured SVMs
- Expectation Maximization
- Non-Parametric Bayes
- Infinite Mixture Models

Part 5: Large-Scale Machine Learning

The fifth set of notes is related to one of my core research areas, which is continuous optimization algorithms designed specifically for machine learning problems. The below notes are mainly from a series of 13 lectures I gave in August 2020 on this topic.

- Convex Optimization (Notes on Norms)
- Gradient Descent Progress (Notes on Convexity Inequalities, Notes on Implementing Gradient Descent)
- Gradient Descent Convergence
- Linear and Superlinear Convergence
- Subgradient Methods
- Projected-Gradient
- Proximal-Gradient
- Structured Regularization
- Coordinate Optimization
- Mirror Descent and Multi-Level Methods
- Randomized Algorithms
- Stochastic Subgradient
- Variance-Reduced Stochastic Gradient
- Kernel Methods and Fenchel Duality
- Online Learning
- Over-Parameterized Models

Part 6: Machine Learning Reading Group

The final set of notes are topics that I have not covered in a formal course, but where I've given overviews in our machine learning reading group.

- Parallel and Distributed Machine Learning
- Online, Active, and Causal Learning
- Reinforcement Learning
- Overview of Other Large/Notable Topics

CPSC 340 and 532M:
Machine Learning and Data Mining

Mark Schmidt
University of British Columbia, Fall 2019
www.cs.ubc.ca/~schmidtm/Courses/340-F19

Some images from this lecture are taken from Google Image Search, contact me if you want the reference

Big Data Phenomenon

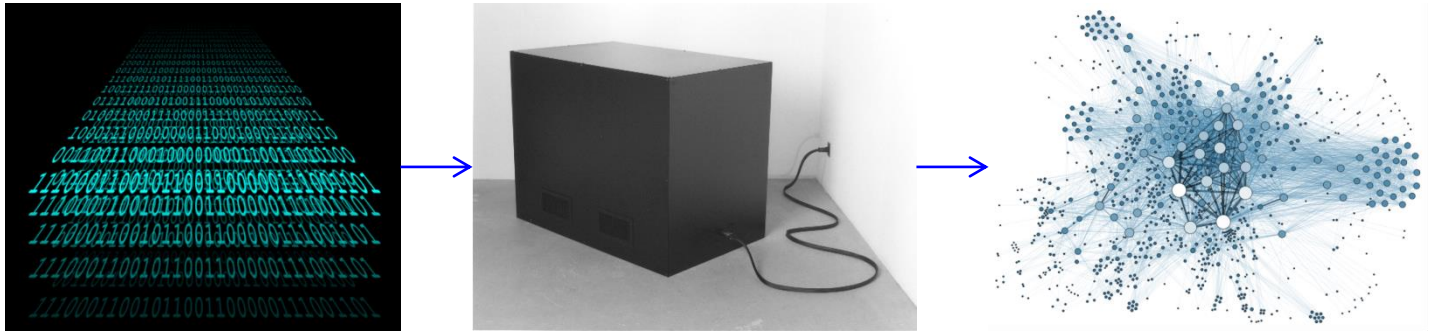
- We are collecting and storing data at an unprecedented rate.
- Examples:
  - YouTube, Facebook, MOOCs, news sites.
  - Credit cards transactions and Amazon purchases.
  - Transportation data (Google Maps, Waze, Uber)
  - Gene expression data and protein interaction assays.
  - Maps and satellite data.
  - Large hadron collider and surveying the sky.
  - Phone call records and speech recognition results.
  - Video game worlds and user actions.

Big Data Phenomenon

- What do you do with all this data?
  - Too much data to search through it manually.
- But there is valuable information in the data.
  - How can we use it for fun, profit, and/or the greater good?
- Data mining and machine learning are key tools we use to make sense of large datasets.

Data Mining

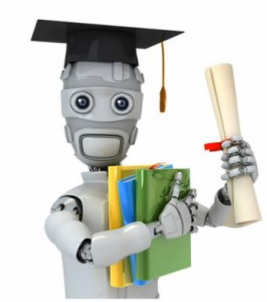
- Automatically extract useful knowledge from large datasets.

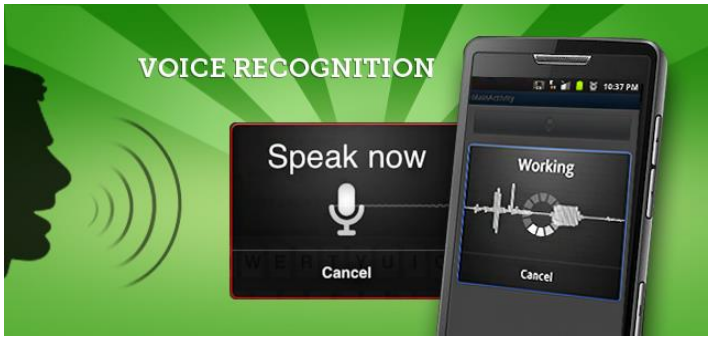


- Usually, to help with human decision making.

Machine Learning

- Using computer to automatically detect patterns in data and use these to make predictions or decisions.

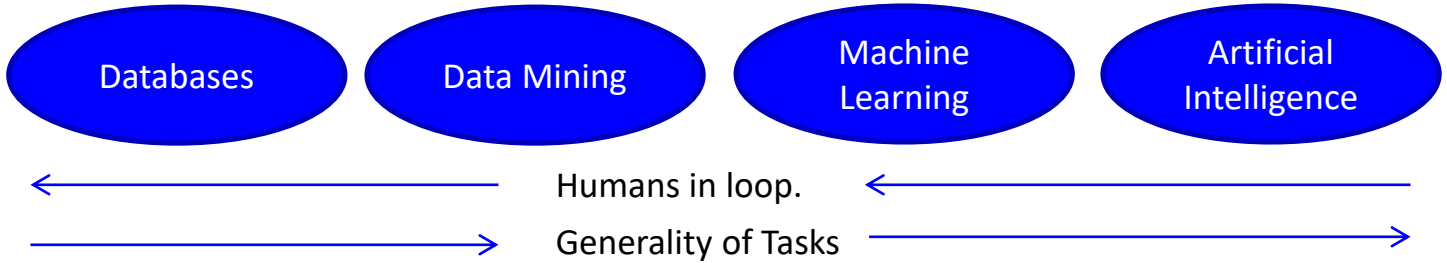




- Most useful when:
  - We want to automate something a human can do.
  - We want to do things a human can't do (look at 1 TB of data).

Data Mining vs. Machine Learning

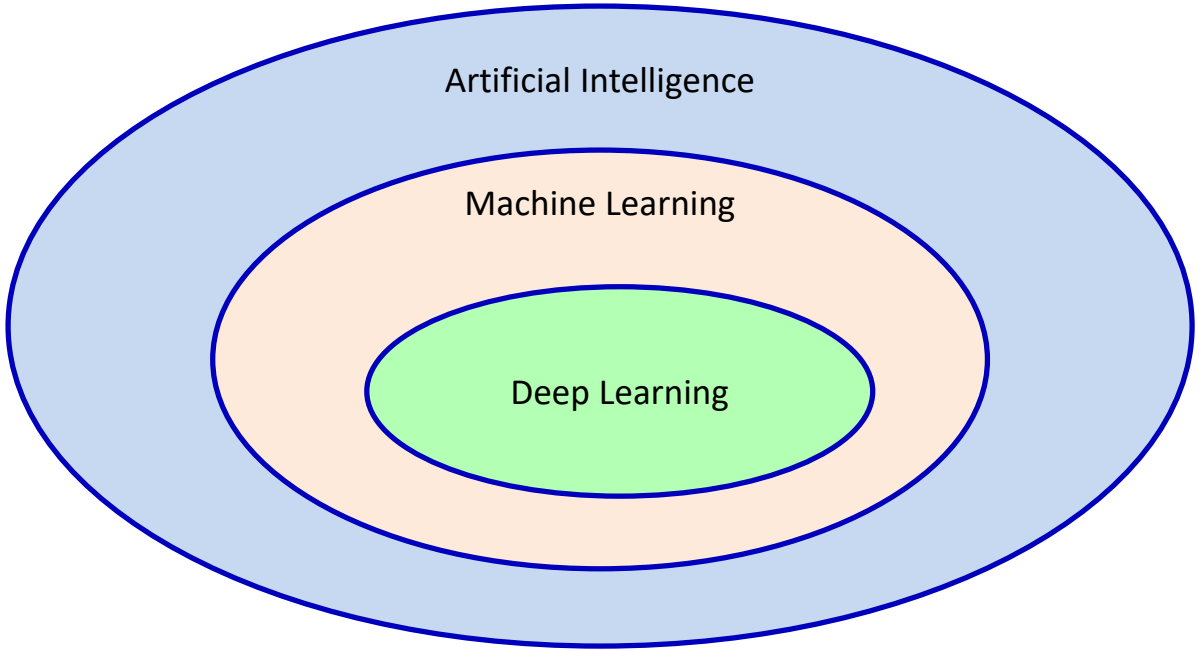
- Data mining and machine learning are very similar:
  - Data mining often viewed as closer to databases.
  - Machine learning often viewed as closer AI.



- Both are similar to statistics, but more emphasis on:
  - Large datasets and computation.
  - Predictions (instead of descriptions).
  - Flexible models (that work on many problems).

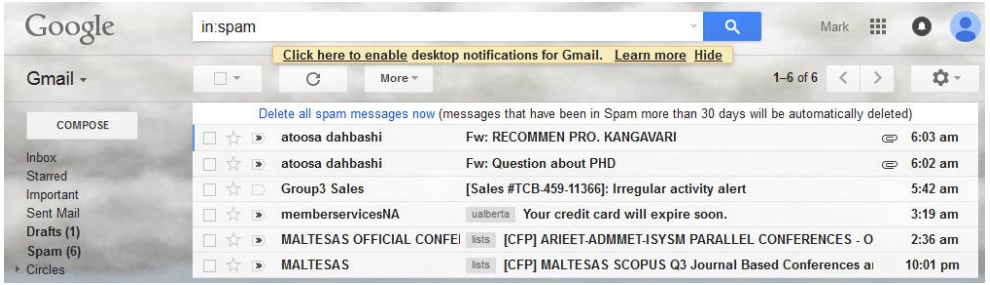
Deep Learning vs. Machine Learning vs. AI

- Traditional we've viewed ML as a subset of AI.
  - And "deep learning" as a subset of ML.

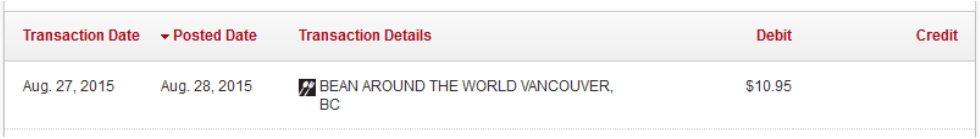


Applications

- Spam filtering:



- Credit card fraud detection:



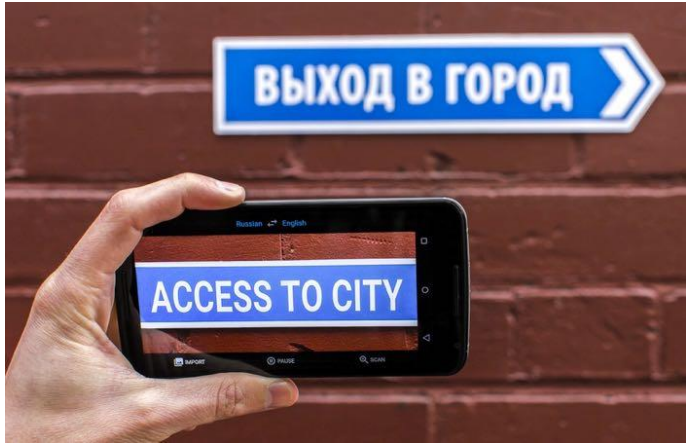
Transaction Date	Posted Date	Transaction Details	Debit	Credit
Aug. 27, 2015	Aug. 28, 2015	BEAN AROUND THE WORLD VANCOUVER, BC	$10.95	

- Product recommendation:

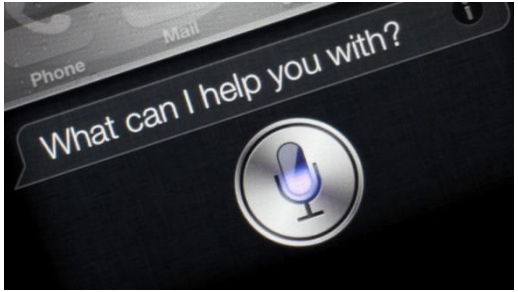
Applications

- Motion capture:

- Optical character recognition and machine translation:

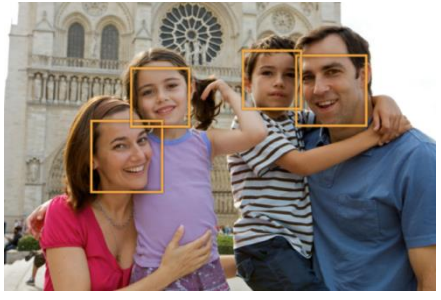


- Speech recognition:

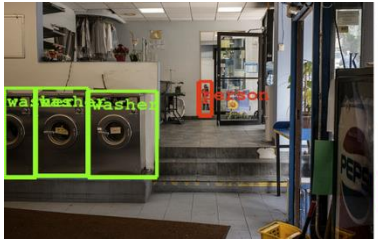


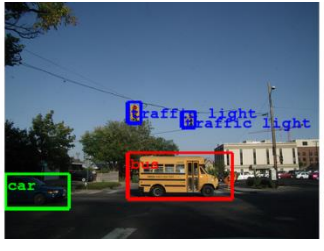
Applications

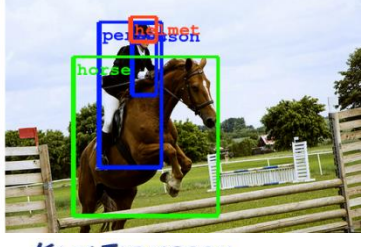
- Face detection:



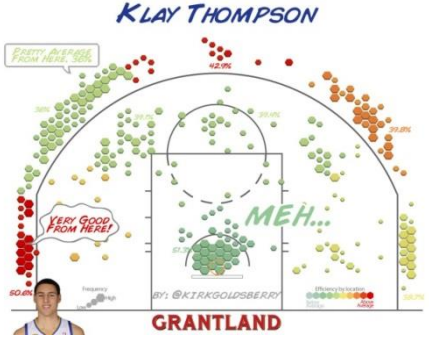
- Object detection:





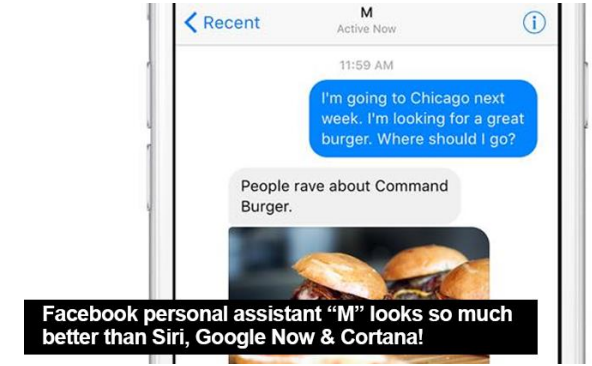


- Sports analytics:

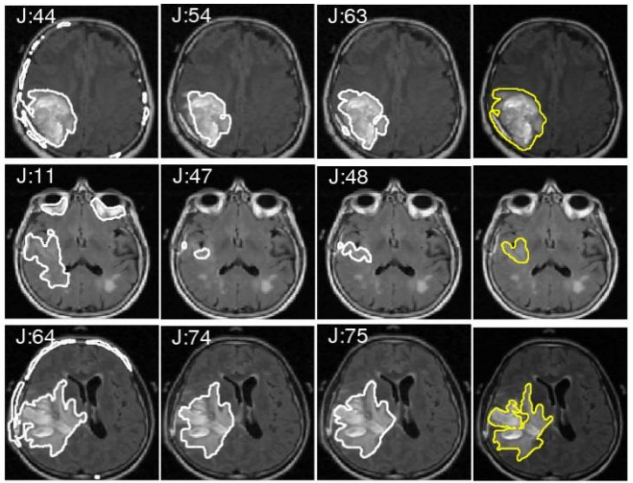


Applications

- Personal Assistants:



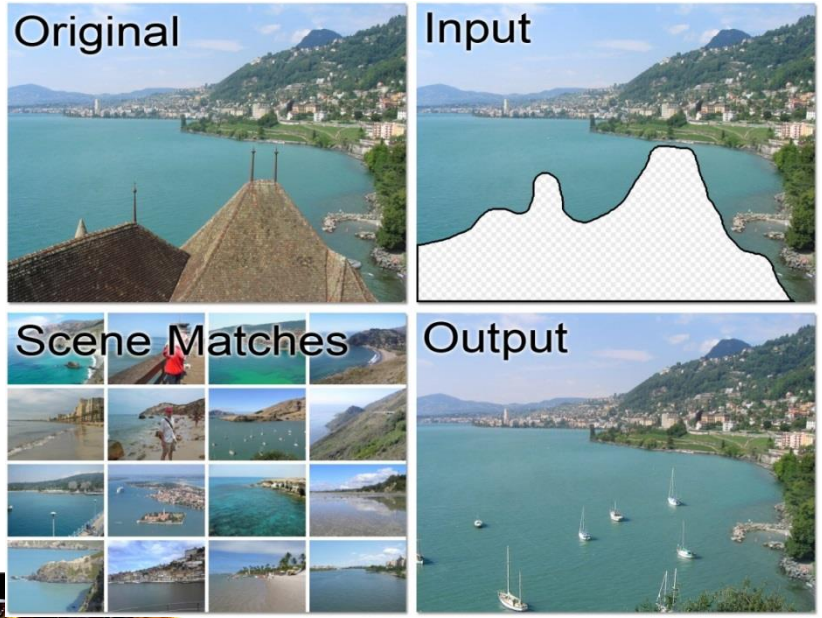
- Medical imaging:



- Self-driving cars:

Applications

- Scene completion:



- Image annotation:

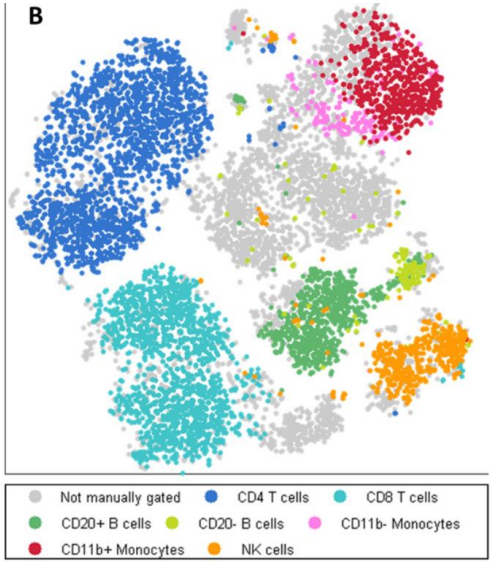

a cat is sitting on a toilet seat
logprob: -7.79


a display case filled with lots of different types of donuts
logprob: -7.78


a group of people sitting at a table with wine glasses
logprob: -6.71

Applications

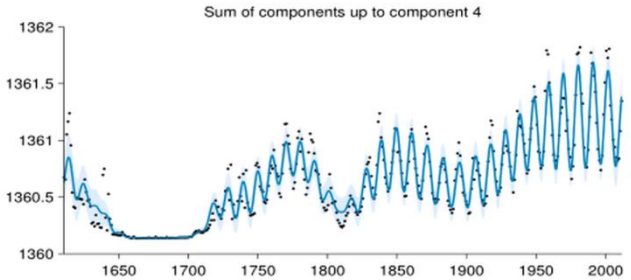
- Discovering new cancer subtypes:



- Automated Statistician:

2.4 Component 4 : An approximately periodic function with a period of 10.8 years. This function applies until 1643 and from 1716 onwards

This component is approximately periodic with a period of 10.8 years. Across periods the shape of this function varies smoothly with a typical lengthscale of 36.9 years. The shape of this function within each period is very smooth and resembles a sinusoid. This component applies until 1643 and from 1716 onwards.

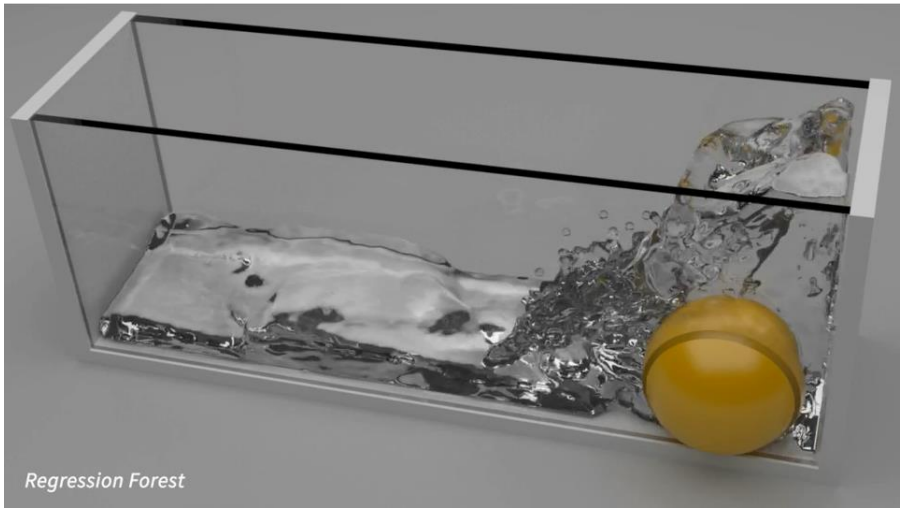


Applications

- Mimicking artistic styles:

Applications

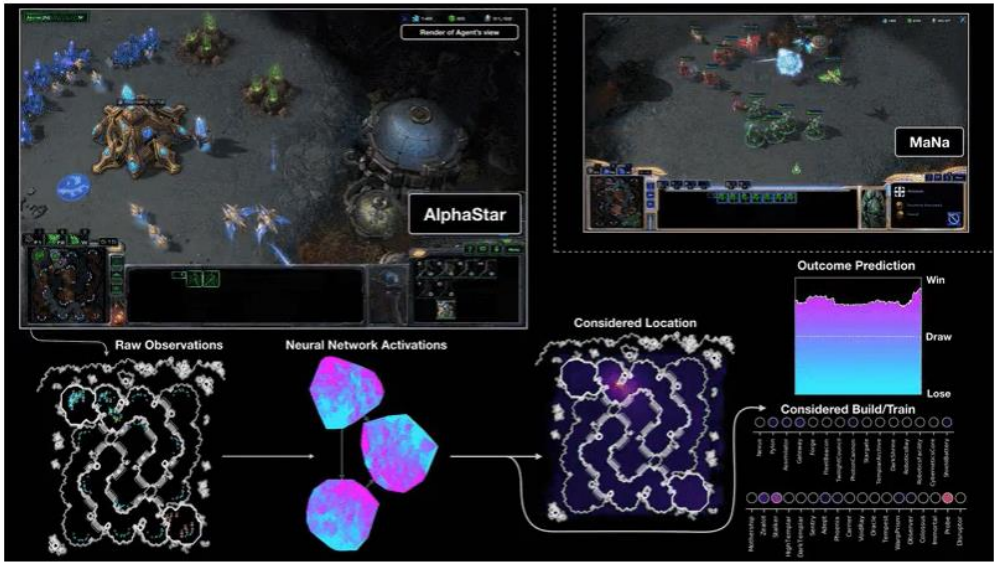
- Fast physics-based animation:



- Mimicking art style in video.
- Recent work on generating text/music/voice/poetry/dance.

Applications

- Beating humans in Go and Starcraft:



- Summary:
  - There is a lot you can do with a bit of statistics and a lot data/computation.

- We are in exciting times.
  - Major recent progress in fields like speech recognition and computer vision.
  - Things are changing a lot on the timescale of 3-5 years.
  - NeurIPS conference sold out in ~11 minutes last year.
  - A bubble in ML investments (most "AI" companies are just doing ML).

- But it is important to know the limitations of what you are doing.
  - "The combination of some data and an aching desire for an answer does not ensure that a reasonable answer can be extracted from a given body of data." – John Tukey
  - A huge number of people applying ML are just "overfitting".
    - Or don't understand the assumptions needed for them to work.
    - Their methods do not work when they are released "into the wild".

Bonus Slides

- I will include a lot of “bonus slides”.
  - May mention advanced variations of methods from lecture.
  - May overview big topics that we don’t have time for.
  - May go over technical details that would derail class.

- You are not expected to learn the material on these slides.
  - But they’re useful if you want to take 540 or work in this area.

- I’ll use this colour of background on bonus slides.

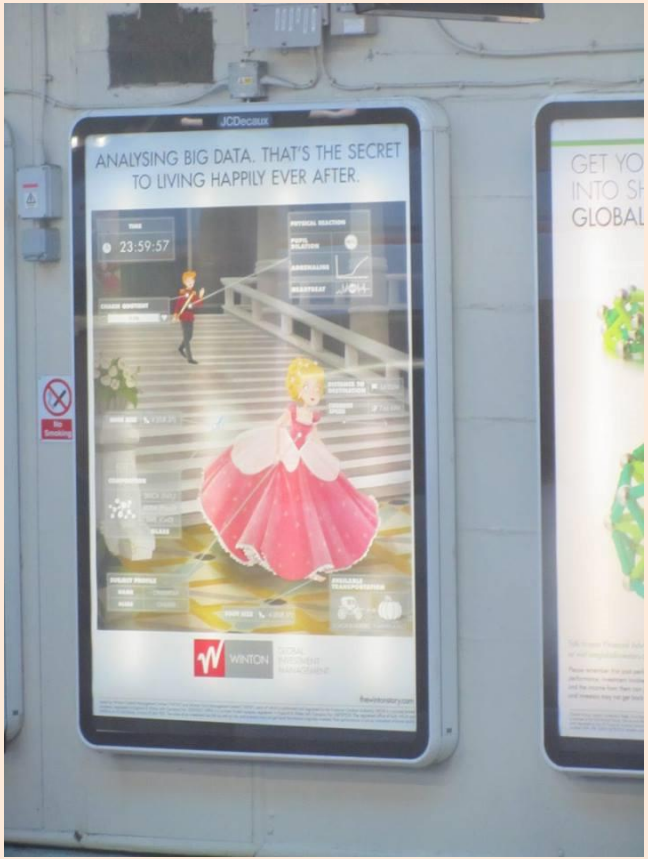
Course Outline

- Next class discusses “exploratory data analysis”.

- After that, the remaining lectures focus on five topics:
  1) Supervised Learning.
  2) Unsupervised learning.
  3) Linear prediction.
  4) Latent-factor models.
  5) Deep learning.

- “What is Machine Learning?” (overview of many class topics)

Photo I took in the UK on the way home from the “Optimization and Big Data” workshop:



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

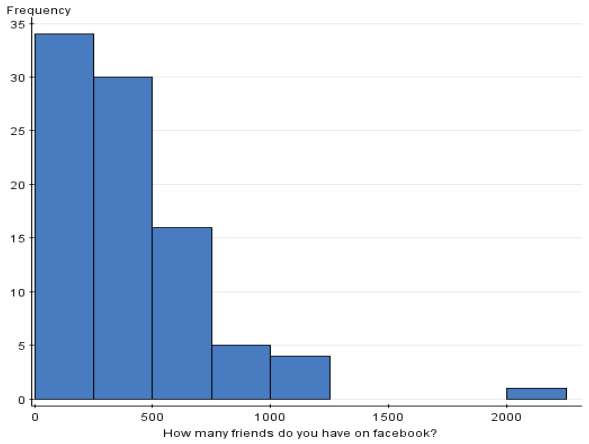
- We discussed example-feature representation:
  - Samples: another name we'll use for examples.

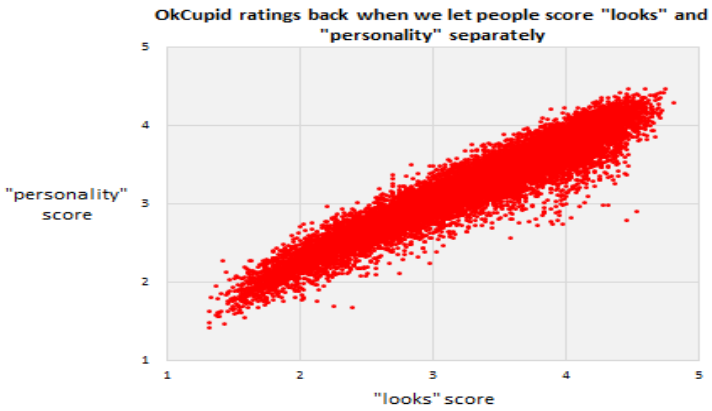
Age	Job?	City	Rating	Income
23	Yes	Van	A	22,000.00
23	Yes	Bur	BBB	21,000.00
22	No	Van	CC	0.00
25	Yes	Sur	AAA	57,000.00

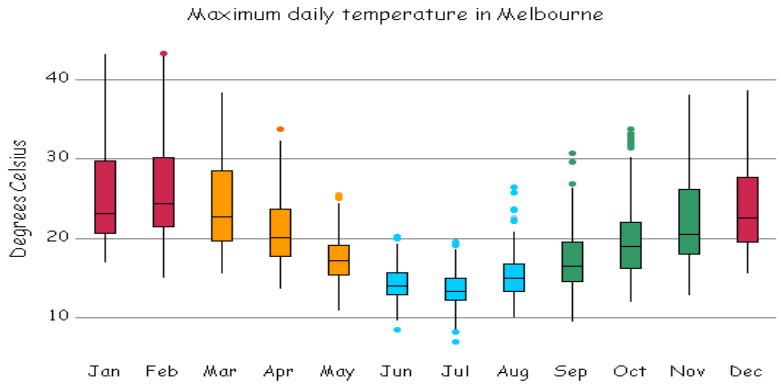
"Feature"

"Example"

- We discussed summary statistics and visualizing data.







http://www.statcrunch.com/5.0/viewresult.php?resid=1024581
http://cdn.okccdn.com/blog/humanexperiments/looks-v-personality.png
http://www.scc.ms.unimelb.edu.au/whatisstatistics/weather.html

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.

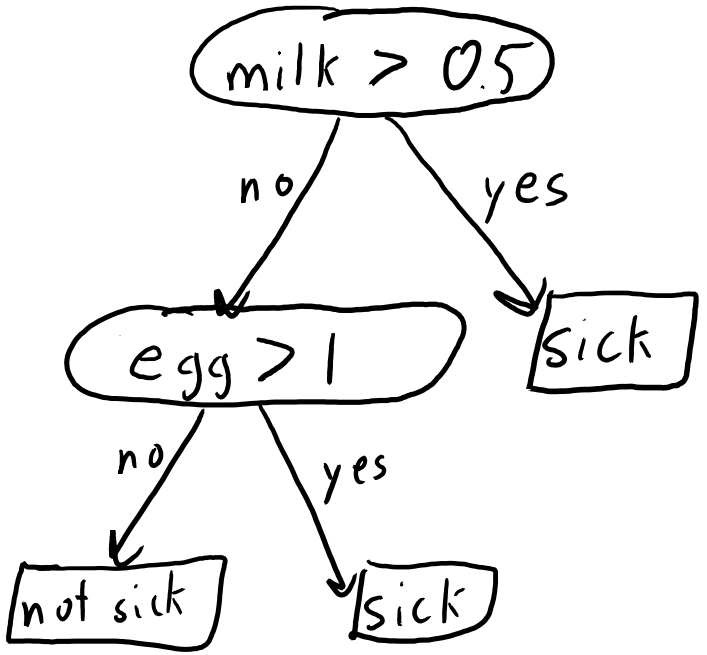
Decision Trees

- 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:

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

Can draw sequences of decisions as a tree:



Supervised Learning as Writing A Program

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

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

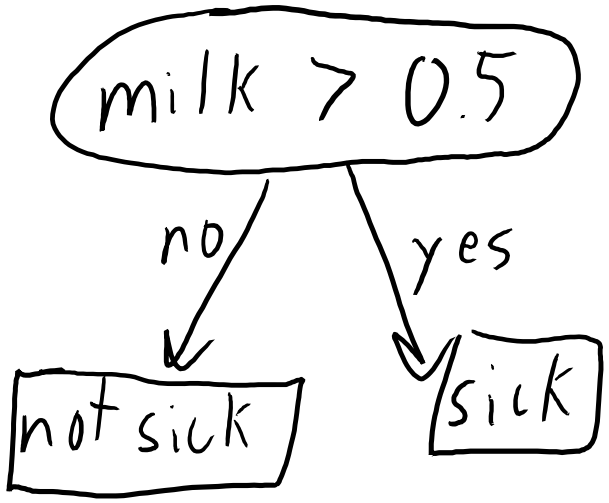
- Supervised learning is useful when you have lots of labeled data BUT:
  1. Problem is too complicated to write a program ourselves.
  2. Human expert can't explain why you assign certain labels.

     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

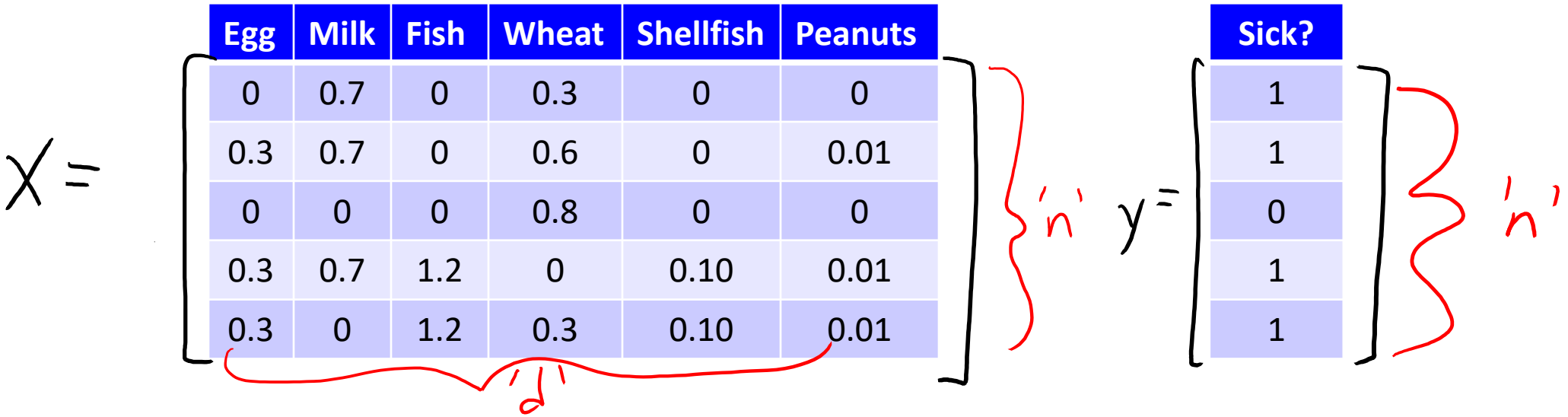
- Let's search for the decision stump maximizing classification score:

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

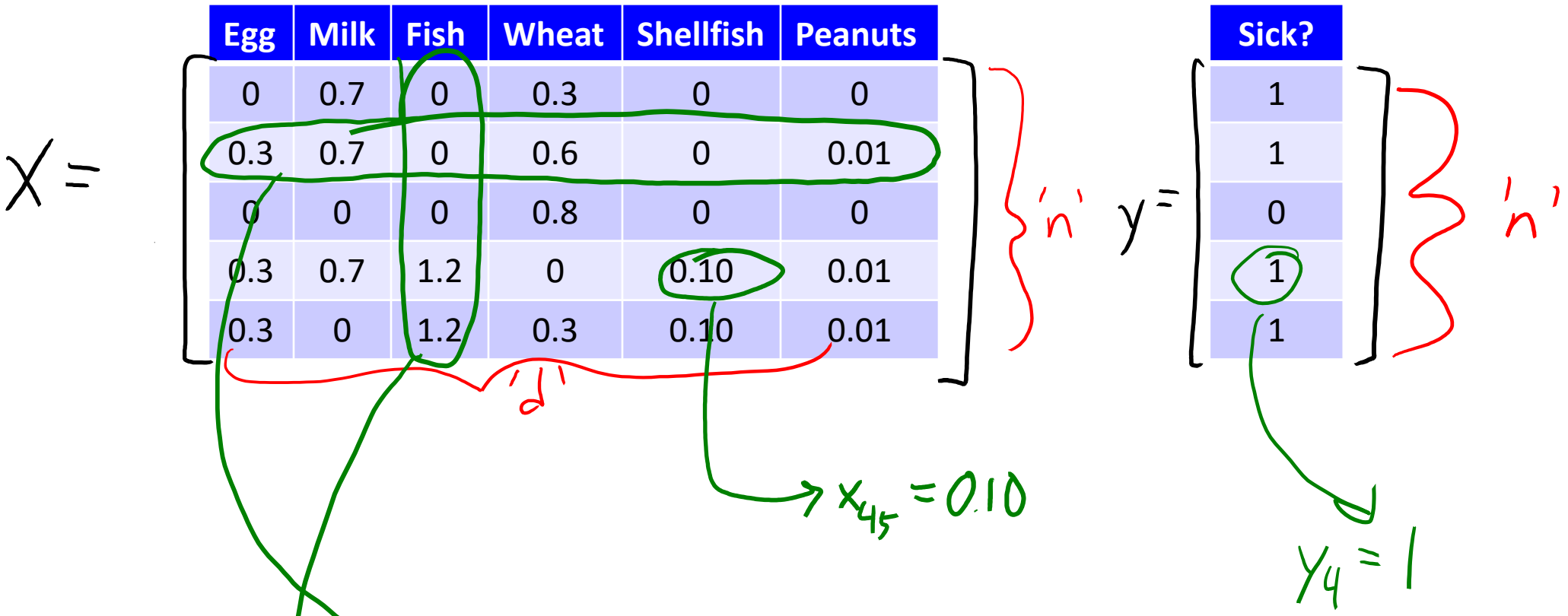
- Highest-scoring rule: (egg > 0) with leaves "sick" and "not sick".
- Notice we only need to test feature thresholds that happen in the data:
  - There is no point in testing the rule (egg > 3), it gets the "baseline" score.
  - There is no point in testing the rule (egg > 0.5), it gets the (egg > 0)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

Supervised Learning Notation (MEMORIZE THIS)



- 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)



Supervised Learning Notation (MEMORIZE THIS)

$X =$

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

'n' 'd'

$y =$

Sick?
1
1
0
1
1

'n'

- Training phase:
  - Use 'X' and 'y' to find a 'model' (like a decision stump).
- Prediction phase:
  - Given an 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_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")

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.

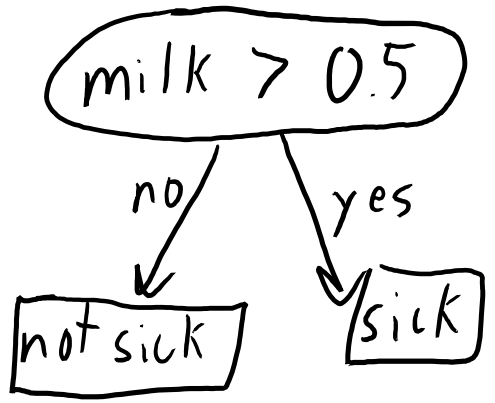
- Most common decision tree learning algorithm in practice:
  - Greedy recursive splitting.

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

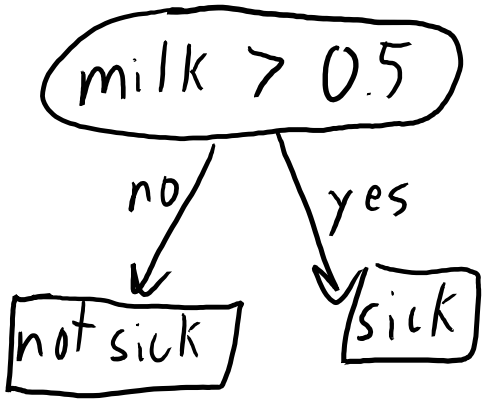
milk ≤ 0.5

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

>0.5

Greedy Recursive Splitting

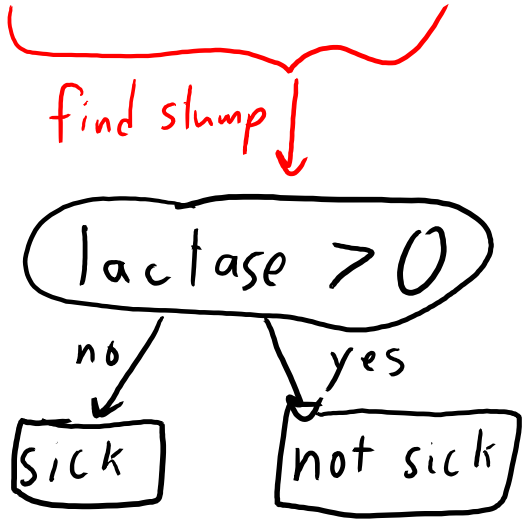
We now have a decision stump and two datasets:



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

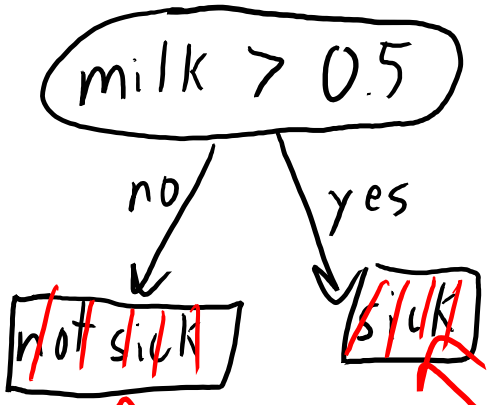
Egg	Milk	…	Sick?
0	0.7		1
1	0.7		1
1	0.6		1
2	0.6		1
0	1		1
1	0.6		0

Fit a decision stump to each leaf's data.



Greedy Recursive Splitting

We now have a decision stump and two datasets:



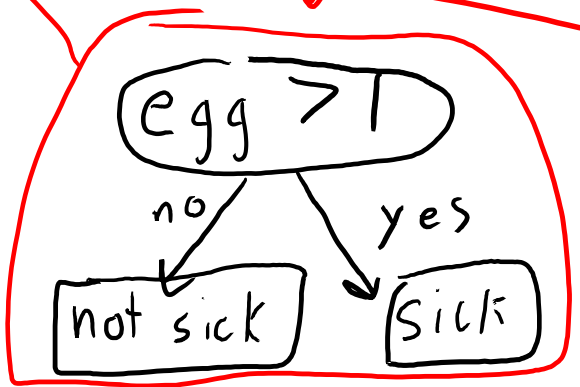
Egg	Milk	…	Sick?
0	0		0
1	0		0
2	0		1
0	0.3		0
2	0		1

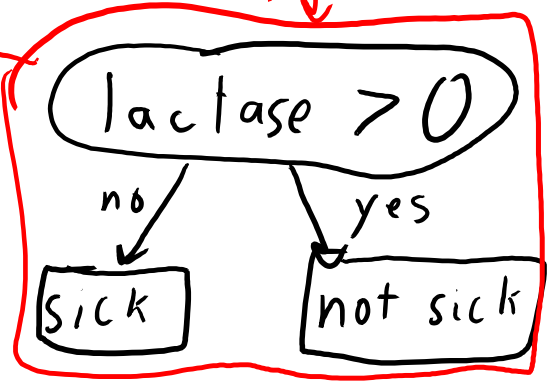
Egg	Milk	…	Sick?
0	0.7		1
1	0.7		1
1	0.6		1
2	0.6		1
0	1		1
1	0.6		0

find stump

find stump

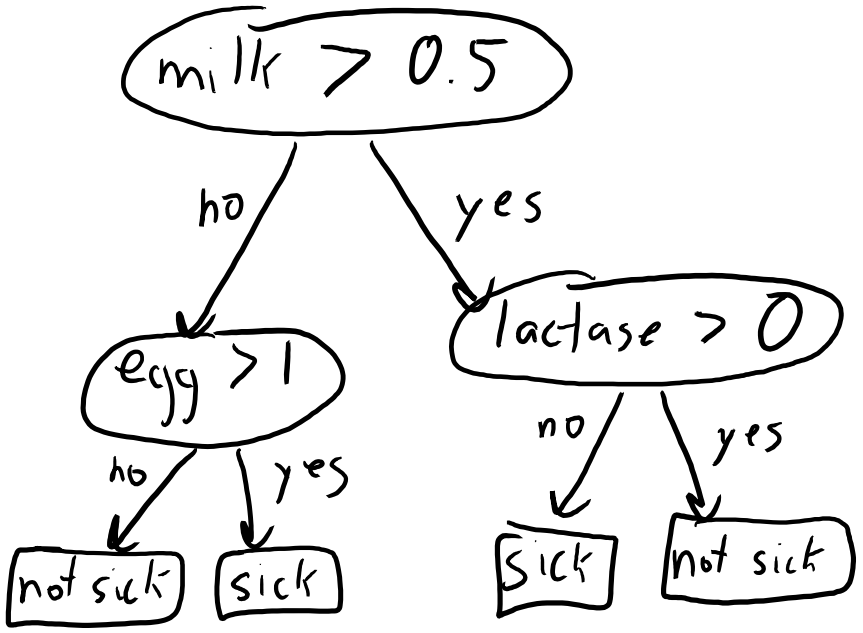
Fit a decision stump to each leaf's data.
Then add these stumps to the tree.





Greedy Recursive Splitting

This gives a "depth 2" decision tree:



It splits the two datasets into four datasets:

milk ≤ 0.5 data

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

milk > 0.5 data

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

milk ≤ 0.5, egg ≤ 1

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

Much more accurate!

milk ≤ 0.5, egg > 1

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

milk > 0.5, lactase ≤ 0

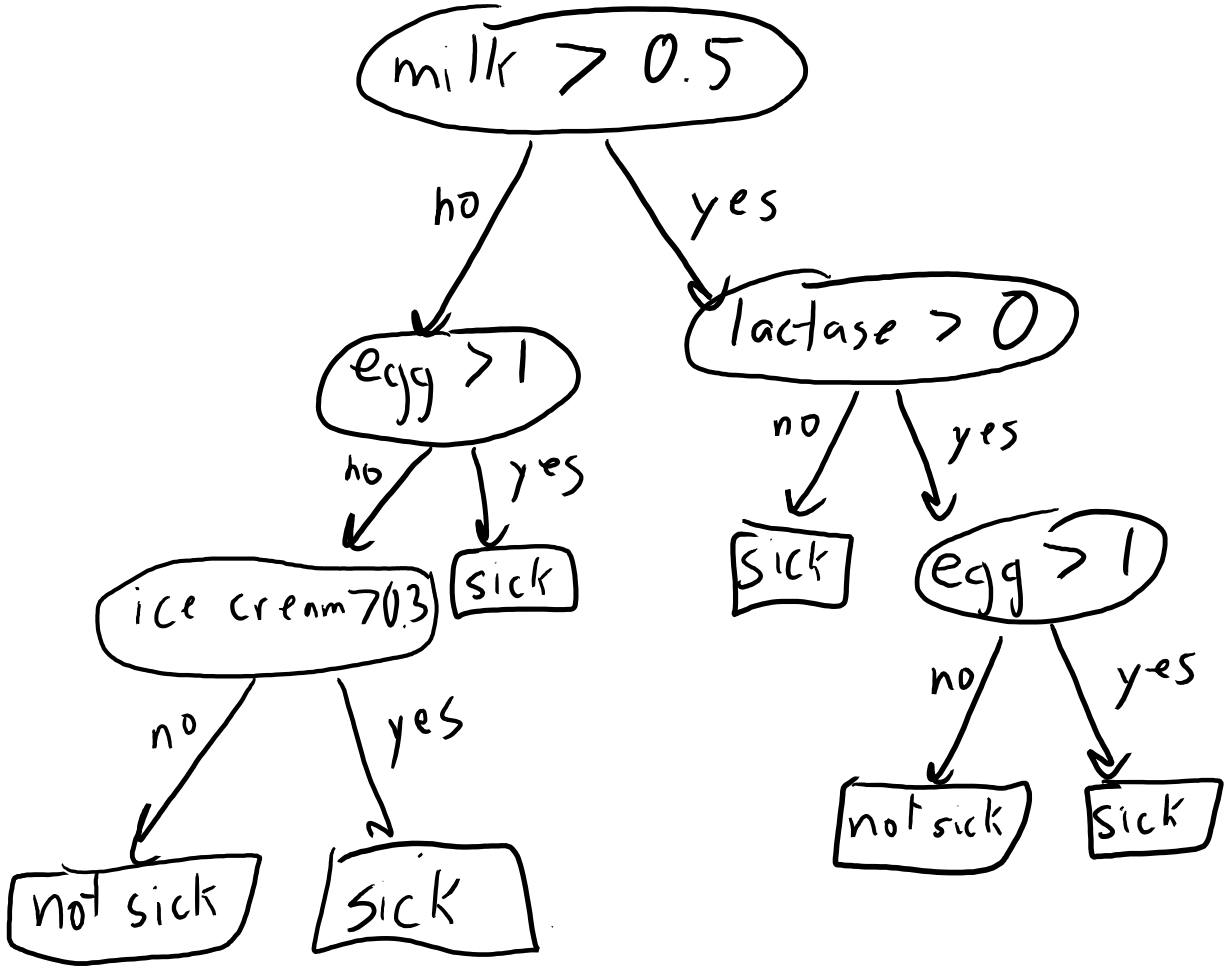
Egg	Milk	...	Sick?
0	0.7		1
1	0.7		1
1	0.6		1
2	0.6		1

milk > 0.5, lactase > 0

Egg	Milk	...	Sick?
1	0.6		0

Greedy Recursive Splitting

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



We might continue splitting until:

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

Which score function should a decision tree used?

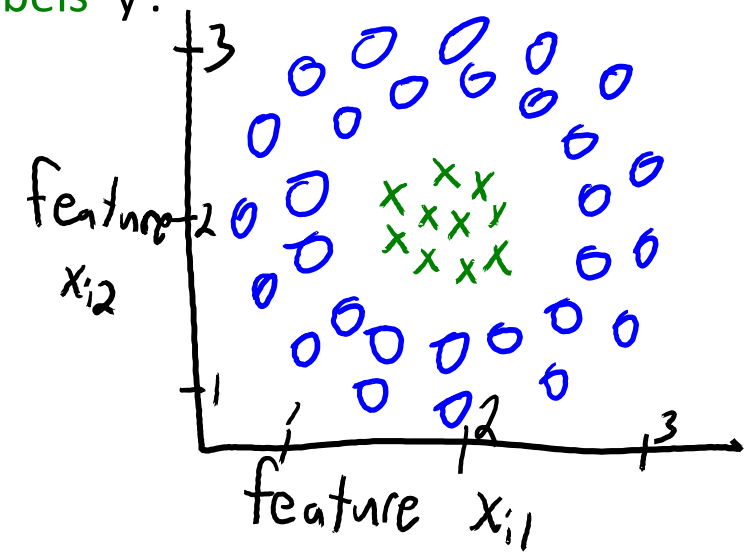
- Shouldn’t we just use accuracy score?
  - For leafs: yes, just maximize accuracy.
  - For internal nodes: not necessarily.

- Maybe no simple rule like (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'.

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



- 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 used?

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

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

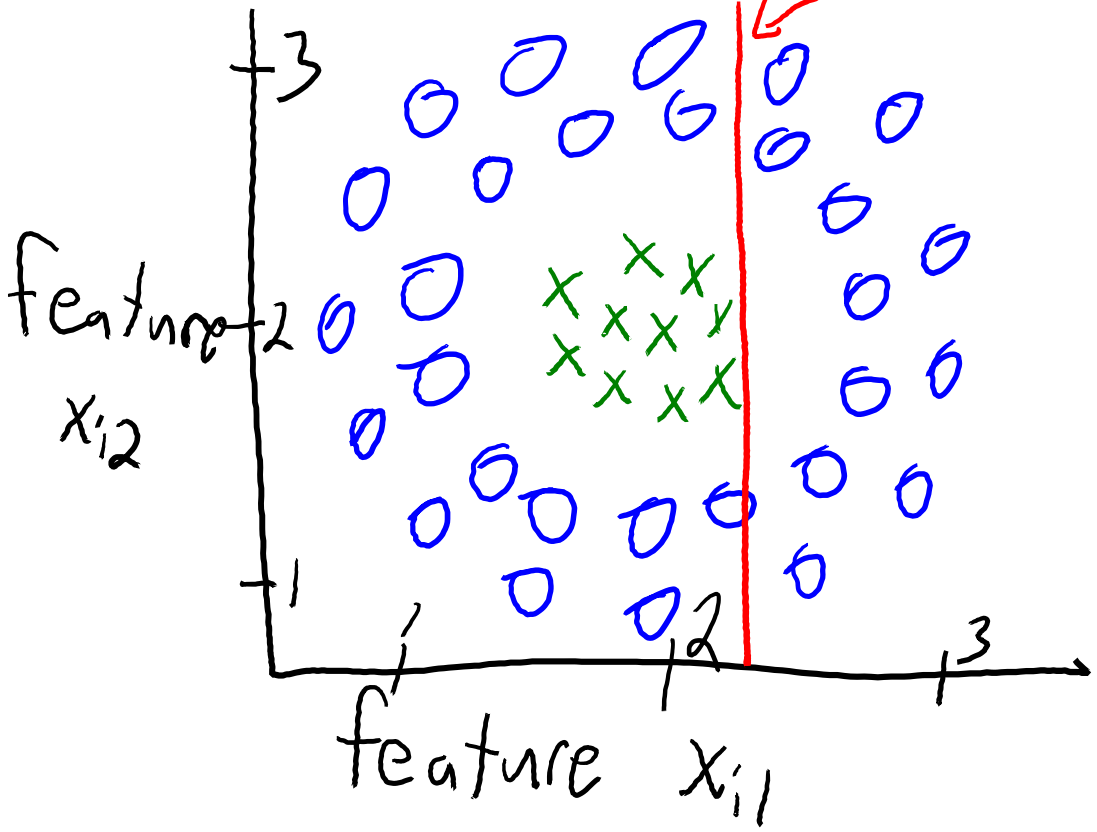
entropy of labels before split

number of examples satisfying rule

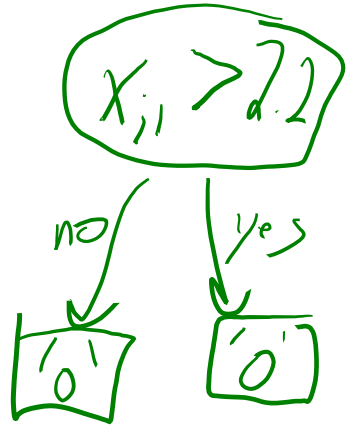
entropy of labels for examples satisfying rule.

- Information gain for baseline rule ("do nothing") is 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

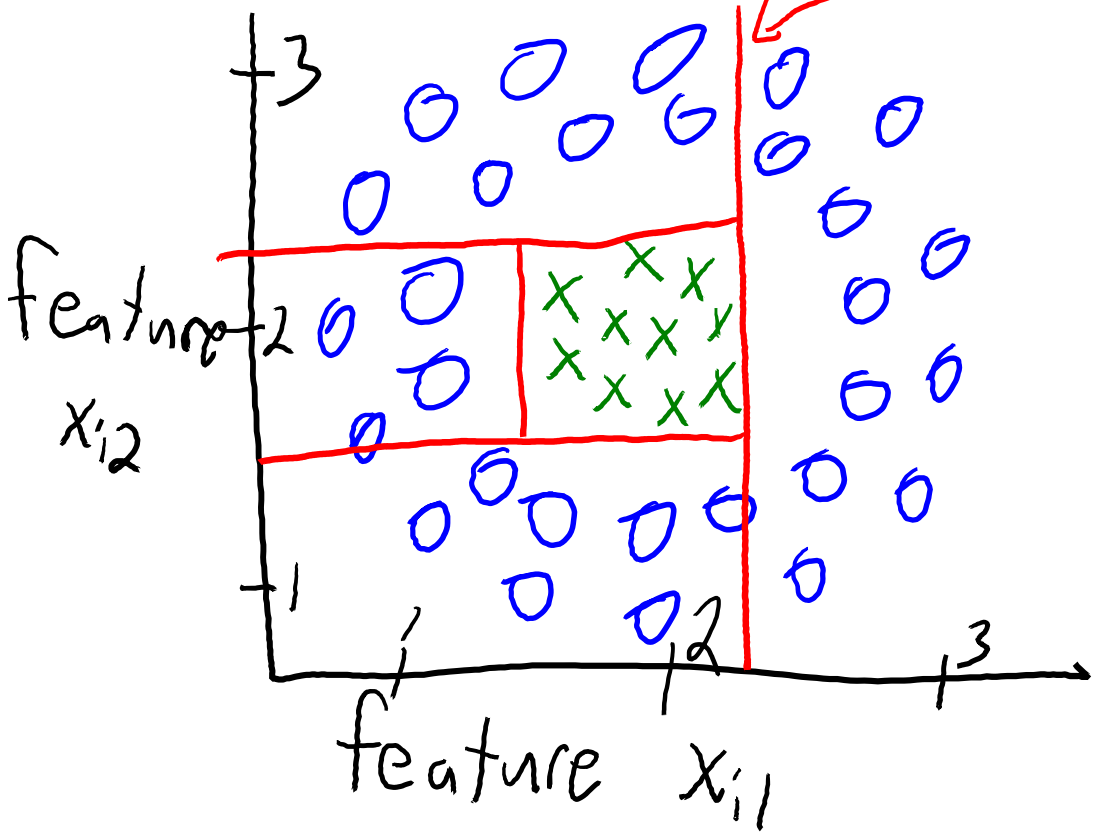


This split makes labels less random.
(Everything on the right is a 'o')

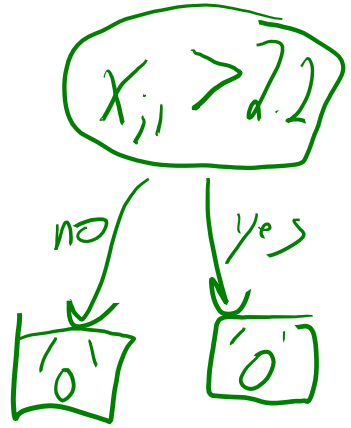


It did not improve accuracy.
(still classifies everything as 'o')

Example Where Accuracy Fails



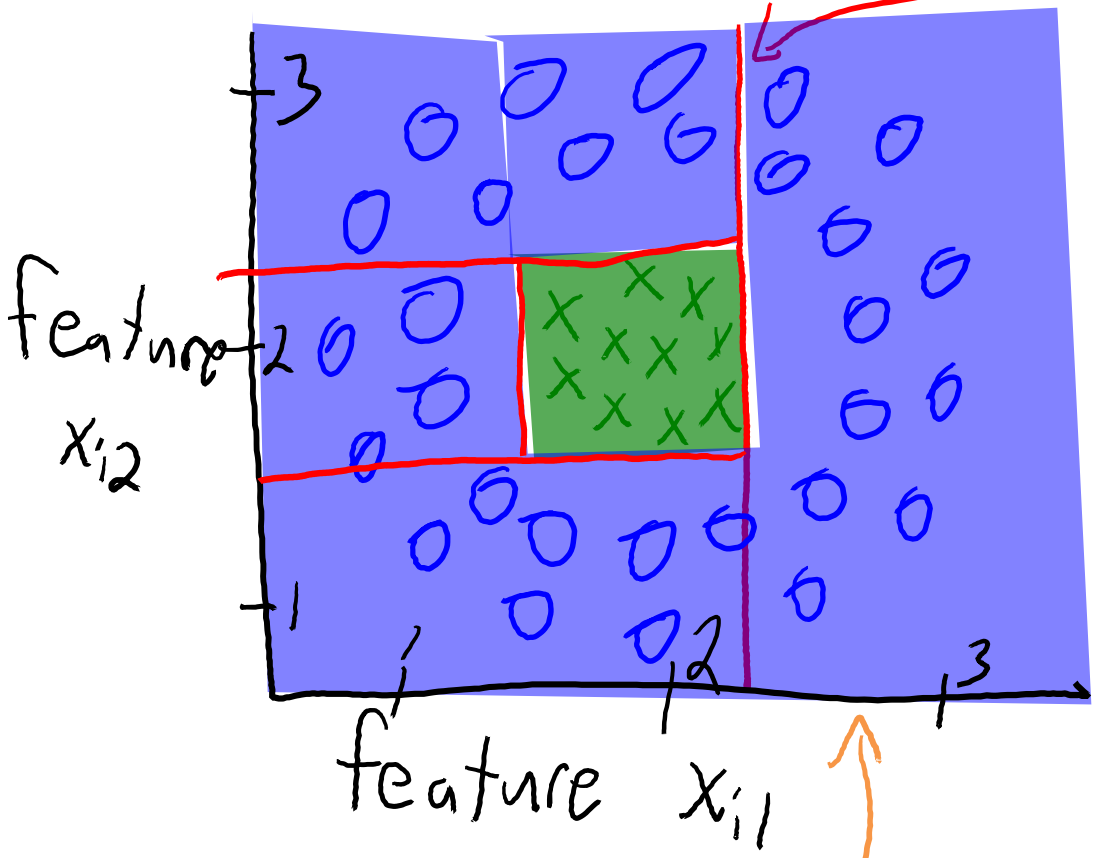
This split makes labels less random.
(Everything on the right is a 'o')



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

But three more splits maximizing infogain lead to perfect accuracy.

Example Where Accuracy Fails



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

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

But three more splits maximizing infogain lead to perfect accuracy.

Using colors to show decisions in regions of space

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 $\{1, 2, \ldots, k\}$

```
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” ⇔ “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 i = 1:n
        t = X[i,j]
        y_above = mode(y[X[:,j] .> t])
        y_below = mode(y[X[:,j] .<= t])
        yhat = fill(y_above, n)
        yhat[X[:,j] .<= t] = y_below
        error = sum(yhat != y)
        if error < minError
            minError = error
            minRule = [j t]
```

for each feature 'j'

for each example 'i'

set threshold to feature 'j' in example 'i'!

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

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

classify all examples based on threshold

count the number of errors.

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 (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.

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

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

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

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

How do we fit stumps in O(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).

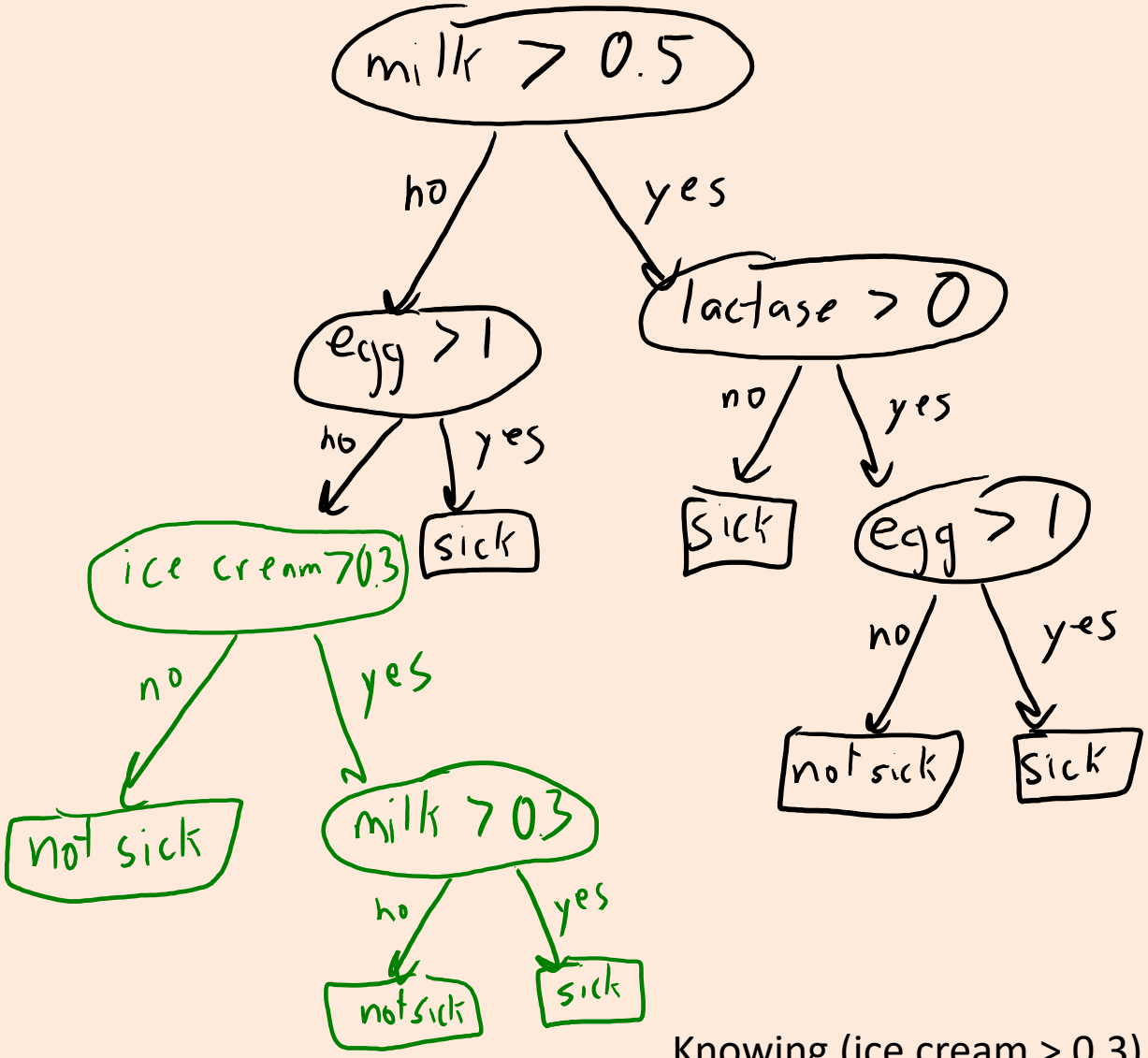
With ‘k’ labels, you can decrease cost using a “max-heap” data structure:

- 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?

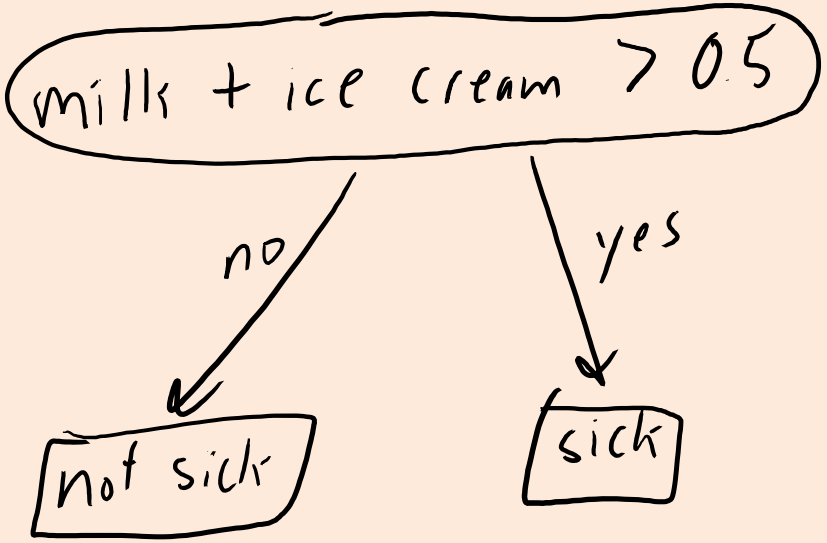
- Yes.



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

Can decision trees have more complicated rules?

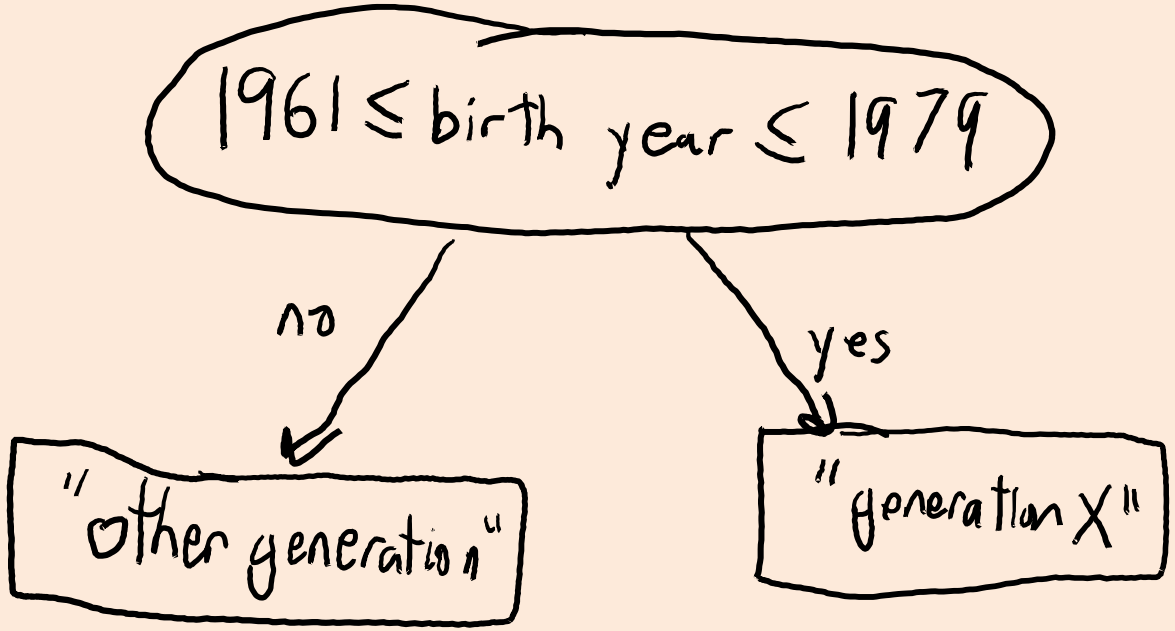
- Yes!
- Rules that depend on more than one feature:



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

Can decision trees have more complicated rules?

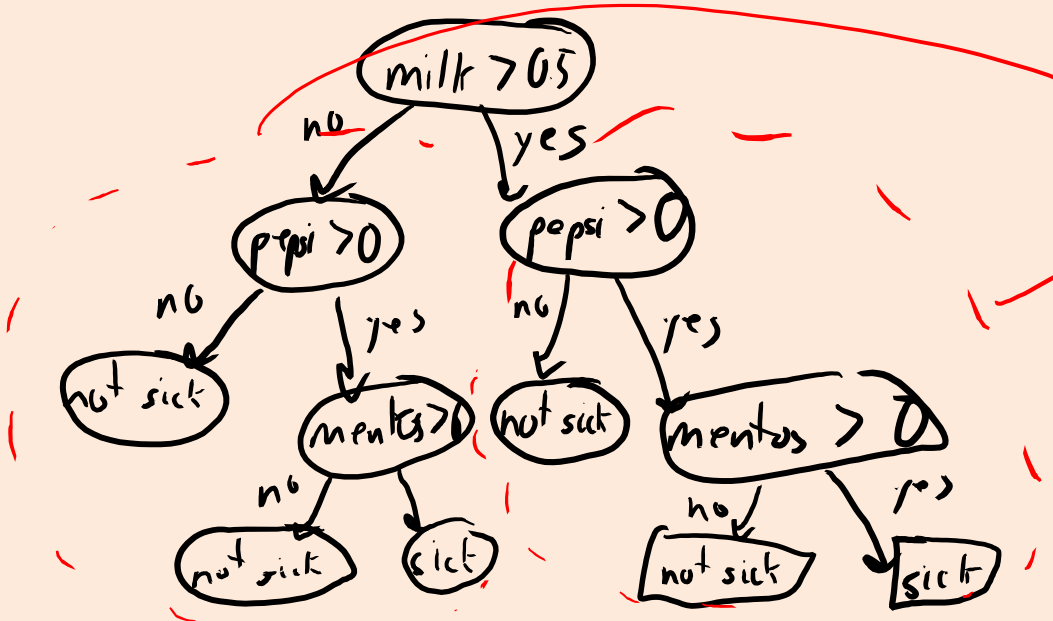
- Yes!
- Rules that depend on more than one threshold:



- "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?

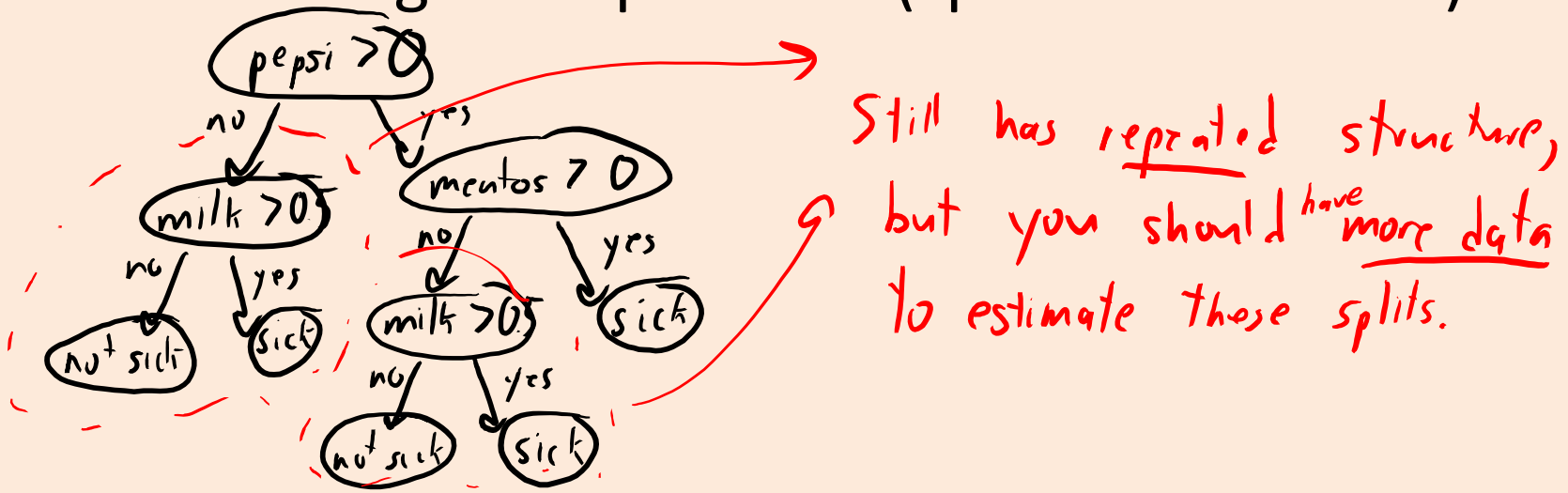
- Can't you just go deeper to correct greedy decisions?
  - Yes, but you need to "re-discover" rules with less data.
- Consider that you are allergic to milk (and drink this often), and also get sick when you (rarely) combine diet coke with mentos.
- Greedy method should first split on milk (helps accuracy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



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.

Overview of Big-O Notation

Mark Schmidt

September 14, 2015

Review of Big-O Notation

- The notation "g(n) = O(f(n))" means:
  - "for all large n, g(n) less than c*f(n) for some constant c > 0".
- Examples:

  $20n + 50 = O(n)$.

  $5n^2 + 34n + 3 = O(n^2)$

  $10 = O(1)$.

  $10*\log(n) + n = O(n)$.

  $n*\log(n) + 20*n = O(n \log n)$.

  $2^n + 1000*n^{10} = O(2^n)$.

Review of Big-O Notation

- "Runtime of algorithm is O(n)" means that:
  - "In worst case, algorithm requires O(n) operations."
  - Typically, 'n' will measure size of input.

- Why is this important?
  - We can only apply $O(2^n)$ algorithms to tiny datasets.
  - We can apply $O(n^2)$ algorithms to medium-sized dataset.
  - We can apply an O(nd) algorithm if one of 'n' or 'd' is medium-sized.
  - We can apply O(n) algorithms to huge datasets.

Review of Big-O Notation

- Examples of algorithm runtimes:
  - Finding the maximum in a list of ‘n’ numbers: O(n).
    - We do a constant number of operations for each of the ‘n’ numbers.
  - Finding item number ‘i’ in a list of ‘n’ numbers: O(1).
    - Just return that number.
  - Printing each element times each other element: $O(n^2)$:
    - A constant number of operations for each combination.
  - Finding an element in a sorted list: O(log n).
  - Sorting a list of N numbers: O(n log n).
    - Well-known result, see algorithms class.
  - Classifying an example in depth-m decision tree: O(m).
    - You do a constant number of operations at each depth.
    - N.B., no dependence on dimension ‘d’.

Review of Big-O Notation

- Examples of algorithm runtimes depending on two parameters:
  - Finding the maximum of an ‘n’ by ‘d’ matrix: O(nd)
  - Fitting decision stump with ‘n’ objects and ‘d’ features:
    - $O(n^2 d)$ if score costs O(n).
    - O(n d log n) if all scores cost O(n log n).
  - Fitting a decision tree of depth ‘m’:
    - Naïve analysis: 2^{m-1} trees, so $O(2^{m-1}$ n d log(n)).
    - But each object appears once at each depth: O(m n d log n).
  - Finding optimal decision tree:
    - There are O(C(n)*n!) tree structures, where C(n) is huge.
    - Then you also have to find the thresholds for each structure.
    - You are never going to do this.

CPSC 340:
Machine Learning and Data Mining

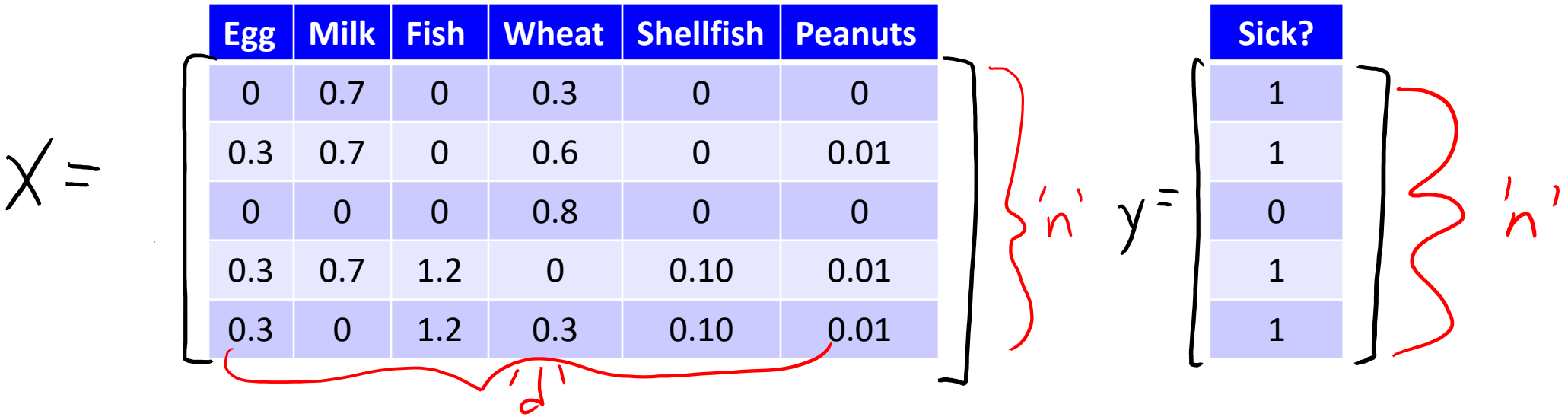
Fundamentals of Learning

Fall 2019

Admin

- Assignment 1 is due Friday: you should be almost done.
  - You need a CS ugrad ID to get signed up for Gradescope.
    - https://www.cs.ubc.ca/getacct
- Waiting list people: everyone should be in soon?
- Course webpage:
  - https://www.cs.ubc.ca/~schmidtm/Courses/340-F19/
- Auditors:
  - Bring your forms at the end of class Friday, assuming we clear wait list.
- Exchange students:
  - If you are still having trouble registering, bring your forms Friday.
  - Contact us on Piazza about getting registered for Gradescope.
- Midterm confirmed (October 17th, 6:30-8:30pm, SWNG 121 and 221).

Last Time: Supervised Learning Notation



- 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 Application

- We motivated supervised learning by the “food allergy” example.

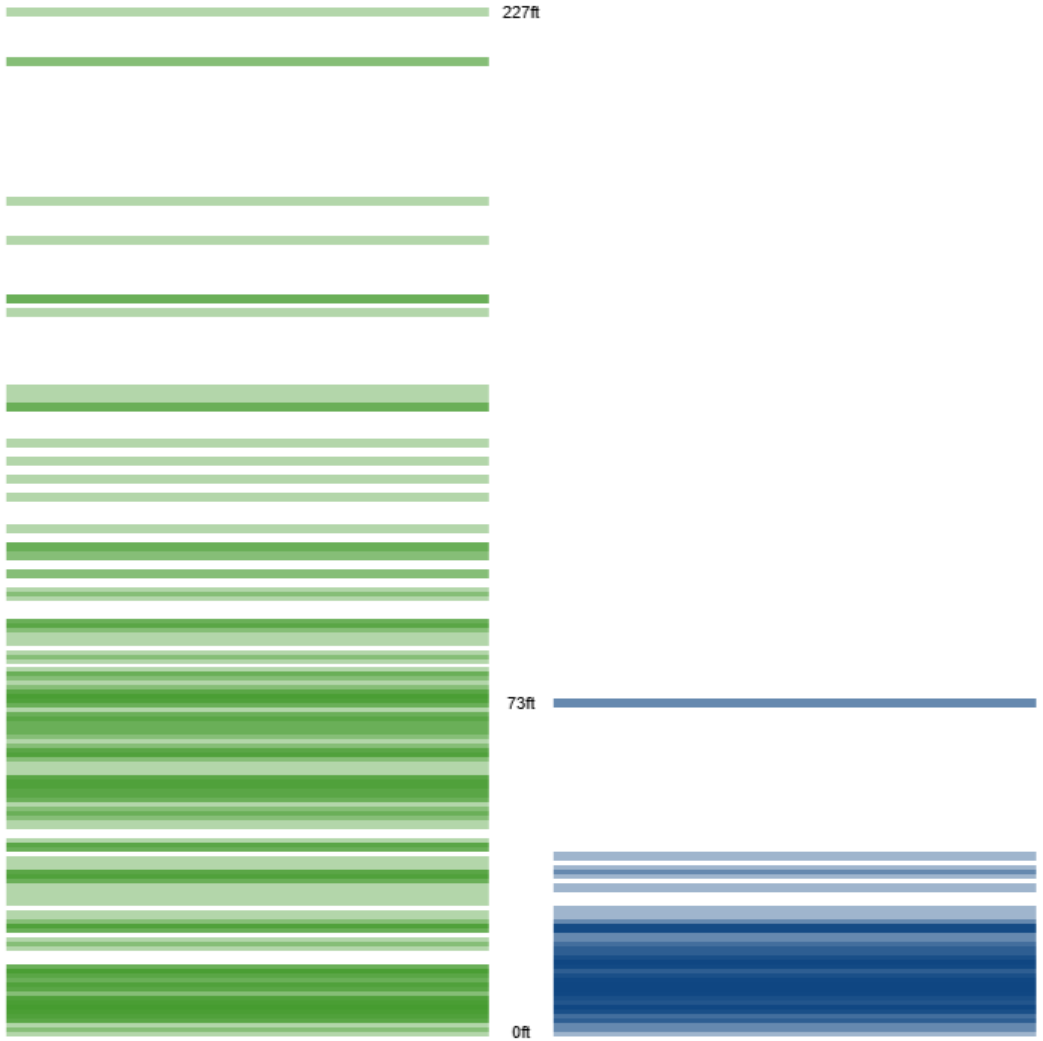
- But we can use supervised learning for any input:output mapping.
  - E-mail spam filtering.
  - Optical character recognition on scanners.
  - Recognizing faces in pictures.
  - Recognizing tumours in medical images.
  - Speech recognition on phones.
  - Your problem in industry/research?

Motivation: Determine Home City

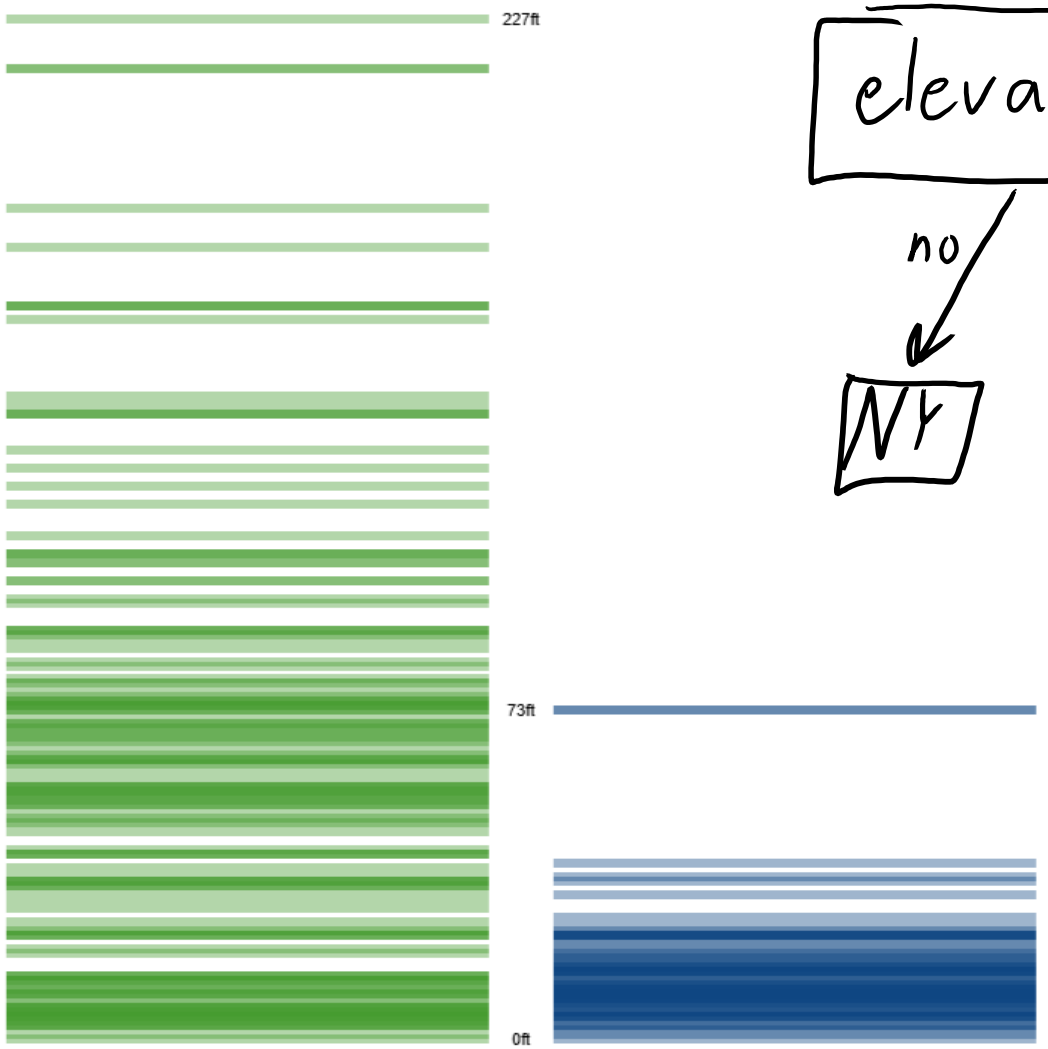
- We are given data from 248 homes.
- For each home/example, we have these features:
  - Elevation.
  - Year.
  - Bathrooms
  - Bedrooms.
  - Price.
  - Square feet.
- Goal is to build a program that predicts SF or NY.

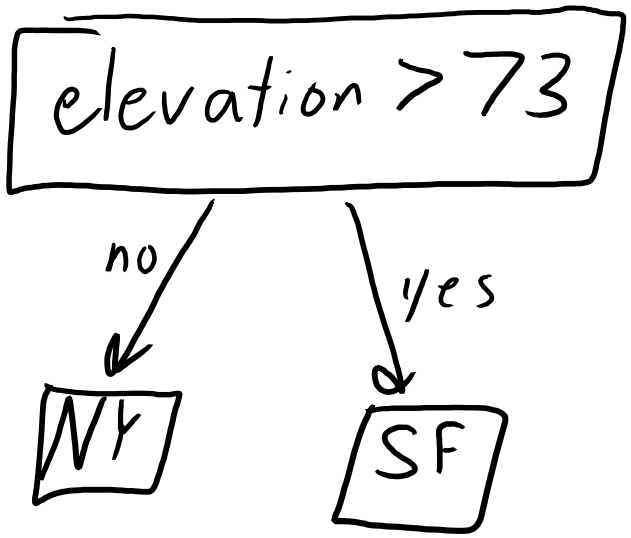
This example and images of it come from:
http://www.r2d3.us/visual-intro-to-machine-learning-part-1

Plotting Elevation

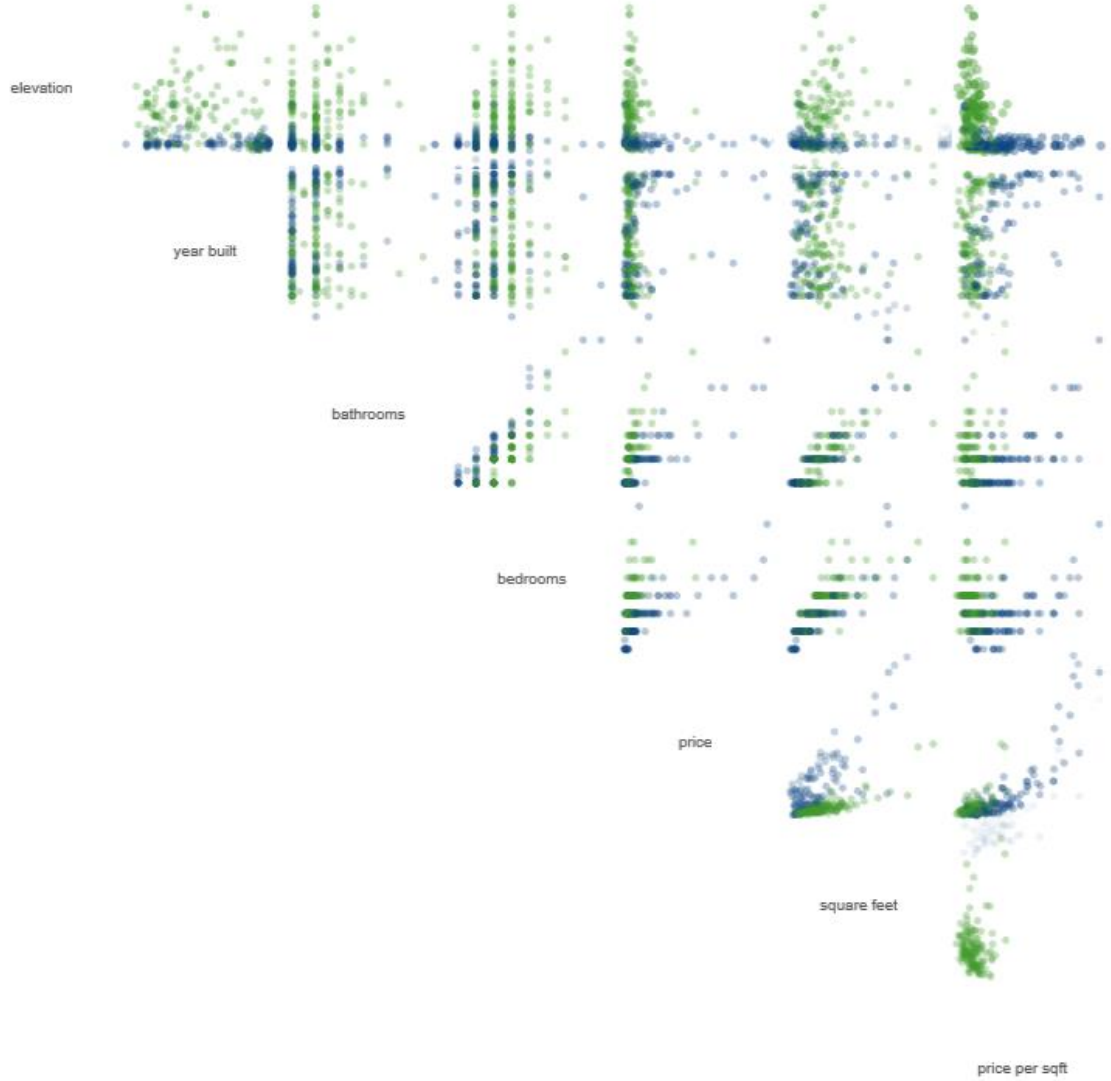


Simple Decision Stump

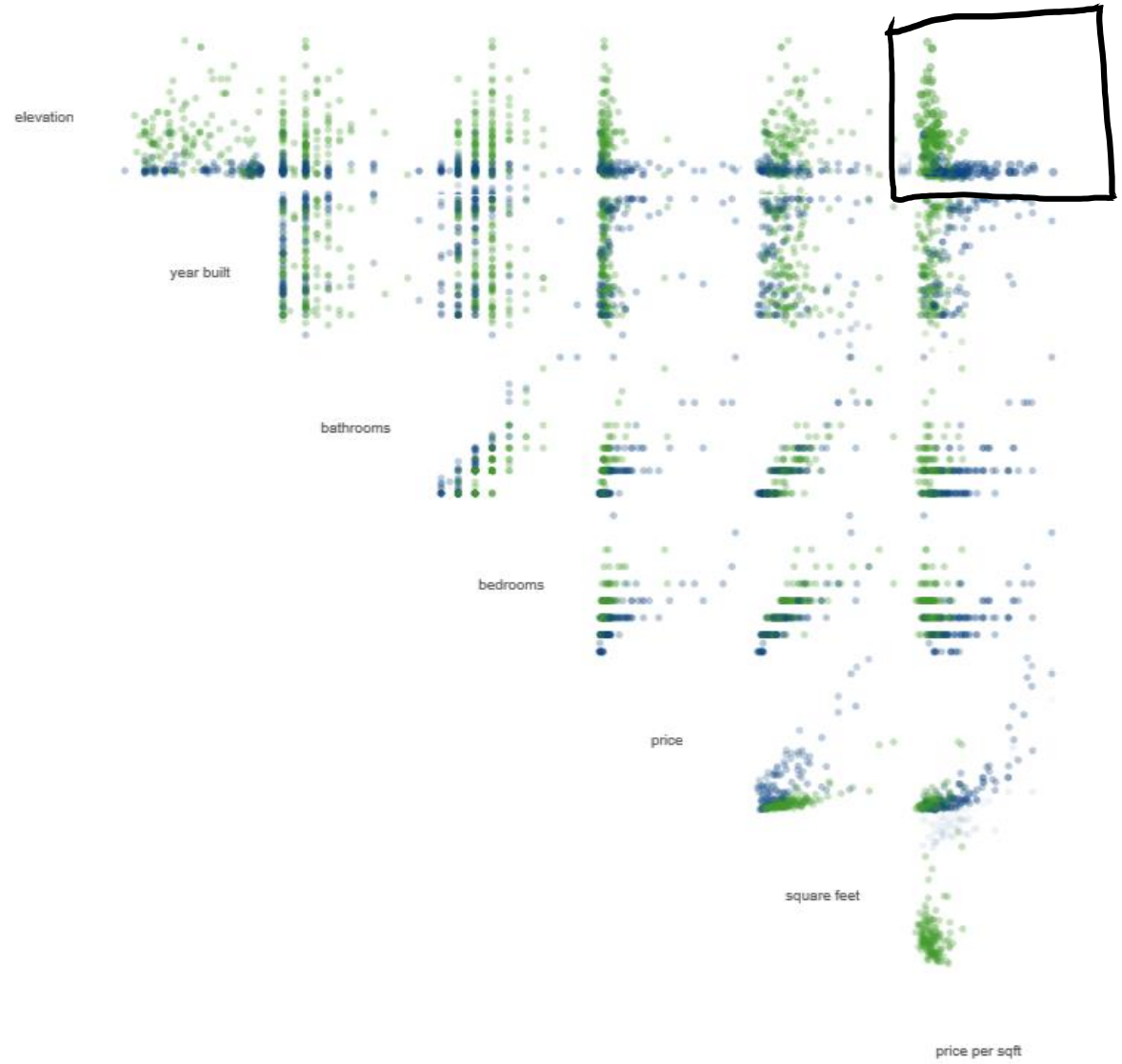




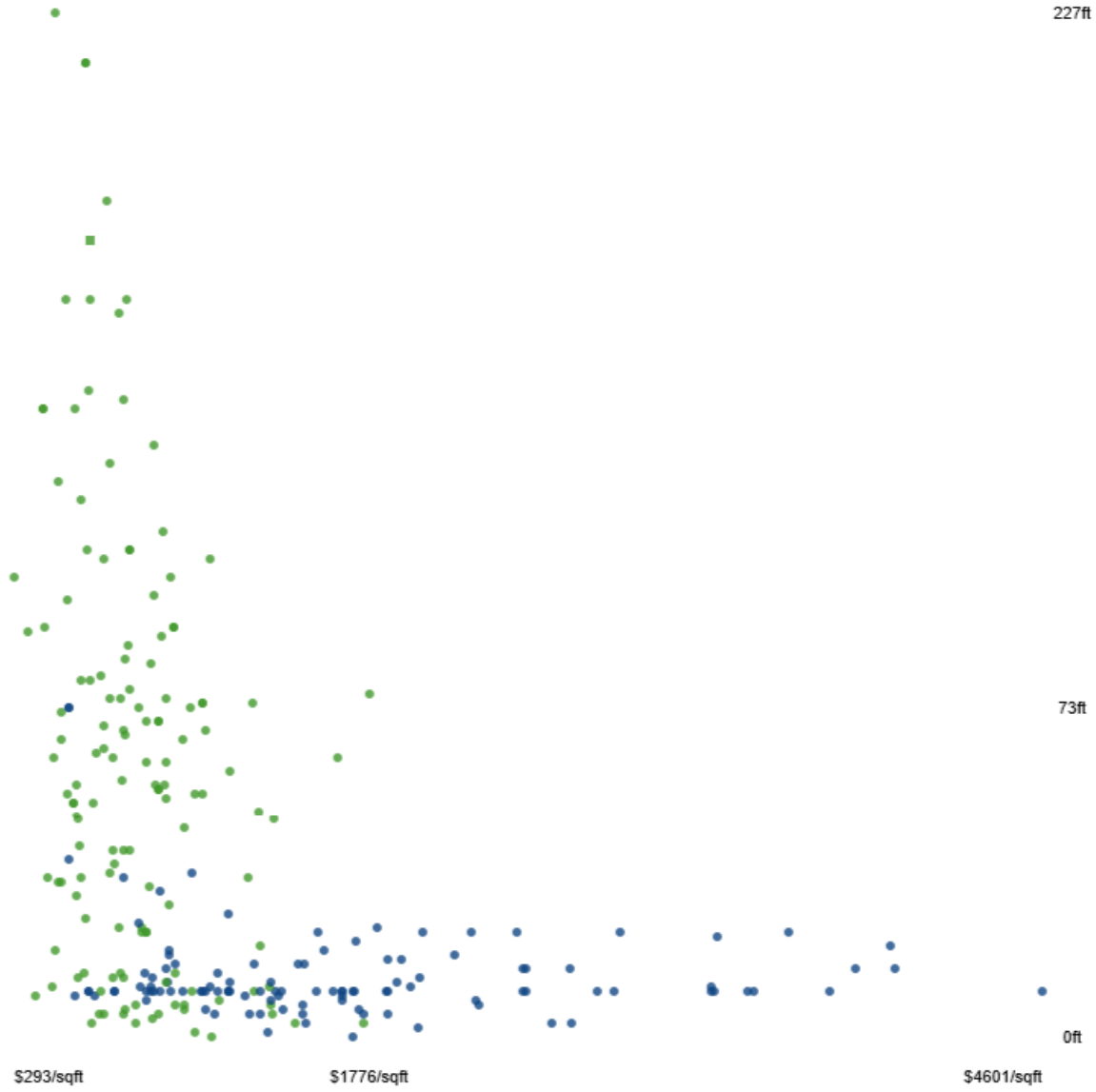
Scatterplot Array



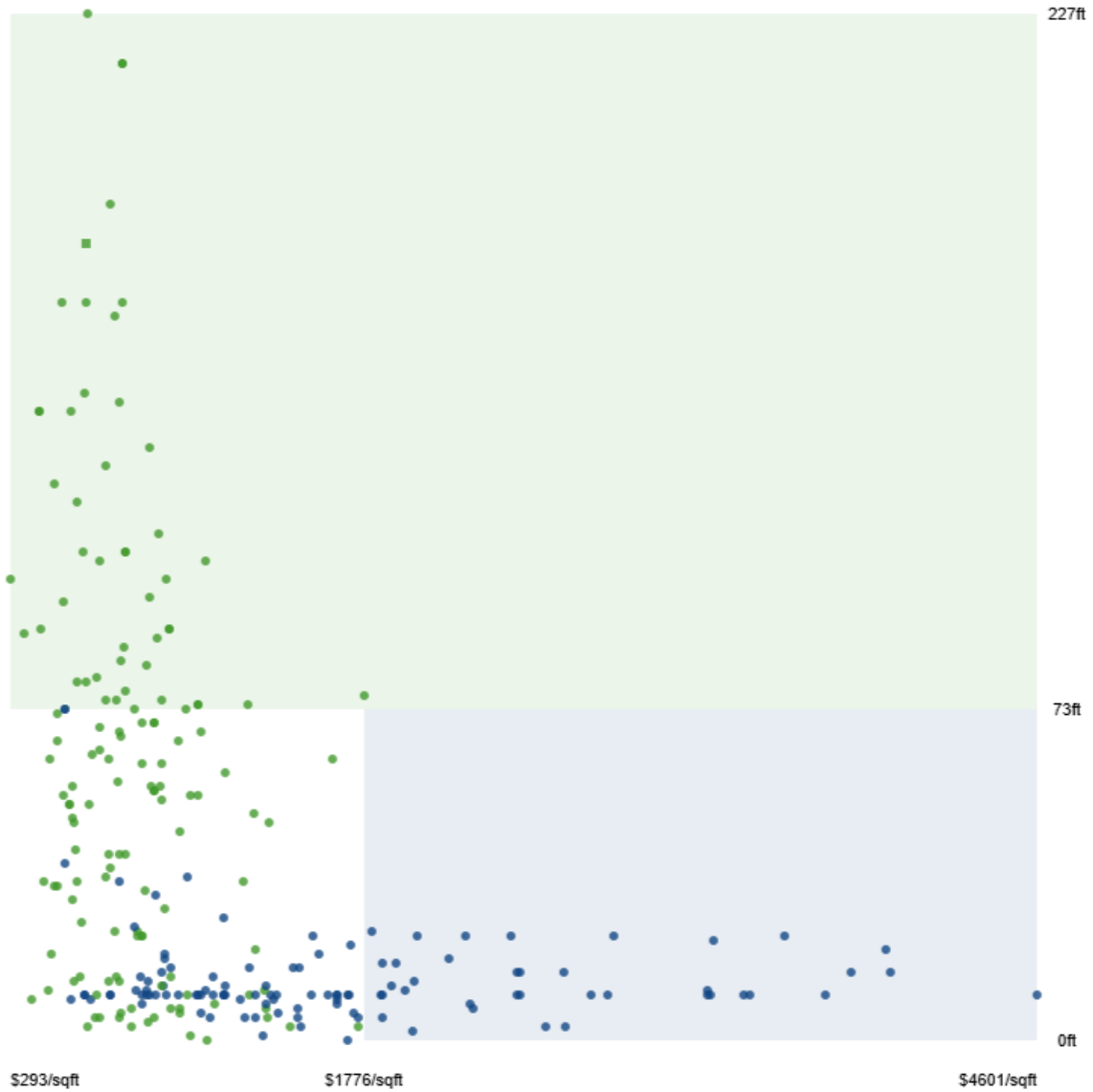
Scatterplot Array



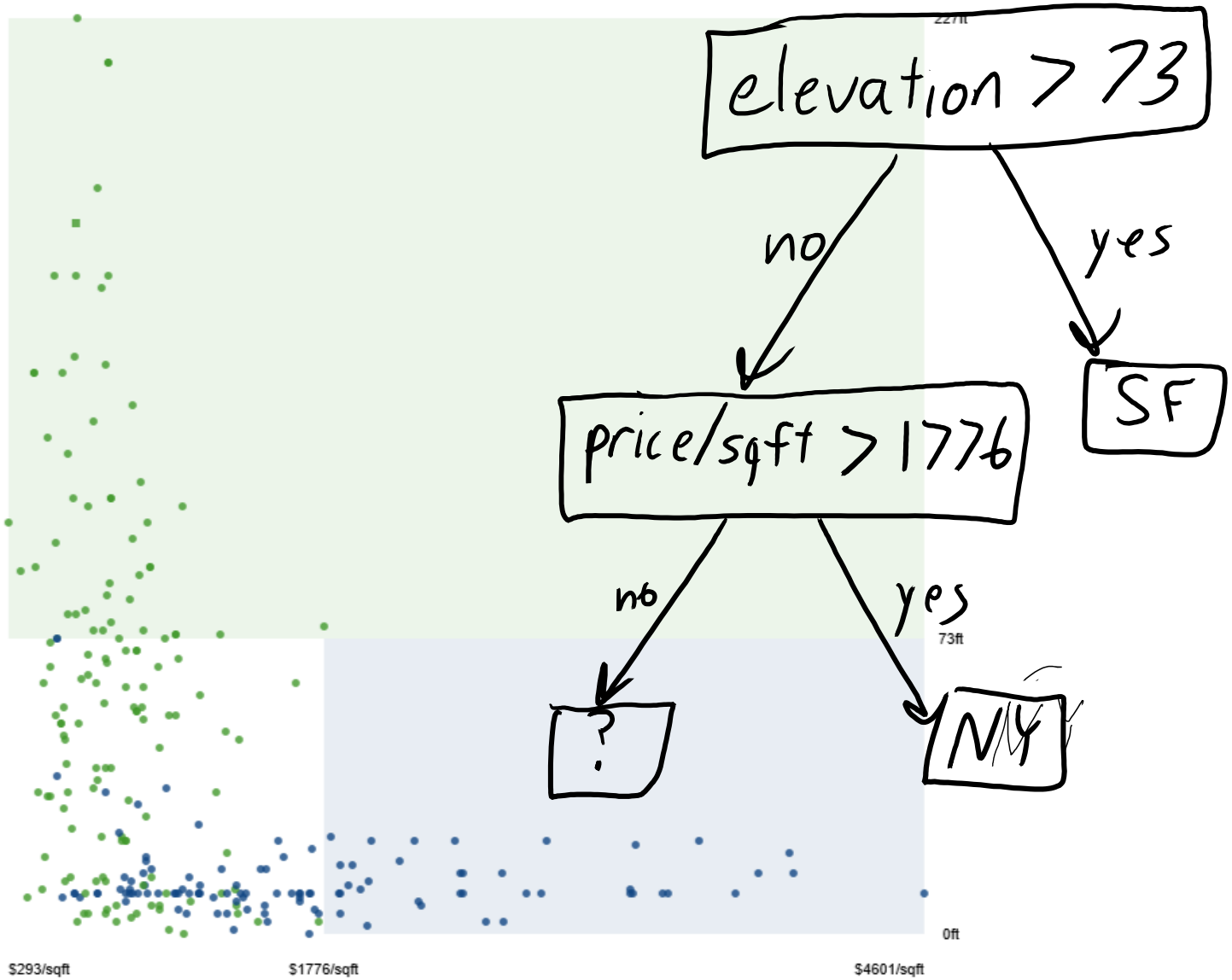
Plotting Elevation and Price/SqFt



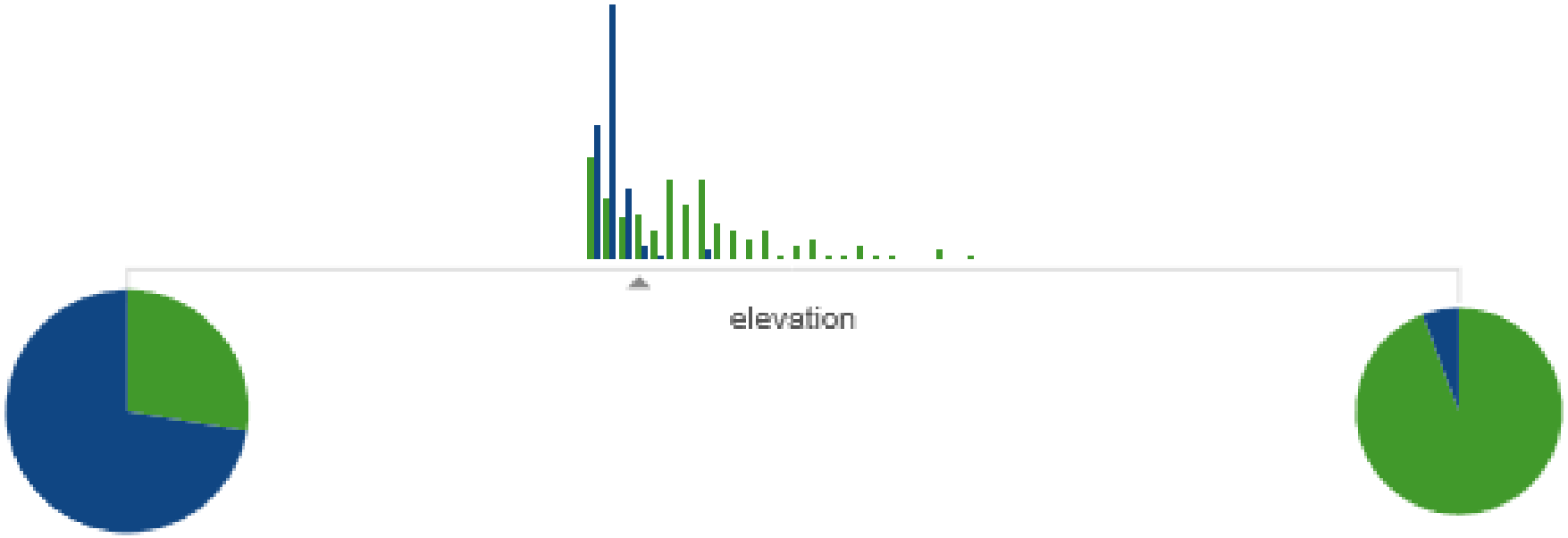
Simple Decision Tree Classification



Simple Decision Tree Classification

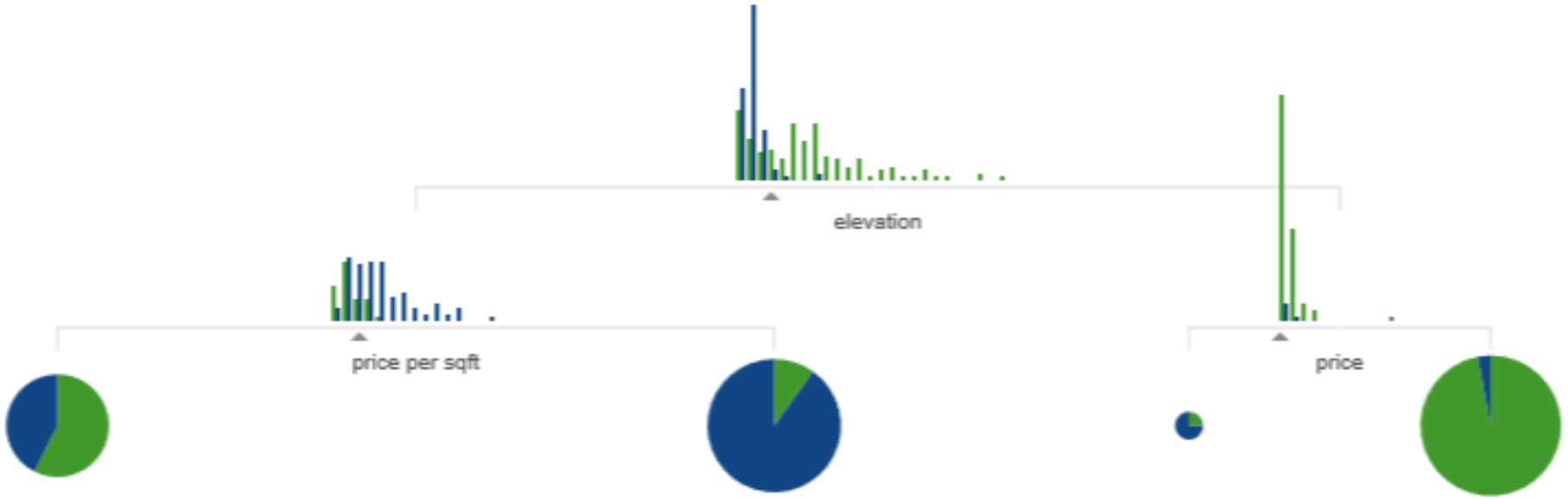


How does the depth affect accuracy?



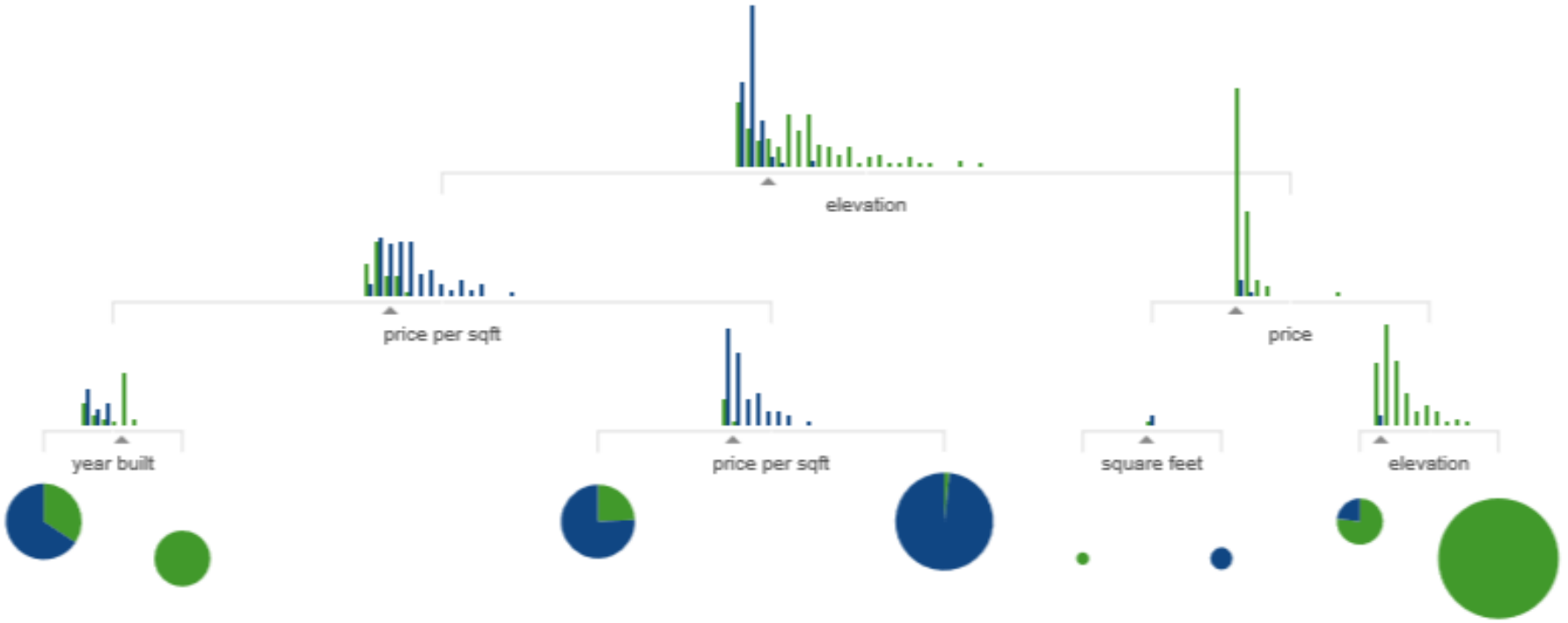
This is a good start (> 75% accuracy).

How does the depth affect accuracy?



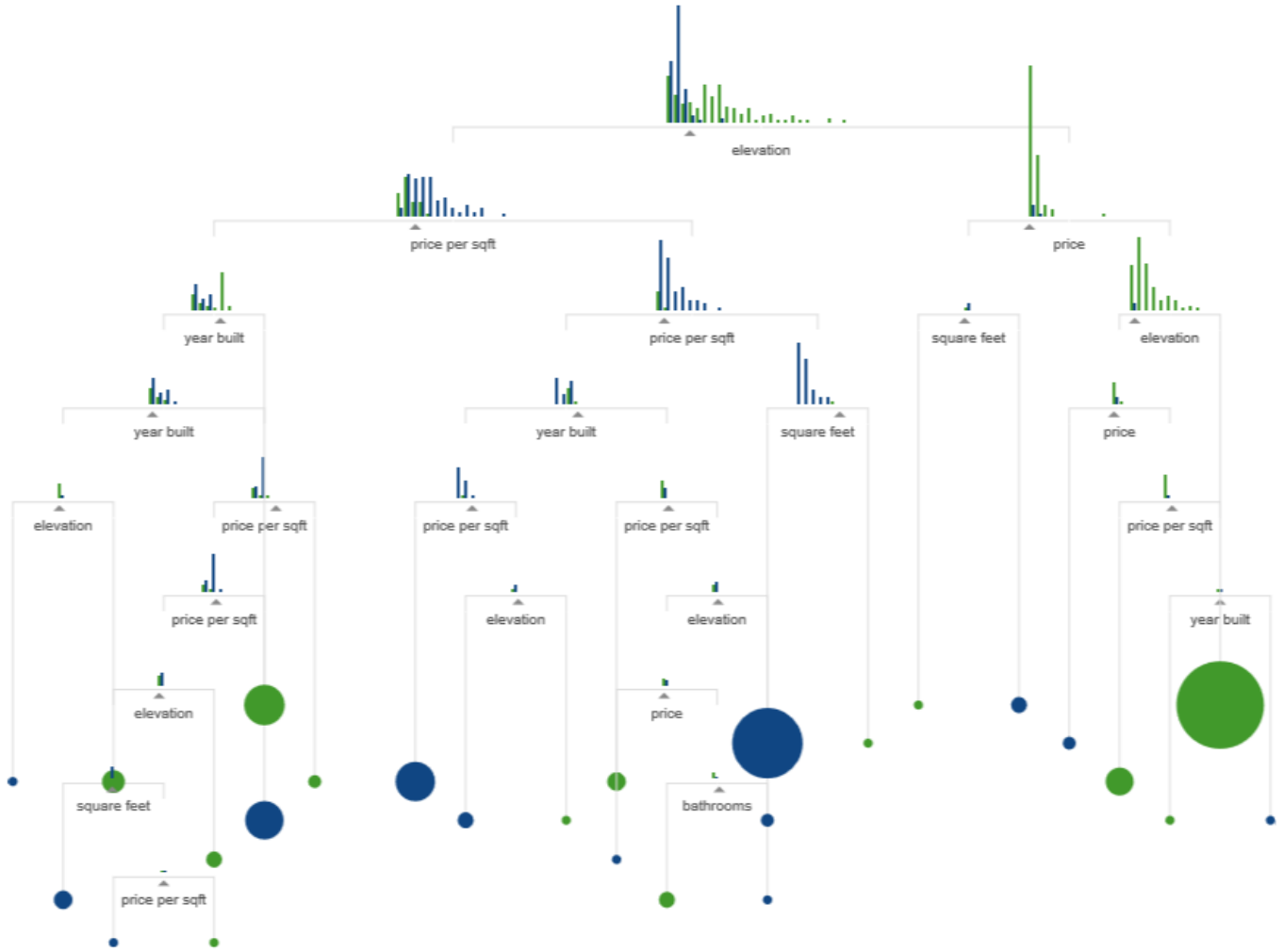
Start splitting the data recursively…

How does the depth affect accuracy?



Accuracy keeps increasing as we add depth.

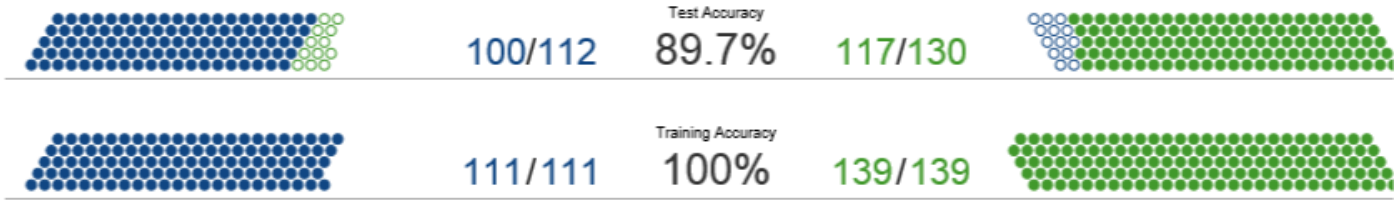
How does the depth affect accuracy?



Eventually, we can perfectly classify all of our data.

Training vs. Testing Error

- With this decision tree, ‘training accuracy’ is 1.
  - It perfectly labels the data we used to make the tree.
- We are now given features for 217 new homes.
- What is the ‘testing accuracy’ on the new data?
  - How does it do on data not used to make the tree?



- Overfitting: lower accuracy on new data.
  - Our rules got too specific to our exact training dataset.
  - Some of the “deep” splits only use a few examples (bad “coupon collecting”).

Supervised Learning Notation

- We are given training data where we know labels:

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

- But there is also testing data we want to label:

$\tilde{X}$=

Egg	Milk	Fish	Wheat	Shellfish	Peanuts	…
0.5	0	1	0.6	2	1	
0	0.7	0	1	0	0	
3	1	0	0.5	0	0	

$\tilde{y}$=

Sick?
?
?
?

Supervised Learning Notation

- Typical supervised learning steps:
  1. Build model based on training data X and y (training phase).
  2. Model makes predictions $\hat{y}$ on test data $\tilde{X}$ (testing phase).

- Instead of training error, consider test error:
  - Are predictions $\hat{y}$ similar to true unseen labels $\tilde{y}$?

Goal of Machine Learning

- In machine learning:
  - What we care about is the test error!
- Midterm analogy:
  - The training error is the practice midterm.
  - The test error is the actual midterm.
  - Goal: do well on actual midterm, not the practice one.
- Memorization vs learning:
  - Can do well on training data by memorizing it.
  - You've only learned if you can do well in new situations.

Golden Rule of Machine Learning

- Even though what we care about is test error:
  - THE TEST DATA CANNOT INFLUENCE THE TRAINING PHASE IN ANY WAY.

- We're measuring test error to see how well we do on new data:
  - If used during training, doesn't measure this.
  - You can start to overfit if you use it during training.
  - Midterm analogy: you are cheating on the test.

Golden Rule of Machine Learning

- Even though what we care about is test error:
  - THE TEST DATA CANNOT INFLUENCE THE TRAINING PHASE IN ANY WAY.

Tom Simonite
June 4, 2015

Why and How Baidu Cheated an Artificial Intelligence Test

Machine learning gets its first cheating scandal.

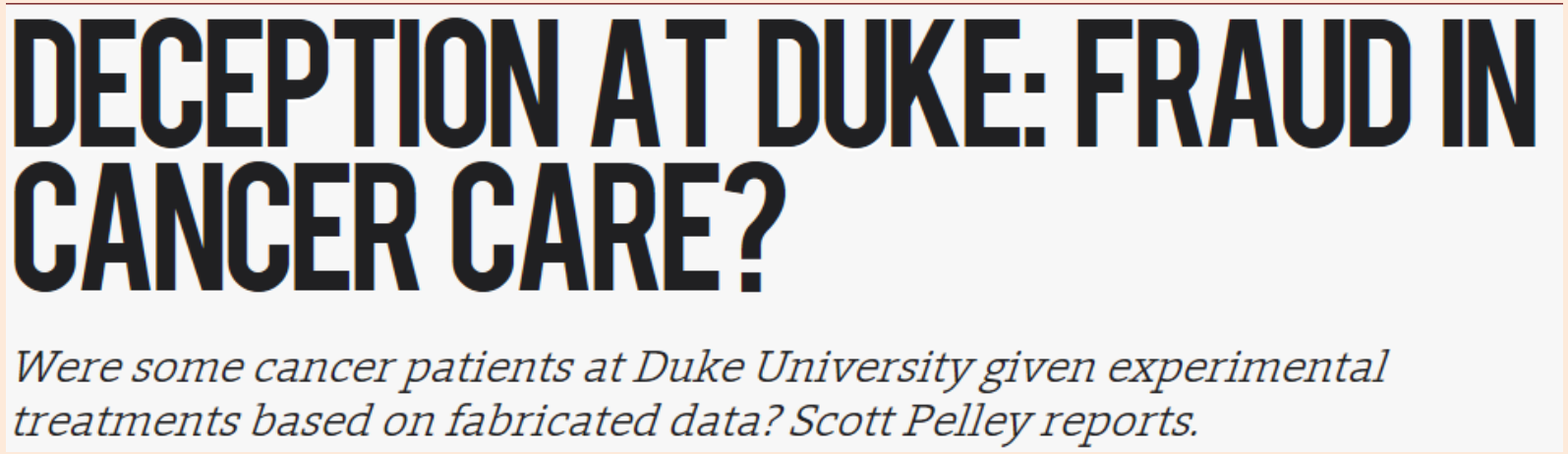
The sport of training software to act intelligently just got its first cheating scandal. Last month Chinese search company Baidu announced that its image recognition software had inched ahead of Google's on a standardized

http://www.technologyreview.com/view/538111/why-and-how-baidu-cheated-an-artificial-intelligence-test/

Golden Rule of Machine Learning

- Even though what we care about is test error:
  - THE TEST DATA CANNOT INFLUENCE THE TRAINING PHASE IN ANY WAY.

- You also shouldn't change the test set to get the result you want.



DECEPTION AT DUKE: FRAUD IN CANCER CARE?

Were some cancer patients at Duke University given experimental treatments based on fabricated data? Scott Pelley reports.

  - http://blogs.sciencemag.org/pipeline/archives/2015/01/14/the_dukepotti_scandal_from_the_inside

https://www.cbsnews.com/news/deception-at-duke-fraud-in-cancer-care/

Digression: Golden Rule and Hypothesis Testing

- Note the golden rule applies to hypothesis testing in scientific studies.
  - Data that you collect can't influence the hypotheses that you test.

- EXTREMELY COMMON and a MAJOR PROBLEM, coming in many forms:
  - Collect more data until you coincidentally get significance level you want.
  - Try different ways to measure performance, choose the one that looks best.
  - Choose a different type of model/hypothesis after looking at the test data.

- If you want to modify your hypotheses, you need to test on new data.
  - Or at least be aware and honest about this issue when reporting results.

Digression: Golden Rule and Hypothesis Testing

- Note the golden rule applies to hypothesis testing in scientific studies.
  - Data that you collect can't influence the hypotheses that you test.

- EXTREMELY COMMON and a MAJOR PROBLEM, coming in many forms:
  - "Replication crisis in Science".
  - "Why Most Published Research Findings are False".
  - "False-Positive Psychology: Undisclosed Flexibility in Data Collection and Analysis Allows Presenting Anything as Significant".
  - "HARKing: Hypothesizing After the Results are Known".
  - "Hack Your Way To Scientific Glory".
  - "Psychology's Replication Crisis Has Made The Field Better" (some solutions)

Is Learning Possible?

- Does training error say anything about test error?
  - In general, NO: Test data might have nothing to do with training data.
  - E.g., "adversary" takes training data and flips all labels.

X =

Egg	Milk	Fish
0	0.7	0
0.3	0.7	1
0.3	0	0

y =

Sick?
1
1
0

$\tilde{X}$ =

Egg	Milk	Fish
0	0.7	0
0.3	0.7	1
0.3	0	0

$\tilde{y}$ =

Sick?
0
0
1

- In order to learn, we need assumptions:
  - The training and test data need to be related in some way.
  - Most common assumption: independent and identically distributed (IID).

IID Assumption

- Training/test data is independent and identically distributed (IID) if:
  - All examples come from the same distribution (identically distributed).
  - The example are sampled independently (order doesn't matter).

Row 1 comes from same distribution as rows 2-3.

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

Row 4 does not depend on values in rows 1-3.

- Examples in terms of cards:
  - Pick a card, put it back in the deck, re-shuffle, repeat. → IID
  - Pick a card, put it back in the deck, repeat.
  - Pick a card, don't put it back, re-shuffle, repeat. } Not IID

IID Assumption and Food Allergy Example

- Is the food allergy data IID?
  - Do all the examples come from the same distribution?
  - Does the order of the examples matter?
- No!
  - Being sick might depend on what you ate yesterday (not independent).
  - Your eating habits might changed over time (not identically distributed).
- What can we do about this?
  - Just ignore that data isn't IID and hope for the best?
  - For each day, maybe add the features from the previous day?
  - Maybe add time as an extra feature?

Learning Theory

- Why does the IID assumption make learning possible?
  - Patterns in training examples are likely to be the same in test examples.
- The IID assumption is rarely true:
  - But it is often a good approximation.
  - There are other possible assumptions.
- Also, we're assuming IID across examples but not across features.

- Learning theory explores how training error is related to test error.
- We'll look at a simple example, using this notation:
  - E_{train} is the error on training data.
  - E_{test} is the error on testing data.

Fundamental Trade-Off

- Start with $E_{test} = E_{test}$, then add and subtract E_{train} on the right:

$$E_{test} = (E_{test} - E_{train}) + E_{train}$$

"test error" "approximation error" "training error"

E_{approx}

- How does this help?
  - If E_{approx} is small, then E_{train} is a good approximation to E_{test}.
- What does E_{approx} ("amount of overfitting") depend on?
  - It tends to get smaller as 'n' gets larger.
  - It tends to grow as model get more "complicated".

Fundamental Trade-Off

- This leads to a fundamental trade-off:
  1. E_{train}: how small you can make the training error.

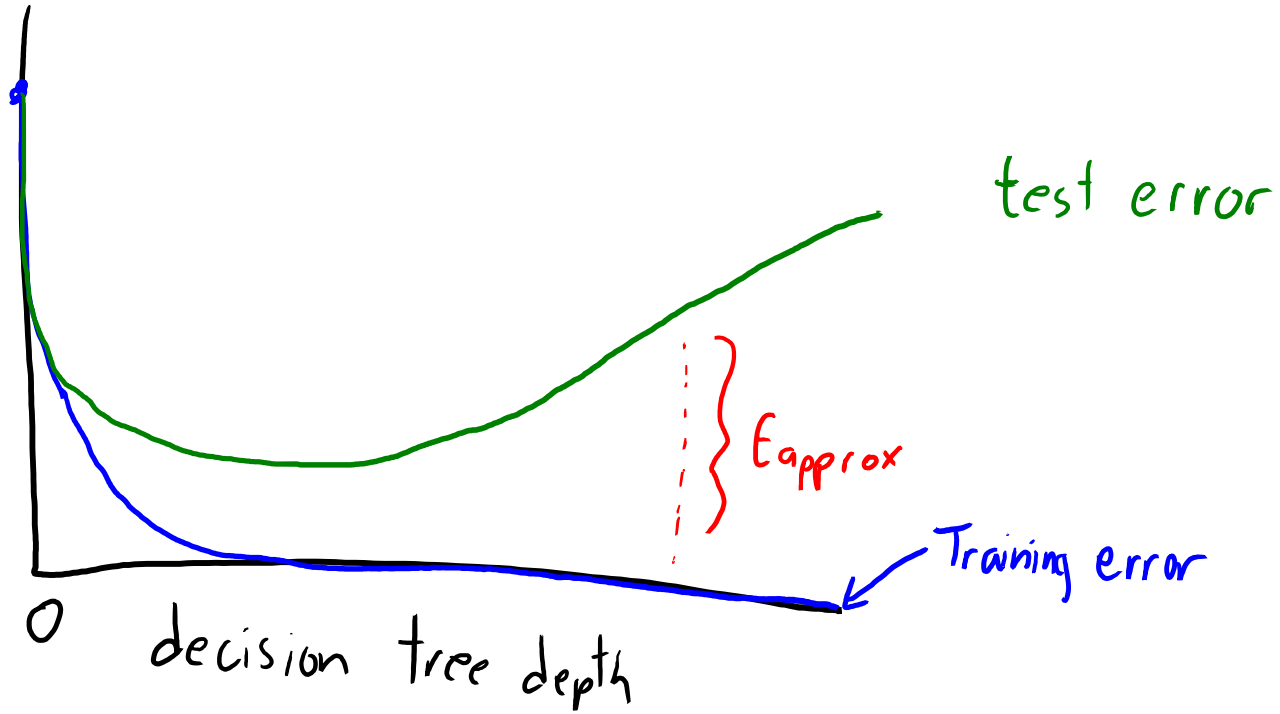
     vs.

  2. E_{approx}: how well training error approximates the test error.

- Simple models (like decision stumps):
  - E_{approx} is low (not very sensitive to training set).
  - But E_{train} might be high.
- Complex models (like deep decision trees):
  - E_{train} can be low.
  - But E_{approx} might be high (very sensitive to training set).

Fundamental Trade-Off

- Training error vs. test error for choosing depth:
  - Training error is high for low depth (underfitting)
  - Training error gets better with depth.
  - Test error initially goes down, but eventually increases (overfitting).

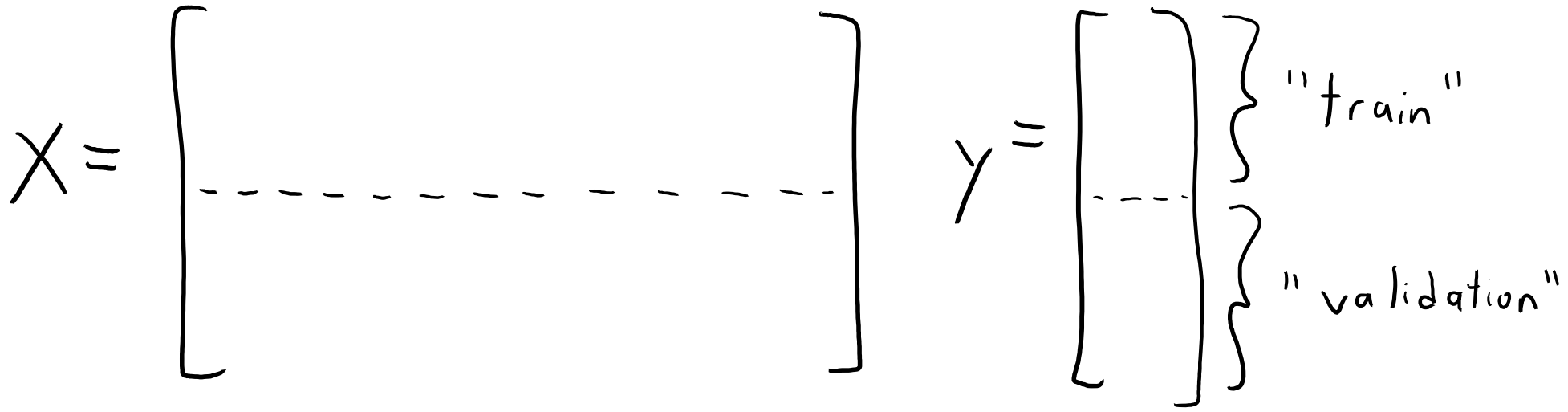


Validation Error

- How do we decide decision tree depth?
- We care about test error.
- But we can't look at test data.
- So what do we do?????

- One answer: Use part of the training data to approximate test error.
- Split training examples into training set and validation set:
  - Train model based on the training data.
  - Test model based on the validation data.

Validation Error



Step 1 is training: model = train(X_{train}, y_{train})

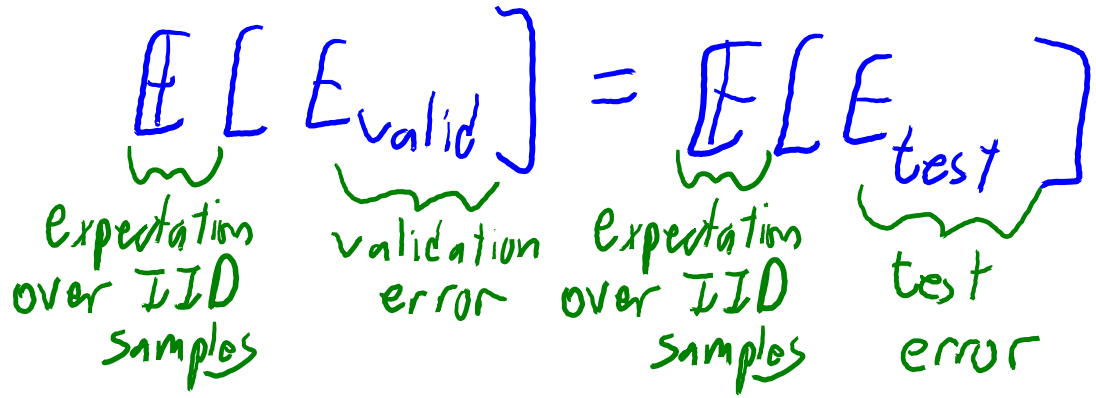
Step 2 is predicting: $\hat{y}$ = predict(model, $X_{validate}$)

Step 3 is validating: error = sum($\hat{y} \neq y_{validate}$)

Note: if examples are ordered, split should be random.

Validation Error

- IID data: validation error is unbiased approximation of test error.



- Midterm analogy:
  - You have 2 practice midterms.
  - You hide one midterm, and spend a lot of time working through the other.
  - You then do the other practice term, to see how well you'll do on the test.

- We typically use validation error to choose "hyper-parameters"…

Notation: Parameters and Hyper-Parameters

- The decision tree rule values are called "parameters".
  - Parameters control how well we fit a dataset.
  - We "train" a model by trying to find the best parameters on training data.

- The decision tree depth is a called a "hyper-parameter".
  - Hyper-parameters control how complex our model is.
  - We can't "train" a hyper-parameter.
    - You can always fit training data better by making the model more complicated.
  - We "validate" a hyper-parameter using a validation score.

- ("Hyper-parameter" is sometimes used for parameters "not fit with data".)

Choosing Hyper-Parameters with Validation Set

- So to choose a good value of depth ("hyper-parameter"), we could:
  - Try a depth-1 decision tree, compute validation error.
  - Try a depth-2 decision tree, compute validation error.
  - Try a depth-3 decision tree, compute validation error.
  - …
  - Try a depth-20 decision tree, compute validation error.
  - Return the depth with the lowest validation error.

- After you choose the hyper-parameter, we usually re-train on the full training set with the chosen hyper-parameter.

Digression: Optimization Bias

- Another name for overfitting is "optimization bias":
  - How biased is an "error" that we optimized over many possibilities?

- Optimization bias of parameter learning:
  - During learning, we could search over tons of different decision trees.
  - So we can get "lucky" and find one with low training error by chance.
    - "Overfitting of the training error".

- Optimization bias of hyper-parameter tuning:
  - Here, we might optimize the validation error over 20 values of "depth".
  - One of the 20 trees might have low validation error by chance.
    - "Overfitting of the validation error".

Digression: Example of Optimization Bias

- Consider a multiple-choice (a,b,c,d) "test" with 10 questions:
  - If you choose answers randomly, expected grade is 25% (no bias).
  - If you fill out two tests randomly and pick the best, expected grade is 33%.
    - Optimization bias of ~8%.
  - If you take the best among 10 random tests, expected grade is ~47%.
  - If you take the best among 100, expected grade is ~62%.
  - If you take the best among 1000, expected grade is ~73%.
  - If you take the best among 10000, expected grade is ~82%.
    - You have so many "chances" that you expect to do well.

- But on new questions the "random choice" accuracy is still 25%.

Factors Affecting Optimization Bias

- If we instead used a 100-question test then:
  - Expected grade from best over 1 randomly-filled test is 25%.
  - Expected grade from best over 2 randomly-filled test is ~27%.
  - Expected grade from best over 10 randomly-filled test is ~32%.
  - Expected grade from best over 100 randomly-filled test is ~36%.
  - Expected grade from best over 1000 randomly-filled test is ~40%.
  - Expected grade from best over 10000 randomly-filled test is ~47%.

- The optimization bias grows with the number of things we try.
  - "Complexity" of the set of models we search over.
- But, optimization bias shrinks quickly with the number of examples.
  - But it's still non-zero and growing if you over-use your validation set!

Summary

- Training error vs. testing error:
  - What we care about in machine learning is the testing error.
- Golden rule of machine learning:
  - The test data cannot influence training the model in any way.
- Independent and identically distributed (IID):
  - One assumption that makes learning possible.
- Fundamental trade-off:
  - Trade-off between getting low training error and having training error approximate test error.
- Validation set:
  - We can save part of our training data to approximate test error.
- Hyper-parameters:
  - Parameters that control model complexity, typically set with a validation set.

- Next time:
  - We discuss the “best” machine learning method.

“test error” vs. “test set error” vs. “validation error”

Chenliang Zhou 8 months ago @Mark
About Q1, wouldn’t the dataset we use to examine our performance be called validation dataset? Mike said that in 340 “testing dataset” refers to those we don’t know.

Lucas Porto 8 months ago I'm now confused about this too. I thought there should be a separate "test set", which you use to measure the performance of your model after training and selection. Selection here meaning hyperparameter tuning with a validation set that not used for training.

Mark Schmidt 8 months ago Unfortunately, there isn't a standard nomenclature for what exactly defines a "test set". But a common convention is this:

1. The "test error" is the expected error over all possible future examples. You can never measure this.

2. We often have a "test set" that we are using to approximate this "test error". So we could say the "test set error" is being used as an approximation of the "test error". If you want this "test set error" to be an unbiased estimate of test error, it should not influence the training in any way. Unfortunately, most people (including your profs) aren't careful about distinguishing "test error" and "test set error".

3. When we tune hyper-parameters, we often use a "validation set" to approximate the "test error". Since we are evaluating the validation error several times, it will have an optimization bias. So it might guide us towards good hyper-parameters (because the bias is typically not that large) but really be used as an unbiased measure of test error.

can't

“test error” vs. “test set error” vs. “validation error”

We are given a huge dataset that we want to make a model from it. We can never know the exact performance of the model for new data that is NOT part of our dataset.

So here is what we can do:

Split the huge dataset into 3 categories:

a) **Training** data: this data is used to train a model
b) **Validation** data: this data is used "**intermediate**" measure the performance of the model we created
c) **Test** data: this data is used as a "**final**" measure of performance of the of the final model that was created

To choose a model do the following:
1. train the model using the **training data**
2. once you have a candidate model, find its performance (i.e validation error) using the **validation data**
3. if you are not satisfied with the performance of the candidate model, find a new model using **training** data and measure its performance using **validation** data. But don't look at your **test** data yet. (i.e do step 1 and 2 again)
4. once you have a model that you are satisfied with (i.e it has low validation error) you can select the model as your **FINAL trained model** meaning that you cannot go back and change the model again.
5. Measure the performance of your **FINAL** model using the **Test** data. You can think of this performance, as the good approximation of the performance of the model on NEW data that the model has never seen.

And the golden rule of ML states that

```
You should NEVER EVER use your test data in order to train a model.
```

If you do so your model will be biased.

Mark Schmidt 8 months ago Great explanation anonymous!

Approximation Error for Selecting Hyper-Parameters

- From the 2019 EasyMarkit AI Hackathon:
  - "We ended up selecting the hyperparameters that gave us the lowest approximation error (gap between train and validation) as opposed to the lowest validation error. This was quite a difficult decision for our team since we were only allowed one submission. However, the model with the lowest validation error had a very high approximation error, which felt too risky, so we went with a model with a slightly higher validation error and much lower approximation error. When the results were announced, the reported test accuracy was within 0.1% of what our model predicted with the validation set."

- This is the type of reasoning you want to do.
  - A high approximation error could indicate low validation error by chance.

“A visual Introduction to machine learning”

- The “housing prices” example is taken from this website:
  - http://www.r2d3.us/visual-intro-to-machine-learning-part-1
- They also have a “Part 2” here:
  - http://www.r2d3.us/visual-intro-to-machine-learning-part-2
- Part 2 covers similar topics to what we covered in this lecture.

Bounding E_{approx}

- Let's assume we have a fixed model 'h' (like a decision tree), and then we collect a training set of 'n' examples.
- What is the probability that the error on this training set (E_{train}), is within some small number ε of the test error (E_{test})?

- From "Hoeffding's inequality" we have:

$$P\left[\left|E_{train}(h) - E_{test}(h)\right| > \varepsilon\right] \leq 2\exp(-2\varepsilon^2 n)$$

- This is great! In this setting the probability that our training error is far from our test error goes down exponentially in terms of the number of samples 'n'.

Bounding E_{approx}

- Unfortunately, the last slide gets it backwards:
  - We usually don't pick a model and then collect a dataset.
  - We usually collect a dataset and then pick the model 'w' based on the data.
- We now picked the model that did best on the data, and Hoeffding's inequality doesn't account for the optimization bias of this procedure.

- One way to get around this is to bound ($E_{test} - E_{train}$) for *all* models in the space of models we are optimizing over.
  - If we bound it for all models, then we bound it for the best model.
  - This gives looser but correct bounds.

Bounding E_{approx}

- If we only optimize over a finite number of events 'k', we can use the "union bound" that for events $\{A_1, A_2, \ldots, A_k\}$ we have:

$$p(A_1 \cup A_2 \cup \cdots \cup A_K) \leq \sum_{i=1}^{k} p(A_i)$$

- Combining Hoeffding's inequality and the union bound gives:

$$p\left(|E_{train}(h_1) - E_{test}(h_1)| > \varepsilon \cup |E_{train}(h_2) - E_{test}(h_2)| > \varepsilon \cup \cdots \cup |E_{train}(h_k) - E_{test}(h_k)| > \varepsilon\right)$$

$$\leq \sum_{i=1}^{k} p\left(|E_{train}(h_i) - E_{test}[h_i]| > \varepsilon\right)$$

$$\leq \sum_{i=1}^{k} 2\exp(-2\varepsilon^2 n)$$

$$\leq 2k\exp(-2\varepsilon^2 n)$$

Bounding E_{approx}

- So, with the optimization bias of setting "h*" to the best 'h' among 'k' models, probability that ($E_{test} - E_{train}$) is bigger than ε satisfies:

$$p(|E_{train}(h^*) - E_{test}(h^*)| > \varepsilon) \leq 2k\exp(-2\varepsilon^2 n)$$

- So optimizing over a few models is ok if we have lots of examples.
- If we try lots of models then ($E_{test} - E_{train}$) could be very large.
- Later in the course we'll be searching over continuous models where k = infinity, so this bound is useless.
- To handle continuous models, one way is via the VC-dimension.
  - Simpler models will have lower VC-dimension.

Refined Fundamental Trade-Off

- Let E_{best} be the irreducible error (lowest possible error for *any* model).
  - For example, irreducible error for predicting coin flips is 0.5.
- Some learning theory results use E_{best} to futher decompose E_{test}:

$$E_{test} = \underbrace{(E_{test} - E_{train})}_{\text{"variance"}} + \underbrace{(E_{train} - E_{best})}_{\text{"bias"}} + \underbrace{E_{best}}_{\text{"noise"}}$$

- This is similar to the bias-variance decomposition:
  - Term 1: measure of variance (how sensitive we are to training data).
  - Term 2: measure of bias (how low can we make the training error).
  - Term 3: measure of noise (how low can any model make test error).

Refined Fundamental Trade-Off

- Decision tree with high depth:
  - Very likely to fit data well, so bias is low.
  - But model changes a lot if you change the data, so variance is high.
- Decision tree with low depth:
  - Less likely to fit data well, so bias is high.
  - But model doesn't change much you change data, so variance is low.

- And degree does not affect irreducible error.
  - Irreducible error comes from the best possible model.

Bias-Variance Decomposition

- You may have seen "bias-variance decomposition" in other classes:
  - Assumes $\tilde{y}_i = \bar{y}_i + \varepsilon$, where ε has mean 0 and variance σ^2.
  - Assumes we have a "learner" that can take 'n' training examples and use these to make predictions $\hat{y}_i$.
- Expected squared test error in this setting is

$$E[(\tilde{y}_i - \hat{y}_i)^2] = E[(\hat{y}_i - \bar{y}_i)]^2 + (E[\hat{y}_i^2] - E[\hat{y}_i]^2) + \sigma^2$$

"test squared error" "bias" "variance" "noise"

  - Where expectations are taken over possible training sets of 'n' examples.
  - Bias is expected error due to having wrong model.
  - Variance is expected error due to sensitivity to the training set.
  - Noise (irreducible error) is the best can hope for given the noise (E_{best}).

Bias-Variance vs. Fundamental Trade-Off

- Both decompositions serve the same purpose:
  - Trying to evaluate how different factors affect test error.

- They both lead to the same 3 conclusions:
  1. Simple models can have high E_{train}/bias, low E_{approx}/variance.
  2. Complex models can have low E_{train}/bias, high E_{approx}/variance.
  3. As you increase 'n', E_{approx}/variance goes down (for fixed complexity).

Bias-Variance vs. Fundamental Trade-Off

- So why focus on fundamental trade-off and not bias-variance?
  - Simplest viewpoint that gives these 3 conclusions.
  - No assumptions like being restricted to squared error.

  - You can measure E_{train} but not E_{approx} (1 known and 1 unknown).
    - If E_{train} is low and you expect E_{approx} to be low, then you are happy.
      - E.g., you fit a very simple model or you used a huge independent validation set.

  - You can't measure bias, variance, or noise (3 unknowns).
    - If E_{train} is low, bias-variance decomposition doesn't say anything about test error.
      - You only have your training set, not distribution over possible datasets.
      - Doesn't say if high E_{test} is due to bias or variance or noise.

Learning Theory

- Bias-variance decomposition is a bit weird compared to our previous decompositions of E_{test}:
  - Bias-variance decomposition considers expectation over *possible training sets*.
  - But doesn't say anything about test error with *your* training set.

- Some keywords if you want to learn about learning theory:
  - Bias-variance decomposition, sample complexity, probably approximately correct (PAC) learning, Vapnik-Chernovenkis (VC) dimension, Rademacher complexity.

- A gentle place to start is the "Learning from Data" book:
  - https://work.caltech.edu/telecourse.html

A Theoretical Answer to "How Much Data?"

- Assume we have a source of IID examples and a fixed class of parametric models.
  - Like "all depth-5 decision trees".
- Under some nasty assumptions, with 'n' training examples it holds that:

  E[test error of best model on training set] – (best test error in class) = O(1/n).

- You rarely know the constant factor, but this gives some guidelines:
  - Adding more data helps more on small datasets than on large datasets.
    - Going from 10 training examples to 20, difference with best possible error gets cut in half.
      - If the best possible error is 15% you might go from 20% to 17.5% (this does **not** mean 20% to 10%).
    - Going from 110 training examples to 120, error only goes down by ~10%.
    - Going from 1M training examples to 1M+10, you won't notice a change.
  - Doubling the data size cuts the error in half:
    - Going from 1M training to 2M training examples, error gets cut in half.
    - If you double the data size and your test error doesn't improve, more data might not help.

Can you test the IID assumption?

- In general, testing the IID assumption is not easy.
  - Usually, you need background knowledge to decide if it's reasonable.

- Some tests do exist, like shuffling the order of data and then measuring if some basic statistics agree.
  - It's reasonable to check if summary statistics of train and test data agree.
    - If not, your trained model may not be so useful.

- Some discussion here:
  - https://stats.stackexchange.com/questions/28715/test-for-iid-sampling

CPSC 340 Notation Guide

September 6, 2020

A guide to notation used in the course. Let me know if things are missing from this document that are not obvious.

Part 1: Supervised Learning

Throughout the course, we use n as the number of training examples and d as the number of the features. We use i when indexing a training example and j when indexing a feature.[1] We use x_{ij} as feature j in training example i, and we use y_i as the label of example i (so there are n values $y_1, y_2, \ldots, y_n$). We use y as a list of the n class labels, containing the label y_i in position i. We use x_i as the list of all features for example i, so x_i has d elements $x_{i1}, x_{i2}, \ldots, x_{id}$ (and there are n lists $x_1, x_2, \ldots, x_n$). We use X as an $n \times d$ matrix containing all the features, so x_{ij} is element (i, j) of X and x_i gives the elements of row i of X. We use x^j to refer to all elements of column j, which is the list of values of feature j across all the n training examples.

Throughout the course, we use t as the number of test examples, and $\tilde{X}$ refers to a $t \times d$ matrix containing the test features. The notation $\tilde{x}_i$ refers to the features of test example i, whle $\tilde{x}_{ij}$ refers to feature j in test example i. We use $\tilde{y}$ as the true labels of the test examples, and $\tilde{y}_i$ as the label of test example i. We use $\hat{y}_i$ as the prediction of a model on example i, whether the prediction is made on training dataor validation or test data (it should be obvious or not relevant from context).

When discussing validation sets, X_{train} and y_{train} are used as the subsets that we train on, while X_{validate} and y_{validate} are used as the subsets that we validate on. We use E to denote a generic prediction error, and usually this is followed with a subscript. For example, E_{train} is the training error, E_{test} is the test error, E_{approx} is the approximation error.

We use c as a class label, and occasionally use n_c as the number of training examples in class c. We use the letter k generically throughout the course as something we count, and ϵ as a generic number that we want to be small.

Some method-specific notations used in this section:

- We use t as a particular decision stump threshold, and k as the number of thresholds.
- $p(y_i = \text{“}spam''|x_i)$ is used for the probability that the label y_i takes the value “spam” given that the features are x_i.
- $p(y_i|x_i)$ is used for the probability that the label is y_i given that the features are x_i (for example, y_i could be “spam” or “not spam” but without specifying a particular value).
- In the naive Bayes section, we're a little sloppy in that we use the same notation for the MLE on the training data and the true population value.

[1] When talking about two training examples, we sometimes use j as the index of the second training example.

- Naive Bayes uses n_{cjk} as the number of times that feature j is equal to k and the class label is c.
- Naive Bayes uses $p(\text{hello}|\text{spam})$ as short for $p(x_{ij} = \text{“hello”} \mid y_i = \text{“spam”})$.
- Decision theory slides use $\text{cost}(\hat{y}_i, \tilde{y}_i)$ as the cost of prediction $\hat{y}_i$ when the true label is $\tilde{y}_i$.
- In the norm slides we use r as a generic vector.
- We use $\|r\|_2$ as the L2-norm (square root of sum of squares of the elements in the vector), $\|r\|_1$ as the L1-norm (sum of absolute values), and $\|r\|_\infty$ as the L∞-norm (max of absolute values). If the number is omitted, as in $\|w\|$, it refers to the L2-norm.

Part 2: Unsupervised Learning

We use $\hat{y}_i$ as the cluster predicted for example i and $\hat{y}$ as the set of predicted clusters for all n training examples. We use C as the set of incides of examples assigned to cluster c.

We use W as a k by d matrix where row c contains mean c. We use w_c as mean c, $w_{\hat{y}_i}$ to refer to the mean of the cluster of example i, and w_{cj} to refer to feature j in mean c. We use w^j as column j of the matrix W. We use $\hat{X}$ as predicted values of the matrix X, and similarly $\hat{x}_i$ are predicted values of x_i and $\hat{x}_{ij}$ are predicted values of x_{ij}.

We use μ as the mean of the data (with μ_j being the mean for feature j if we have more than one feature) and σ as the standard deviation (with σ_j being the standard deviation for feature j if we have more than one feature).

Part 3: Linear models

In this section we start treating x_i and y_i as vectors, so we now have to be careful about whether vectors are row-vectors or column-vectors. Our default choice is that everything is a column-vector, so each x_i is a $d \times 1$ vector and y is an $n \times 1$ vector. Since x_i is now a column-vector, we need to be careful to define row i of X as x_i^T (instead of just x_i).

We use w as the $d \times 1$ vector of regression weights. We normally index into w using w_j. We sometimes add a y-intercept (“bias”) variable and use w_0 to denote this variable (in some settings later in the course β is used instead of w_0).

We use $\nabla f(w)$ to denote the gradient of a function f with respect to w. Assuming w has length d, this is a $d \times 1$ vector where position j contains the partial derivative of f with respect to w_j. We use r as the vector of “residuals”, $r = Xw - y$. An individual element i of r would be $r_i = w^T x_i - y_i$.

Gradient descent uses w^t as the parameter vector on iteration t (so t has a separate meaning than “number of test examples” here). The distinction between w^t (iteration t of gradient descent) and w^j (column j of matrix W) should be clear from the context. We use α^t as the step size on iteration t. We use w^* as a minimum of $f(w)$. Stochastic gradient uses f_i to refer to the loss function on example i.

We use Z as an $n \times k$ matrix of features obtained under a change of basis, and z_i as the list of k features in the new basis for example i. When we do linear regression under a change of basis, we use v as the $k \times 1$ vector of parameters (instead of the usual $d \times 1$ vector w). We use $\tilde{Z}$ as the transformation of test data $\tilde{X}$.

We use λ as the (scalar) regularization parameter. It is assumed to be non-negative (and will almost always be positive).

We use $\text{sign}(\alpha)$ as a function that return $+1$ if α is positive and -1 if α is negative.

Multi-class classification uses the same matrix W as we used for k-means, and we use w_{y_i} as the w_c value for the true label y_i.

We use $h(z_i)$ as the sigmoid function applied element-wise to a vector z_i.

Some method-specific notation used in this section:

- p is used as the degree of the polynomial in the polynomial basis, and we sometimes use Z_p when we want to specify specifically that we've used a degree-p basis.
- We use K as the $n \times n$ Gram matrix, containin $z_i^T z_j$ in position (i, j). We use $\tilde{K}$ as the $t \times n$ matrix containing $\tilde{z}_i^T z_j$ in position (i, j). We use u as the $n \times 1$ parameter vector when doing kernel methods for lienar models.The kernel function is written as $k(x_i, x_j)$.
- When introducing MLE/MAP, we use D as generic data (indexed by D_i if it splits into IID training examples), w as generic parameters, and $\hat{w}$ as the predicted MLE or MAP value of w.

Part 4: Latent-Factor Models

Linear latent-factor models use the approximation $X \approx ZW$, where we use the same notation for Z and W as above: Z is $n \times k$ with z_i^T as the rows and z_{ic} as individual elements, W is $k \times d$ with w_c^T as the rows and w^j as the columns and w_{cj} as the individual elements. To avoid expressions like $(w^j)^T z_i$, for inner products in this section we sometimes use notation $\langle w, x \rangle$ to represent the inner product $w^T x$. We use $\|X\|_F$ as the Frobenius norm (square root of sum of elements squared).

Part 5: Neural Networks

This section continues using the same notation, but we now use $W^{(l)}$ and $Z^{(l)}$ as the values in layer l. We also use w_{c0} as the bias on hidden unit c, and m as the number of layers.

When we introduce convolutions we use x as signal, w as a filter, and z as the output of the filter.

Notes on Probability

Mark Schmidt

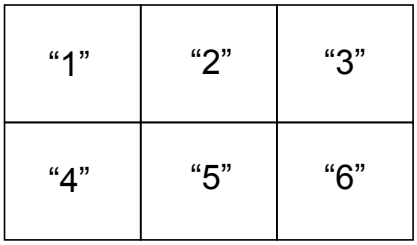
March 25, 2020

1 Probabilites

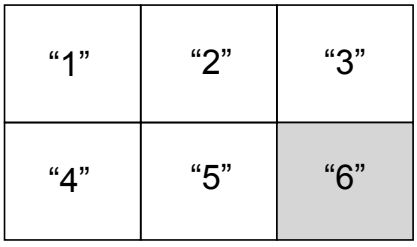
Consider an event A that may or may not happen. For example, if we roll a dice then we may or may not roll a 6. We use the notation $p(A)$ to denote the **probability** of event A happening, which is the likeliness that the event A will actually happen. Probabilities map from events A to a number between 0 and 1,

$$0 \leq p(A) \leq 1,$$

where a value of 0 means "definitely will not happen", a value of 0.5 means that it happens half of the time, and a value of 1 means "definitely will happen". It is helpful to think of probabilities as areas that divide up a geometric object. For example, we can represent the dice example with the following diagram:



We have set up this figure so that the area associated with each event is proportional to its probability. In this case, each possible value of the dice takes up 1/6 of the area, so we have that $p(6) = 1/6$.



We can use $\neg A$ to represent the event that 'A does not happen', and its probability is given by

$$p(\neg A) = 1 - p(A).$$

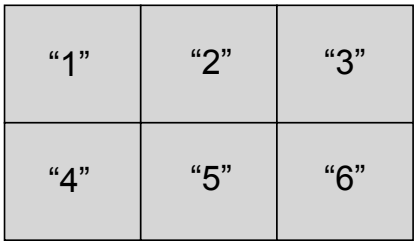
Thus, the probability of *not* rolling a 6 is given by $1 - 1/6 = 5/6$. From the area figure, we see that all the events where rolling a 6 do not happen correspond to 5/6 of the total area.

2 Random Variables

A **random variable** X is a variable that takes different values with certain probabilities. We can then consider probabilities of events involving the random variable, such as the event that $X = x$ for a specific value x. We usually use the notation $p(X = x)$ to denote the probability of the even that the random variable X takes the value x. In the dice example, X could be the value that we roll, and in that case we have $p(X = 6) = 1/6$. Often we will simply write $p(x)$ instead of $p(X = x)$, since the random variable is usually obvious from the context. Let's use $\mathcal{X}$ as the set of all possible values that the random variable X might take. In the dice example, this would be the set $\{1, 2, 3, 4, 5, 6\}$. Because the random variable must take some value, we have that the probabilities over all values must sum to one,

$$\sum_{x \in \mathcal{X}} p(x) = 1.$$

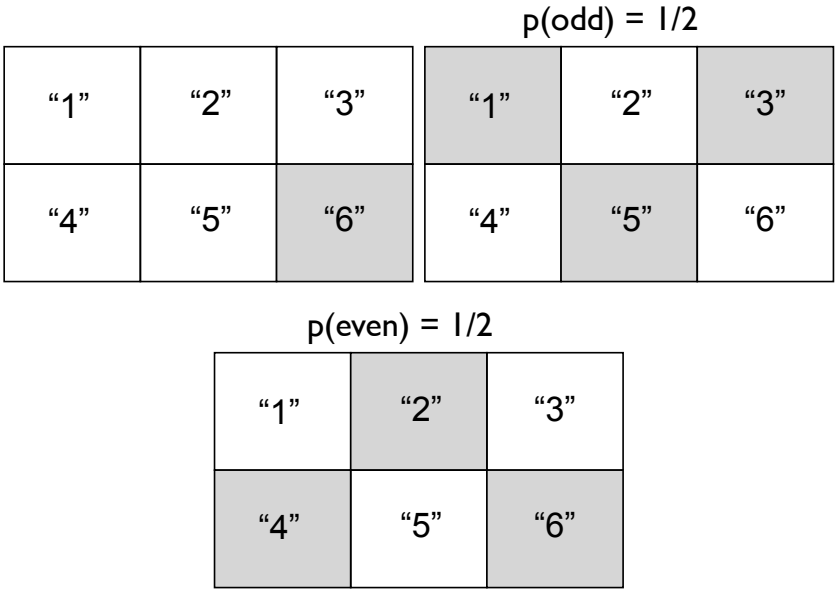
Geometrically, this just means that if we consider all events, that this includes the entire probability space:



In this note, we'll assume that random variables can only take a finite number of possible values. For continuous random variables, we replace sums like these with integrals.

3 Joint Probability

We are often interested in probabilities involving more than one event. For example, if we have two possible events A and B, we might want to know the probability that *both* of them happen. We use the notation $p(A, B)$ to denote the probability of both A and B happening, and we call this the **joint probability**. In terms of areas, this probability is given by the *intersection* of the areas of the two events. For example, consider the probability that we roll a 6 and we roll an odd number, $p(6, \text{odd})$. This probability is zero since the areas where this is true do not intersect.



On the other hand, $p(6, \text{even}) = 1/6$ since the intersection of rolling a 6 with rolling an even number is simply the area associated with rolling a 6. Note that the order of the arguments is not important, so $p(A, B) = p(B, A)$.

An important identity is the **marginalization rule**. This rule says that if we sum the joint probability $p(A, X = x)$ over all possible values x of a random variable X, then we obtain the probability of the event A,

$$p(A) = \sum_{x \in \mathcal{X}} p(A, X = x). \tag{1}$$

For example, the probability of rolling an even number is given by

$$p(\text{even}) = \sum_{i=1}^{6} p(i, \text{even}) = 0 + 1/6 + 0 + 1/6 + 0 + 1/6 = 1/2,$$

which corresponds to adding up all areas where the number is even. If we apply this marginalization rule twice, then we see that the joint probability summed over all values must be equal to one,

$$\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(X = x, Y = y) = \sum_{x \in \mathcal{X}} p(X = x) = 1.$$

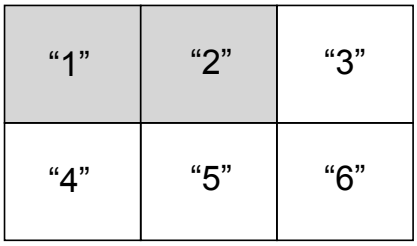
4 Union of Events

Instead of considering the probability of events A and B both occurring, we might instead be interested in the probability of *at least one* of them occurring. This is denoted by $p(A \cup B)$, and in terms of areas corresponds to the union of the areas associated with A and B. This union is given by

$$p(A \cup B) = p(A) + p(B) - p(A, B),$$

where the last term subtracts the common area that is counted in both $p(A)$ and $p(B)$. For example, the probability of rolling a 1 or a 2 is given by

$$p(1 \cup 2) = p(1) + p(2) - p(1, 2) = 1/6 + 1/6 - 0 = 1/3,$$



Simiarly, the probability of rolling a 1 or an odd number is given by

$$p(1 \cup \text{odd}) = p(1) + p(\text{odd}) - p(1, \text{odd}) = 1/6 + 1/2 - 1/6 = 1/2.$$

5 Conditional Probability

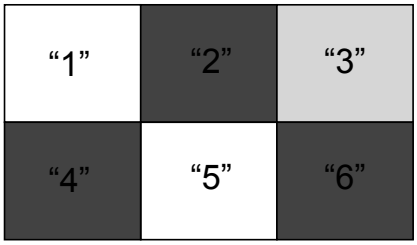
We are often interested in the probability of an event A, *given that we know an event B occurred*. This is called the **conditional probability** and it is denoted by $p(A \mid B)$. Viewed from the perspective of areas, this is the area of A restricted to the region where B happened, divided by the total area taken up by B. Mathematically, this gives

$$p(A \mid B) = \frac{p(A, B)}{p(B)}, \tag{2}$$

where we have $p(B) \neq 0$ since it happened. For example, the probability of rolling a 3 given that you rolled an odd number is given by

$$p(3 \mid \text{odd}) = \frac{p(3, \text{odd})}{p(\text{odd})} = \frac{1/6}{1/2} = \frac{1}{3}.$$

Geometrically, we remove the area associated with numbers that are not odd, and compute the area of where the event happened divided by the total area that is left:



Observe that conditional probabilities sum up to one when we sum over the left variable,

$$\sum_{x \in \mathcal{X}} p(x \mid B) = \sum_{x \in \mathcal{X}} \frac{p(x, B)}{p(B)} = \frac{1}{p(B)} \sum_{x \in \mathcal{X}} p(x, B) = \frac{p(B)}{p(B)} = 1,$$

where we have used the marginalization rule (1). If we sum over the conditioning variable B it does not need to sum up to one,

$$\sum_{x \in \mathcal{X}} p(A \mid x) \neq 1,$$

in general.

6 Product Rule and Bayes Rule

By re-arranging the conditional probability inequality, we obtain the **product rule**,

$$p(A, B) = p(A \mid B)p(B),$$

and similarly

$$p(A, B) = p(B \mid A)p(A).$$

This lets us express joint probabilities (which can be hard to deal with) in terms of conditional probabilities (which are often easier to deal with). It also gives a variation on the marginalization rule (1),

$$p(A) = \sum_{x \in \mathcal{X}} p(A, X = x) = \sum_{x \in \mathcal{X}} p(A \mid X = x)p(X = x).$$

By applying the product rule in the definition of conditional probability (2), we obtain **Bayes rule**,

$$p(A \mid B) = \frac{p(A, B)}{p(B)} = \frac{p(B \mid A)p(A)}{p(B)}.$$

This lets us express the conditional probability of A given B in terms of the reverse conditional probability (of B given A). We sometimes also write Bayes rule using the notation

$$p(A \mid B) \propto p(B \mid A)p(A),$$

where the '$\propto$' sign means that the values are equal up to a constant value that makes the conditional probabilities sum up to one over all values of A. Another form of Bayes rule that you often see comes from applying the marginalization rule (1) and then the product rule to $p(B)$,

$$p(x \mid B) = \frac{p(B \mid x)p(x)}{p(B)} = \frac{p(B \mid x)p(x)}{\sum_{x \in \mathcal{X}} p(B, x)} = \frac{p(B \mid x)p(x)}{\sum_{x \in \mathcal{X}} p(B \mid x)p(x)}.$$

7 Conditional Probabilities with More Than 2 Variables

We can also define conditional probabilities involving more than two events. We use the notation $p(A, B \mid C)$ to denote the probability of both A and B happening, given that C happened,

$$p(A, B \mid C) = \frac{p(A, B, C)}{p(C)}.$$

We similarly use the notation $p(A \mid B, C)$ to denote the probability that A happens, given that both B and C happened,

$$p(A \mid B, C) = \frac{p(A, B, C)}{p(B, C)}.$$

8 Conditional Probability Identities

If we use C to denote the event that we are conditioning on, then if we keep C the right side of the conditioning bar then all of the identities above generalize to conditional probabilities. Conceptually, we're just redefining the total probability area as the area where C happened. For example, the marginalization rule (1) is changed to

$$\sum_{x \in \mathcal{X}} p(A, x \mid C) = p(A \mid C),$$

the union of events is change to

$$p(A \cup B \mid C) = p(A \mid C) + p(B \mid C) - p(A, B \mid C),$$

the product rule is changed to

$$p(A, B \mid C) = p(A \mid B, C)p(B \mid C),$$

Bayes rule is changed to

$$p(A \mid B, C) = \frac{p(B \mid A, C)p(A \mid C)}{p(B \mid C)},$$

and so on.

9 Independence and Conditional Independence

We say that two events are **independent** if their joint probability equals the product of their individual probabilities,

$$p(A, B) = p(A)p(B).$$

In this case we use the notation $A \perp B$. Two random variables are independent if this is true for all values that the random variables can take.

By using the product rule, we see that two variables are independent iff

$$p(A)p(B) = p(A, B) = p(A \mid B)p(B),$$

or equivalently that

$$p(A \mid B) = p(A).$$

This means that knowing that B happened tells us nothing about the probability of A happening, and vice versa.

A generalization of independence is **conditional independence**, where we consider independence given that we know a third event C occurred,

$$p(A, B \mid C) = p(A \mid C)p(B \mid C),$$

and in this case we use the notation $A \perp B \mid C$. Conditional independence can be much weaker than marginal independence, and we often make use of it to model high-dimensional probability distributions.

10 Expectation

If we have a random variable X that can take values $x \in \mathcal{X}$, we define the **expectation** of X or $\mathbb{E}[X]$ by

$$\mathbb{E}[X] = \sum_{x \in \mathcal{X}} p(X = x)x,$$

where if X is continuous we again replace the sum with an integral. In the case of rolling a dice, we have $p(X = x) = 1/6$ for all x so we have

$$\mathbb{E}[X] = \sum_{x=1}^{6} p(X = x)x = \sum_{i=1}^{6} \frac{x}{6} = 3.5.$$

We can think of the expectation as a generalization of the notion of an *average*. If the probabilities are uniform then the expectation is the average over values of possible X, but the expectation can also give us

the "average" value we expect to see if the probabilities are non-uniform. For example, if we flip a coin that lands heads ($x = 1$) 75% of the time and lands tails ($x = 0$) 25% of the time, then the expectation is

$$\mathbb{E}[X] = (0.75)(1) + (0.25)(0) = 0.75,$$

which is a weighted average of the possibilities that reflects their probabilities of actually happening.

We can also talk about the *expected value of a function* f that depends on a random variable,

$$\mathbb{E}[f(X)] = \sum_{x \in \mathcal{X}} p(X = x) f(X).$$

Because f depends on a random variable, we call f a random function. Because the expectation is just a sum, *expectation is a linear operator* meaning that if α and β are (non-random) scalar values while f and g are functions then we have

$$\mathbb{E}[\alpha f(X) + \beta g(X)] = \alpha \mathbb{E}[f(X)] + \beta \mathbb{E}[g(X)].$$

However, note that in general we may have $\mathbb{E}[f(X)g(X)] \neq \mathbb{E}[f(X)]\mathbb{E}[g(X)]$. We define the *conditional expectation* of a random variable X given the value of another random variable $Y = y$ by

$$\mathbb{E}[X \mid Y = y] = \sum_{x \in \mathcal{X}} p(X = x \mid Y = y) f(X).$$

Notice that this is a function of Y but doesn't actually depend on X.

An important property of conditional expectations is that

$$\mathbb{E}[\mathbb{E}[X = x \mid Y = y]] = \mathbb{E}[X],$$

where the outer expectation is taken with respect to Y. This is called the *tower property*, *law of total expectation*, or the *iterated expectation* rule. It words it says that the expected value of X, averaged over all possible values that Y that we could condition on, is still the expected value of X.

CPSC 340:
Machine Learning and Data Mining

Probabilistic Classification

Fall 2019

Admin

- Assignment 1 is due tonight: you should be almost done.
  - You can use 1 late days to submit Monday, 2 for Wednesday.
- Waiting list people: everyone should be in.
- Course webpage:
  - https://www.cs.ubc.ca/~schmidtm/Courses/340-F19/
- Auditors:
  - Bring your forms at the end of class.
- Exchange students:
  - If you are still having trouble registering, bring me your forms.
  - Contact us on Piazza about getting registered for Gradescope.

Last Time: Training, Testing, and Validation

- Training step:

  Input: set of 'n' training examples x_i with labels y_i

  Output: a model that maps from arbitrary x_i to a $\hat{y}_i$

- Prediction step:

  Input: set of 't' testing examples $\tilde{x}_i$ and a model.

  Output: predictions $\hat{y}_i$ for the testing examples.

- What we are interested in is the test error:
  - Error made by prediction step on new data.

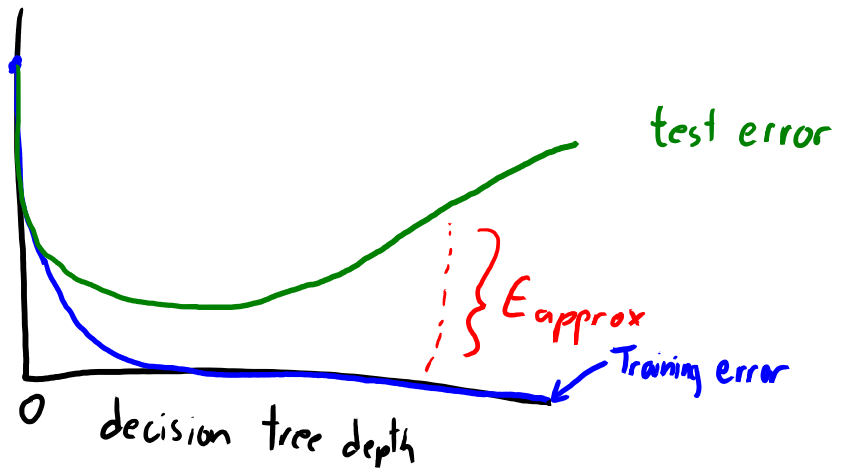
Last Time: Fundamental Trade-Off

- We decomposed test error to get a fundamental trade-off:

$$E_{test} = E_{approx} + E_{train}$$

"test error" "approximation error" "training error"

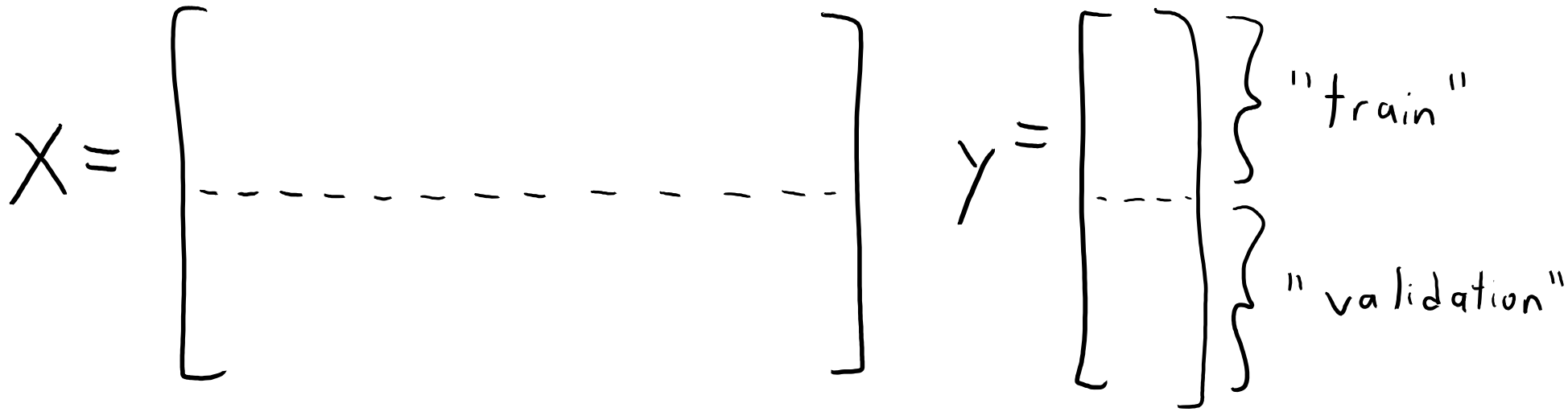
  - Where $E_{approx} = (E_{test} - E_{train})$.



- E_{train} goes down as model gets complicated:
  - Training error goes down as a decision tree gets deeper.
- But E_{approx} goes up as model gets complicated:
  - Training error becomes a worse approximation of test error.

Last Time: Validation Error

- Golden rule: we can't look at test data during training.
- But we can approximate E_{test} with a validation error:
  - Error on a set of training examples we "hid" during training.



  - Find the decision tree based on the "train" rows.
  - Validation error is the error of the decision tree on the "validation" rows.
    - We typically choose "hyper-parameters" like depth to minimize the validation error.

Overfitting to the Validation Set?

- Validation error usually has lower optimization bias than training error.
  - Might optimize over 20 values of "depth", instead of millions+ of possible trees.

- But we can still overfit to the validation error (common in practice):
  - Validation error is only an unbiased approximation if you use it once.
  - Once you start optimizing it, you start to overfit to the validation set.

- This is most important when the validation set is "small":
  - The optimization bias decreases as the number of validation examples increases.

- Remember, our goal is still to do well on the test set (new data), not the validation set (where we already know the labels).

Should you trust them?

- Scenario 1:
  - "I built a model based on the data you gave me."
  - "It classified your data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably not:
  - They are reporting training error.
  - This might have nothing to do with test error.
  - E.g., they could have fit a very deep decision tree.
- Why 'probably'?
  - If they only tried a few very simple models, the 98% might be reliable.
  - E.g., they only considered decision stumps with simple 1-variable rules.

Should you trust them?

- Scenario 2:
  - "I built a model based on half of the data you gave me."
  - "It classified the other half of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably:
  - They computed the validation error once.
  - This is an unbiased approximation of the test error.
  - Trust them if you believe they didn't violate the golden rule.

Should you trust them?

- Scenario 3:
  - "I built 10 models based on half of the data you gave me."
  - "One of them classified the other half of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably:
  - They computed the validation error a small number of times.
  - Maximizing over these errors is a biased approximation of test error.
  - But they only maximized it over 10 models, so bias is probably small.
  - They probably know about the golden rule.

Should you trust them?

- Scenario 4:
  - "I built 1 billion models based on half of the data you gave me."
  - "One of them classified the other half of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably not:
  - They computed the validation error a huge number of times.
  - They tried so many models, one of them is likely to work by chance.
- Why 'probably'?
  - If the 1 billion models were all extremely-simple, 98% might be reliable.

Should you trust them?

- Scenario 5:
  - "I built 1 billion models based on the first third of the data you gave me."
  - "One of them classified the second third of the data with 98% accuracy."
  - "It also classified the last third of the data with 98% accuracy."
  - "It should get 98% accuracy on the rest of your data."

- Probably:
  - They computed the first validation error a huge number of times.
  - But they had a second validation set that they only looked at once.
  - The second validation set gives unbiased test error approximation.
  - This is ideal, as long as they didn't violate golden rule on the last third.
  - And assuming you are using IID data in the first place.

Validation Error and Optimization Bias

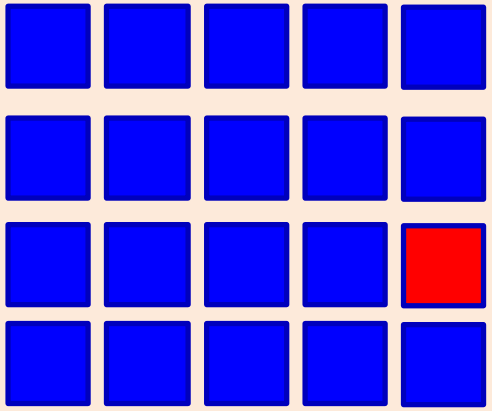
- Optimization bias is small if you only compare a few models:
  - Best decision tree on the training set among depths 1, 2, 3,…, 10.
  - Risk of overfitting to validation set is low if we try 10 things.

- Optimization bias is large if you compare a lot of models:
  - All possible decision trees of depth 10 or less.
  - Here we're using the validation set to pick between a billion+ models:
    - Risk of overfitting to validation set is high: could have low validation error by chance.
  - If you did this, you might want a second validation set to detect overfitting.

- And optimization bias shrinks as you grow size of validation set.

Aside: Optimization Bias leads to Publication Bias

- Suppose that 20 researchers perform the exact same experiment:

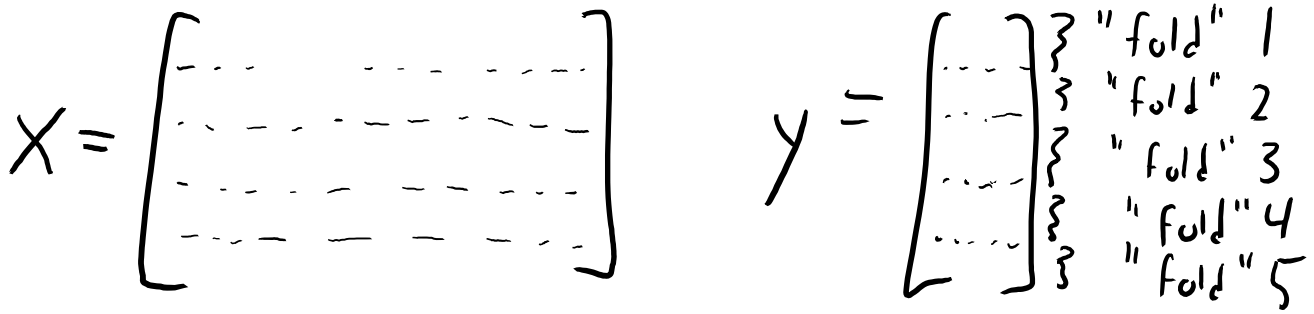


- They each test whether their effect is "significant" ($p < 0.05$).
  - 19/20 find that it is not significant.
  - But the 1 group finding it's significant publishes a paper about the effect.

- This is again optimization bias, contributing to publication bias.
  - A contributing factor to many reported effects being wrong.

Cross-Validation (CV)

- Isn't it wasteful to only use part of your data?
- 5-fold cross-validation:
  - Train on 80% of the data, validate on the other 20%.
  - Repeat this 5 more times with different splits, and average the score.

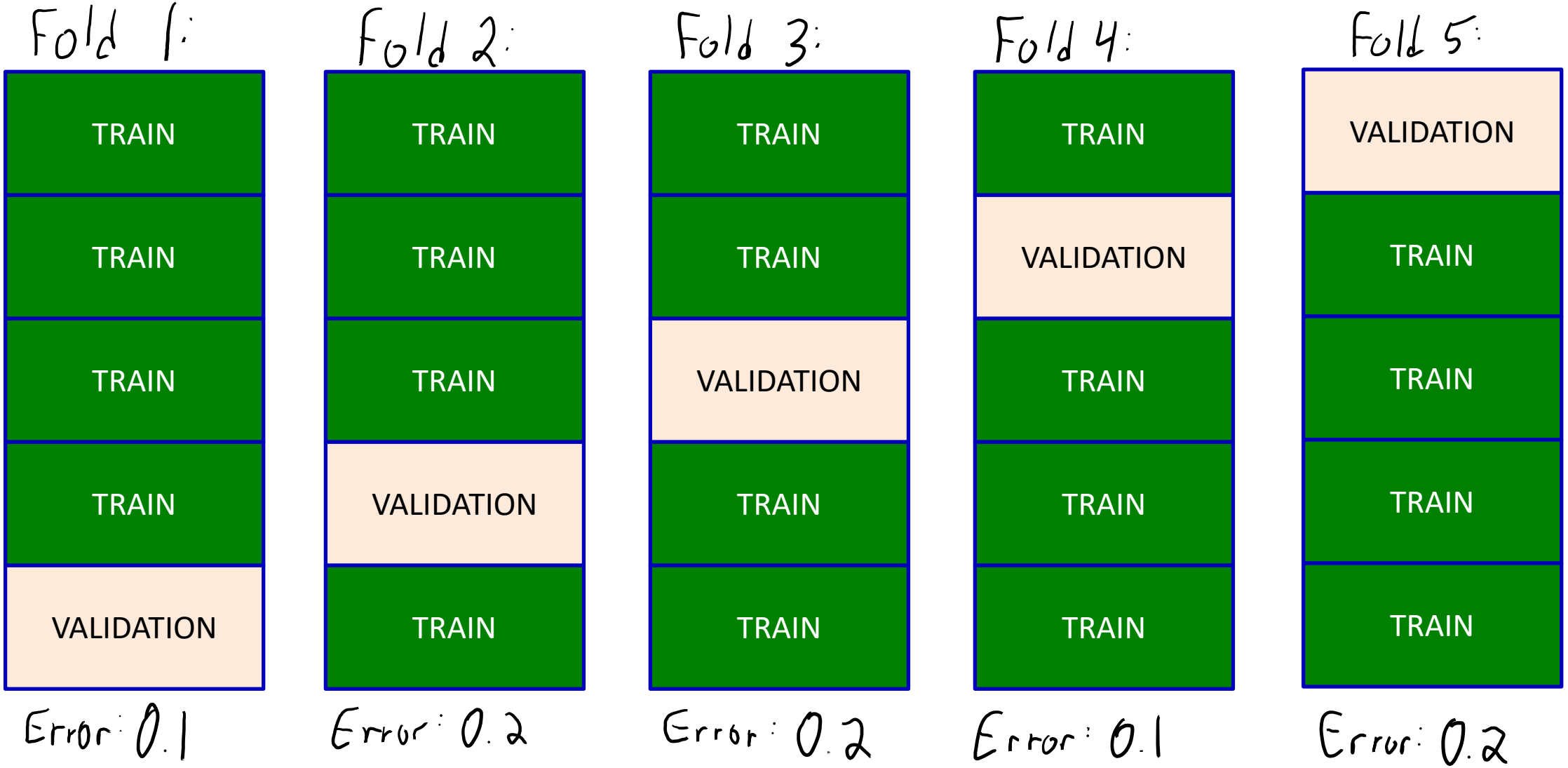


1. Train on folds $\{1,2,3,4\}$, compute error on fold 5.
2. Train on folds $\{1,2,3,5\}$, compute error on fold 4.
3. Train on folds $\{1,2,4,5\}$, compute error on fold 3.

⋮

6. Take average of the 5 errors as approximation of test error

Cross-Validation (CV)



CV error estimate for this hyper-parameter: mean(errors) = 0.16

Cross-Validation Pseudo-Code

To choose depth

```
for depth in 1:20
    compute cross-validation score
return depth with highest score
```

To compute 5-fold cross-validation score:

```
for fold in 1:5
    train 80% that doesn't include fold
    test on fold
return average test error
```

Notes:

- This fits 100 models! (20 depths times 5 folds)
- We get one (average) score for each of the 20 depths.
- Use this score to pick depth

Cross-Validation (CV)

- You can take this idea further ("k-fold cross-validation"):
  - 10-fold cross-validation: train on 90% of data and validate on 10%.
    - Repeat 10 times and average (test on fold 1, then fold 2,…, then fold 10),
  - Leave-one-out cross-validation: train on all but one training example.
    - Repeat n times and average.
- Gets more accurate but more expensive with more folds.
  - To choose depth we compute the cross-validation score for each depth.

- As before, if data is ordered then folds should be random splits.
  - Randomize first, then split into fixed folds.

(pause)

The “Best” Machine Learning Model

- Decision trees are not always most accurate on test error.
- What is the “best” machine learning model?
- An alternative measure of performance is the generalization error:
  - Average error over all x_i vectos that are not seen in the training set.
  - “How well we expect to do for a *completely unseen* feature vector”.
- No free lunch theorem (proof in bonus slides):
  - There is **no** “best” model achieving the best generalization error for every problem.
  - If model A generalizes better to new data than model B on one dataset, there is another dataset where model B works better.
- This question is like asking which is “best” among “rock”, “paper”, and “scissors”.

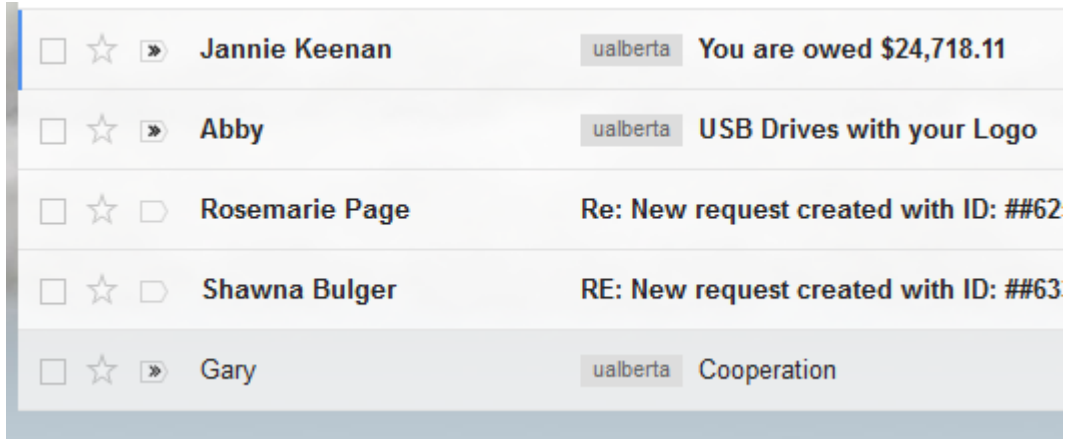
The “Best” Machine Learning Model

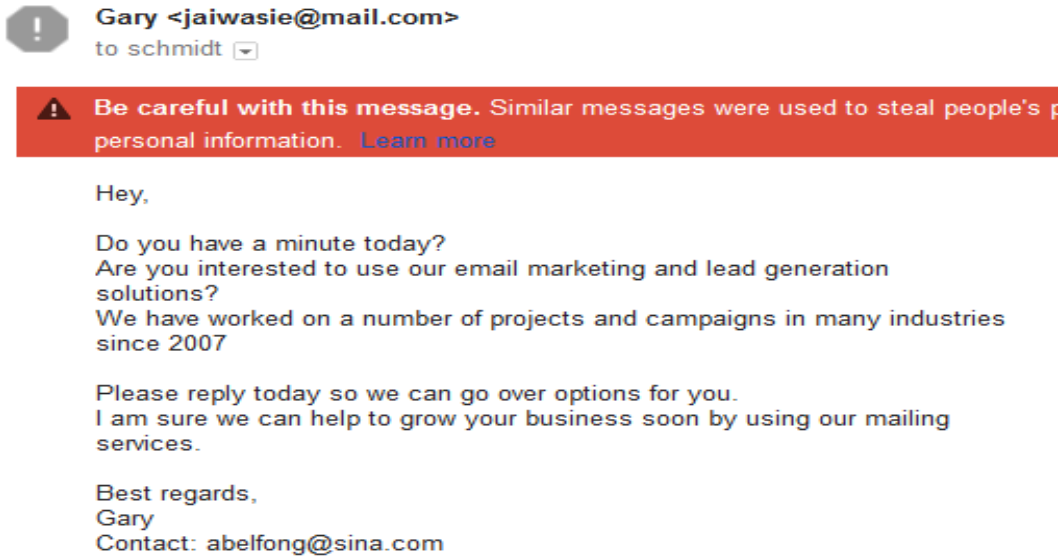
- Implications of the lack of a “best” model:
  - We need to learn about and try out multiple models.
- So which ones to study in CPSC 340?
  - We’ll usually motivate each method by a specific application.
  - But we’re focusing on models that have been effective in many applications.

- Caveat of no free lunch (NFL) theorem:
  - The world is very structured.
  - Some datasets are more likely than others.
  - Model A really could be better than model B on every real dataset in practice.
- Machine learning research:
  - Large focus on models that are useful across many applications.

Application: E-mail Spam Filtering

- Want a build a system that detects spam e-mails.
  - Context: spam used to be a big problem.



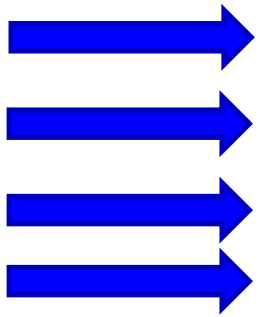


- Can we formulate as supervised learning?

Spam Filtering as Supervised Learning

- Collect a large number of e-mails, gets users to label them.

$	Hi	CPSC	340	Vicodin	Offer	…	Spam?
1	1	0	0	1	0	…	1
0	0	0	0	1	1	…	1
0	1	1	1	0	0	…	0
…	…	…	…	…	…	…	…



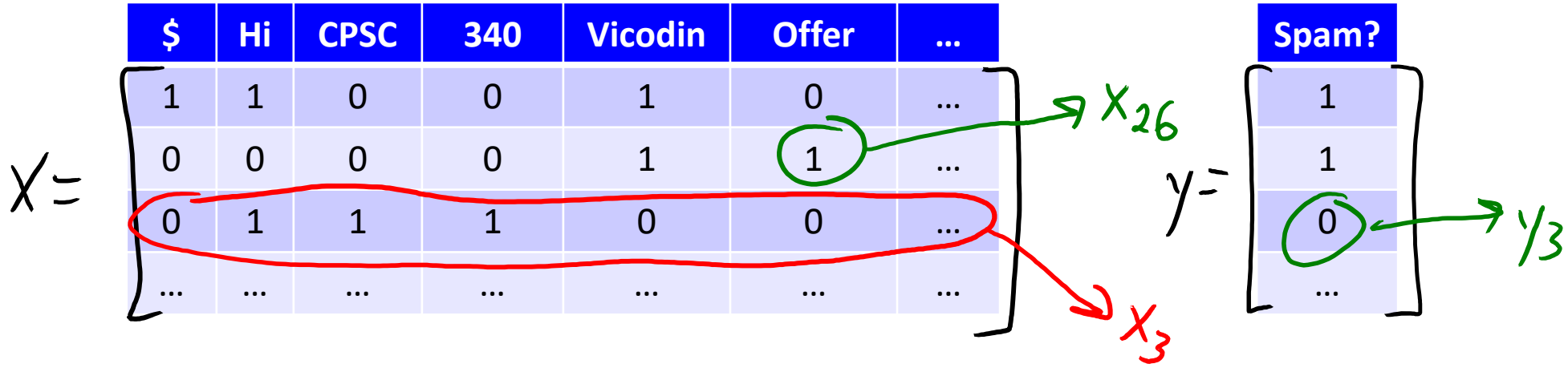
- We can use ($y_i = 1$) if e-mail 'i' is spam, ($y_i = 0$) if e-mail is not spam.
- Extract features of each e-mail (like bag of words).
  - ($x_{ij} = 1$) if word/phrase 'j' is in e-mail 'i', ($x_{ij} = 0$) if it is not.

Feature Representation for Spam

- Are there better features than bag of words?
  - We add bigrams (sets of two words):
    - “CPSC 340”, “wait list”, “special deal”.
  - Or trigrams (sets of three words):
    - “Limited time offer”, “course registration deadline”, “you’re a winner”.
  - We might include the sender domain:
    - <sender domain == “mail.com”>.
  - We might include regular expressions:
    - <your first and last name>.

Review of Supervised Learning Notation

- We have been using the notation 'X' and 'y' for supervised learning:



- X is matrix of all features, y is vector of all labels.
  - We use y_i for the label of example 'i' (element 'i' of 'y').
  - We use x_{ij} for feature 'j' of example 'i'.
  - We use x_i as the list of features of example 'i' (row 'i' of 'X').
    - So in the above x_3 = [0 1 1 1 0 0 …].
    - In practice, only store list of non-zero features for each x_i (small memory requirement).

Probabilistic Classifiers

- For years, best spam filtering methods used naïve Bayes.
  - A probabilistic classifier based on Bayes rule.
  - It tends to work well with bag of words.
  - Recently shown to improve on state of the art for CRISPR "gene editing" (link).

- Probabilistic classifiers model the conditional probability, $p(y_i \mid x_i)$.
  - "If a message has words x_i, what is probability that message is spam?"

- Classify it as spam if probability of spam is higher than not spam:
  - If $p(y_i = \text{"spam"} \mid x_i) > p(y_i = \text{"not spam"} \mid x_i)$
    - return "spam".
  - Else
    - return "not spam".

Spam Filtering with Bayes Rule

- To model conditional probability, naïve Bayes uses Bayes rule:

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- So we need to figure out three types of terms:
  - Marginal probability $p(y_i)$ that an e-mail is spam.
  - Marginal probability $p(x_i)$ that an e-mail has the set of words x_i.
  - Conditional probability $p(x_i \mid y_i)$ that a spam e-mail has the words x_i.
    - And the same for non-spam e-mails.

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- What do these terms mean?

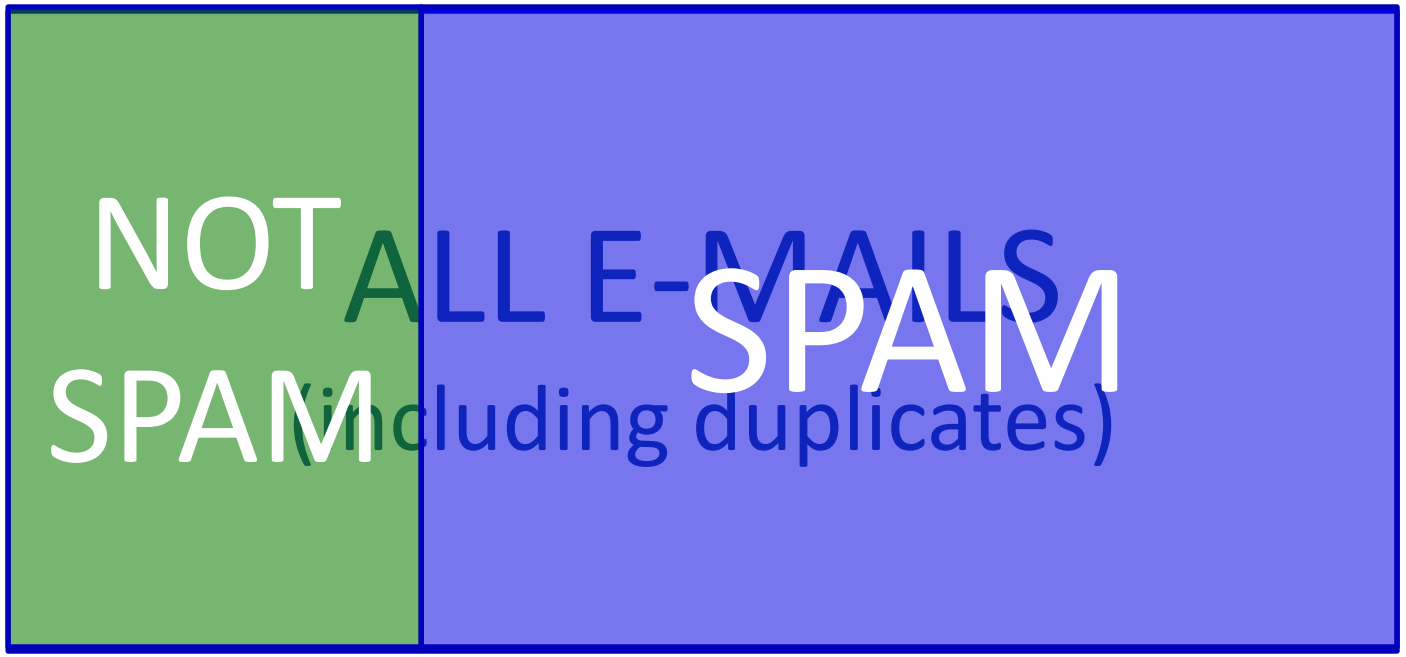
ALL E-MAILS

(including duplicates)

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(y_i$ = "spam") is probability that a random e-mail is spam.
  - This is easy to approximate from data: use the proportion in your data.



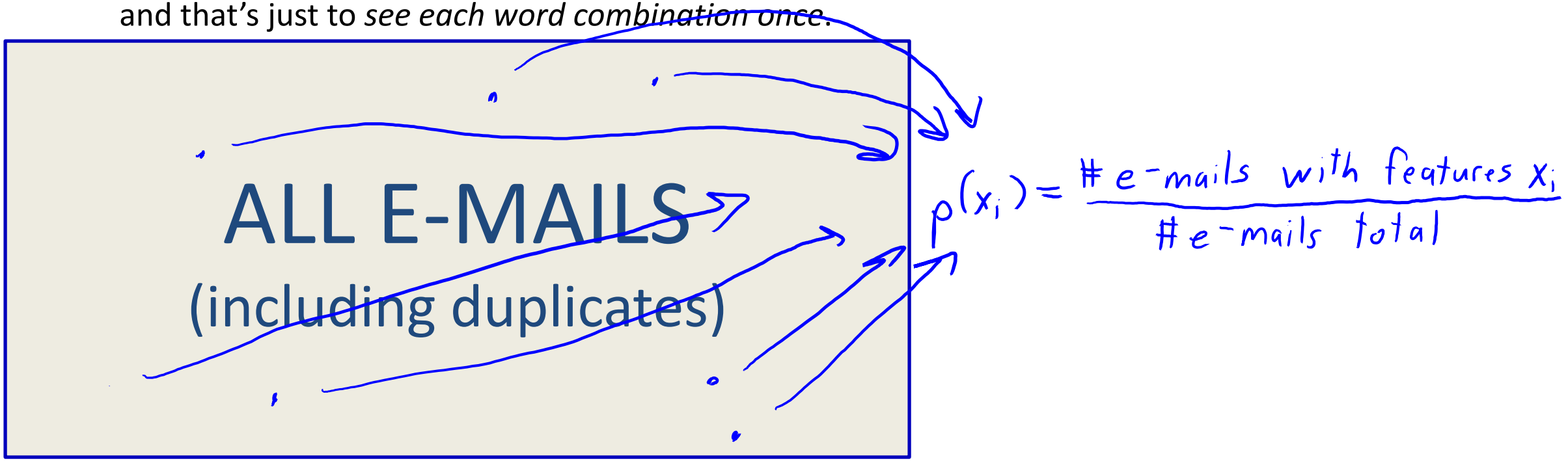
$$p(y_i = \text{"spam"}) = \frac{\#\text{ spam messages}}{\#\text{ total messages}}$$

This is an "estimate" of the true probability. In particular, this formula is a "maximum likelihood estimate" (MLE). We will cover likelihoods and MLEs later in the course.

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(x_i)$ is probability that a random e-mail has features x_i:
  - Hard to approximate: with 'd' words we need to collect 2^d "coupons",
    and that's just to *see each word combination once*.



Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(x_i)$ is probability that a random e-mail has features x_i:
  - Hard to approximate: with 'd' words we need to collect 2^d "coupons",
    but it turns out we can ignore it:

Naive Bayes returns "spam" if $p(y_i = \text{"spam"} \mid x_i) > p(y_i = \text{"not spam"} \mid x_i)$

By Bayes rule this means $\frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)} > \frac{p(x_i \mid y_i = \text{"not spam"})\, p(y_i = \text{"not spam"})}{p(x_i)}$

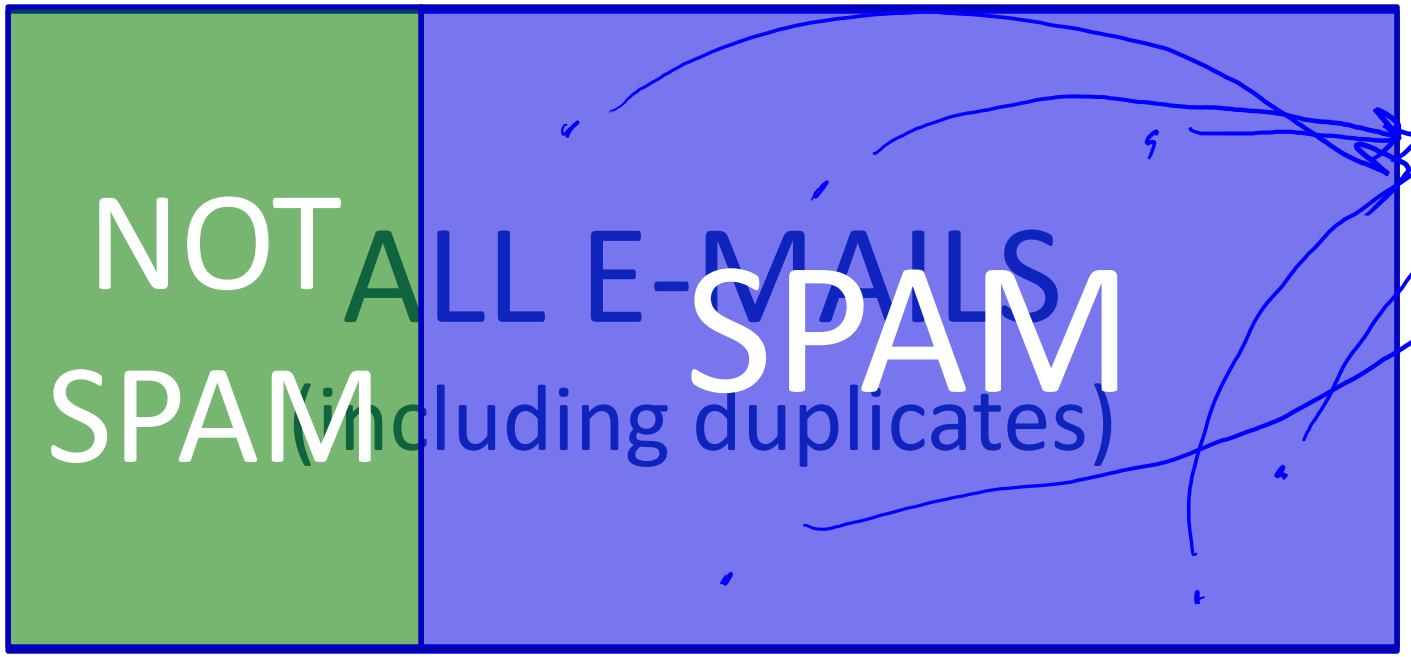
Multiply both sides by $p(x_i)$:

$$p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"}) > p(x_i \mid y_i = \text{"not spam"})\, p(y_i = \text{"not spam"})$$

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(x_i \mid y_i = \text{“spam”})$ is probability that spam has features x_i.



$$p(x_i \mid y_i = \text{"spam"}) = \frac{\#\text{ spam messages with features } x_i}{\#\text{ spam messages}}$$

- Also hard to approximate.
  - And we need it.

Naïve Bayes

- Naïve Bayes makes a big assumption to make things easier:

$$p(\text{hello}=1, \text{vicodin}=0, 340=1 \mid \text{spam}) \approx p(\text{hello}=1 \mid \text{spam})\, p(\text{vicodin}=0 \mid \text{spam})\, p(340=1 \mid \text{spam})$$

HARD

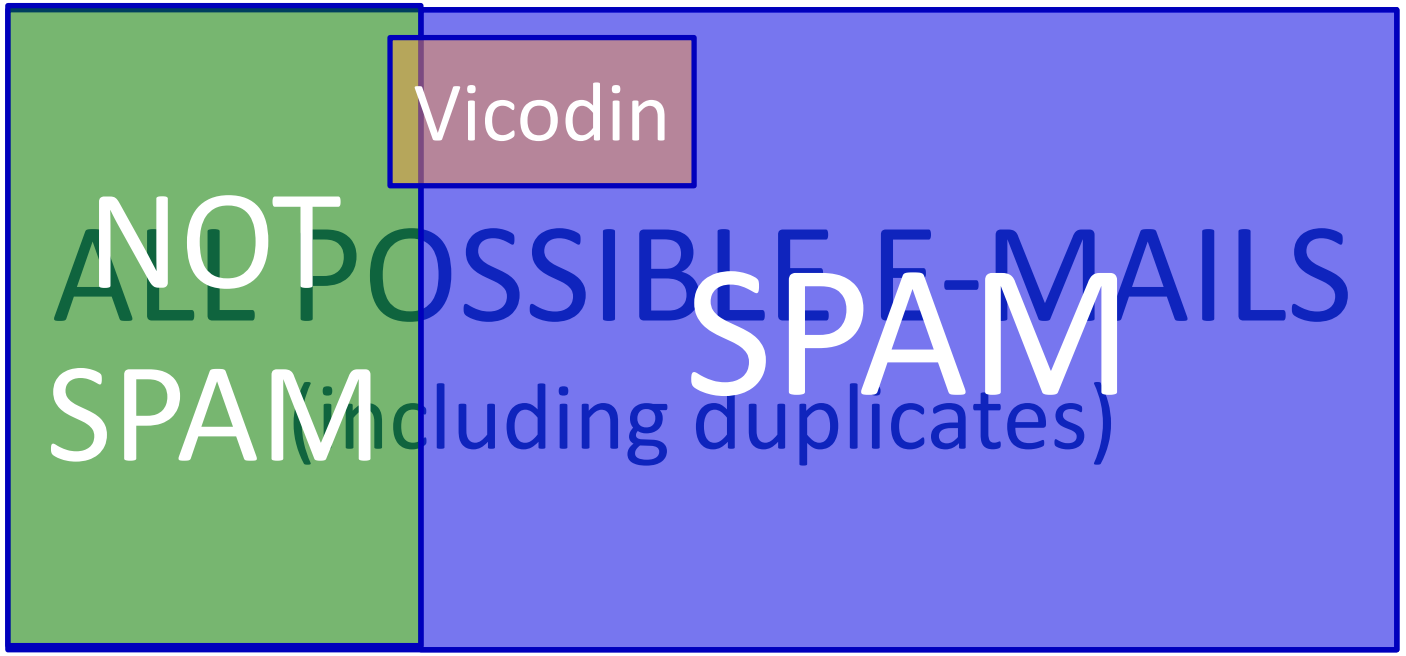
easy easy easy

- We assume *all* features x_i are conditionally independent give label y_i.
  - Once you know it's spam, probability of "vicodin" doesn't depend on "340".
  - Definitely not true, but sometimes a good approximation.

- And now we only need easy quantities like p("vicodin" = 0| y_i = "spam").

Naïve Bayes

- p("vicodin" = 1 | "spam" = 1) is probability of seeing "vicodin" in spam.



- Easy to estimate:

$$p(\text{vicodin} = 1 \mid \text{spam} = 1) = \frac{\#\text{ spam messages w/ vicodin}}{\#\text{ spam messages}}$$

Again, this is a "maximum likelihood estimate" (MLE). We will cover how to derive this later.

Summary

- Optimization bias: using a validation set too much overfits.
- Cross-validation: allows better use of data to estimate test error.
- No free lunch theorem: there is no "best" ML model.
- Probabilistic classifiers: try to estimate $p(y_i \mid x_i)$.
- Naïve Bayes: simple probabilistic classifier based on counting.
  - Uses conditional independence assumptions to make training practical.

- Next time:
  - A "best" machine learning model as 'n' goes to ∞.

Back to Decision Trees

- Instead of validation set, you can use CV to select tree depth.

- But you can also use these to decide whether to split:
  - Don't split if validation/CV error doesn't improve.
  - Different parts of the tree will have different depths.
- Or fit deep decision tree and use [cross-]validation to prune:
  - Remove leaf nodes that don't improve CV error.

- Popular implementations that have these tricks and others.

Random Subsamples

- Instead of splitting into k-folds, consider "random subsample" method:
  - At each "round", choose a random set of size 'm'.
    - Train on all examples except these 'm' examples.
    - Compute validation error on these 'm' examples.

- Advantages:
  - Still an unbiased estimator of error.
  - Number of "rounds" does not need to be related to "n".
- Disadvantage:
  - Examples that are sampled more often get more "weight".

Cross-Validation Theory

- Does CV give unbiased estimate of test error?
  - Yes!
    - Since each data point is only used once in validation, expected validation error on each data point is test error.
  - But again, if you use CV to select among models then it is no longer unbiased.

- What about variance of CV?
  - Hard to characterize.
  - CV variance on ‘n’ data points is worse than with a validation set of size ‘n’.
    - But we believe it is close.

- Does cross-validation remove optimization bias?
  - No, but the bias might be smaller since you have more “test” points.

Handling Data Sparsity

- Do we need to store the full bag of words 0/1 variables?
  - No: only need list of non-zero features for each e-mail.

$	Hi	CPSC	340	Vicodin	Offer	…
1	1	0	0	1	0	…
0	0	0	0	1	1	…
0	1	1	1	0	0	…
1	1	0	0	0	1	…

vs.

Non-Zeroes
{1,2,5,…}
{5,6,…}
{2,3,4,…}
{1,2,6,…}

  - Math/model doesn't change, but more efficient storage.

Proof of No Free Lunch Theorem

- Let's show the "no free lunch" theorem in a simple setting:
  - The x^i and y^i are binary, and y^i being a deterministic function of x^i.
- With 'd' features, each "learning problem" is a map from each of the 2^d feature combinations to 0 or 1: $\{0,1\}^d$ -> $\{0,1\}$

Feature 1	Feature 2	Feature 3
0	0	0
0	0	1
0	1	0
…	…	…

Map 1	Map 2	Map 3	…
0	1	0	…
0	0	1	…
0	0	0	…
…	…	…	…

- Let's pick one of these maps ("learning problems") and:
  - Generate a set training set of 'n' IID samples.
  - Fit model A (convolutional neural network) and model B (naïve Bayes).

Proof of No Free Lunch Theorem

- Define the "unseen" examples as the $(2^d - n)$ not seen in training.
  - Assuming no repetitions of x^i values, and $n < 2^d$.
  - Generalization error is the average error on these "unseen" examples.
- Suppose that model A got 1% error and model B got 60% error.
  - We want to show model B beats model A on another "learning problem".

- Among our set of "learning problems" find the one where:
  - The labels y^i agree on all training examples.
  - The labels y_i disagree on all "unseen" examples.
- On this other "learning problem":
  - Model A gets 99% error and model B gets 40% error.

Proof of No Free Lunch Theorem

- Further, across all "learning problems" with these 'n' examples:
  - Average generalization error of **every** model is 50% on unseen examples.
    - It's right on each unseen example in exactly half the learning problems.
  - With 'k' classes, the average error is (k-1)/k (random guessing).

- This is kind of depressing:
  - For general problems, no "machine learning" is better than "predict 0".

- But the proof also reveals the problem with the NFL theorem:
  - Assumes every "learning problem" is equally likely.
  - World encourages patterns like "similar features implies similar labels".

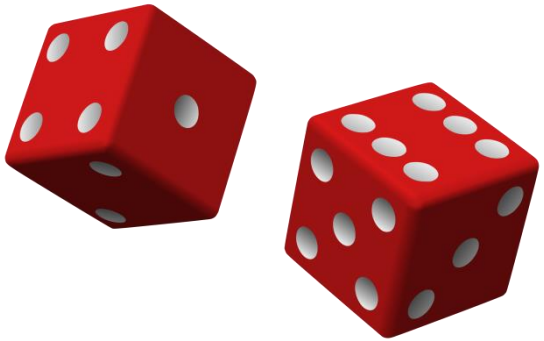
Overview of Probability

Mark Schmidt

September 12, 2017

Practical Application…

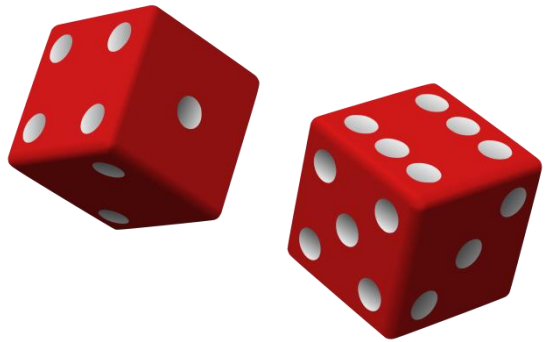
- Dungeons & Dragons scenario:
  - You roll dice 1:
    - Roll 5 or 6 you sneak past monster.
    - Otherwise, you are eaten.
  - If you survive, you roll dice 2:
    - Roll 4-6, find pizza.
    - Otherwise, you find nothing.



https://en.wikipedia.org/wiki/Dice_throw_%28review%29
http://www.dungeonsdragonscartoon.com/2011/11/cloak.html

Practical Application …

- Dungeons & Dragons scenario:
  - You roll dice 1:
    - Roll 5 or 6 you sneak past monster.
    - Otherwise, you are eaten.
  - If you survive, you roll dice 2:
    - Roll 4-6, find pizza.
    - Otherwise, you find nothing.



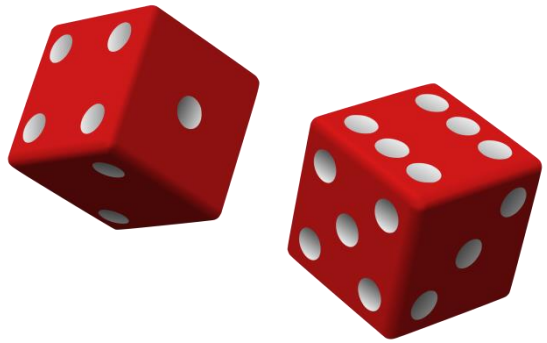
- Probabilities defined on ‘event space’:

D1\D2	1	2	3	4	5	6
1						
2						
3		$D_1=3, D_2=2$				
4						
5						
6						

https://en.wikipedia.org/wiki/Dice_throw_%28review%29
http://www.dungeonsdragonscartoon.com/2011/11/cloak.html

Practical Application …

- Dungeons & Dragons scenario:
  - You roll dice 1:
    - Roll 5 or 6 you sneak past monster.
    - Otherwise, you are eaten.
  - If you survive, you roll dice 2:
    - Roll 4-6, find pizza.
    - Otherwise, you find nothing.



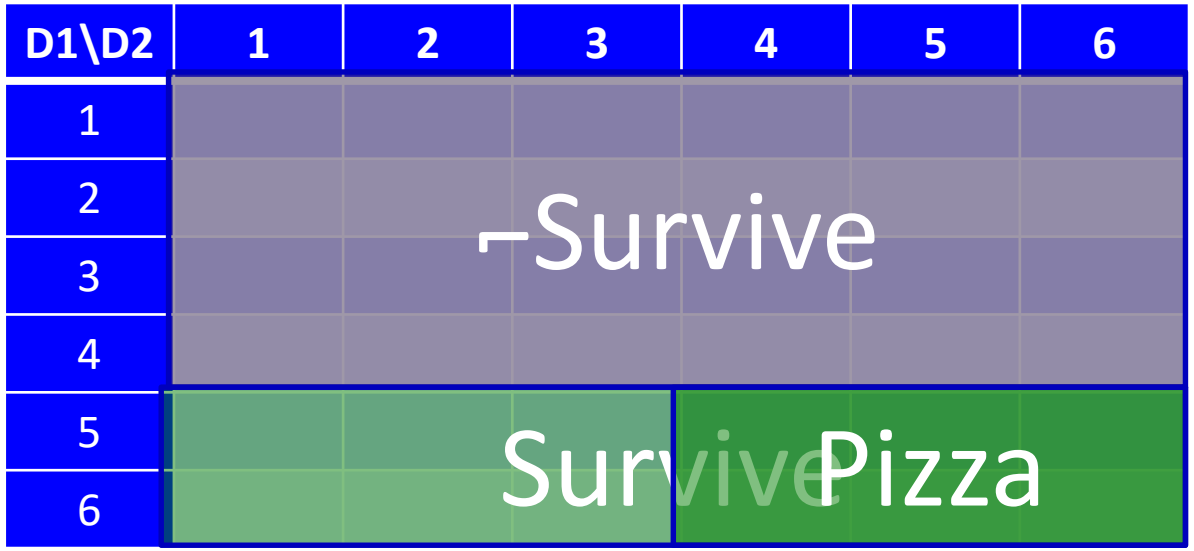
- Probabilities defined on ‘event space’:

D1\D2	1	2	3	4	5	6
1	Survive					
2						
3						
4						
5	Survive				Pizza	
6						

https://en.wikipedia.org/wiki/Dice_throw_%28review%29
http://www.dungeonsdragonscartoon.com/2011/11/cloak.html

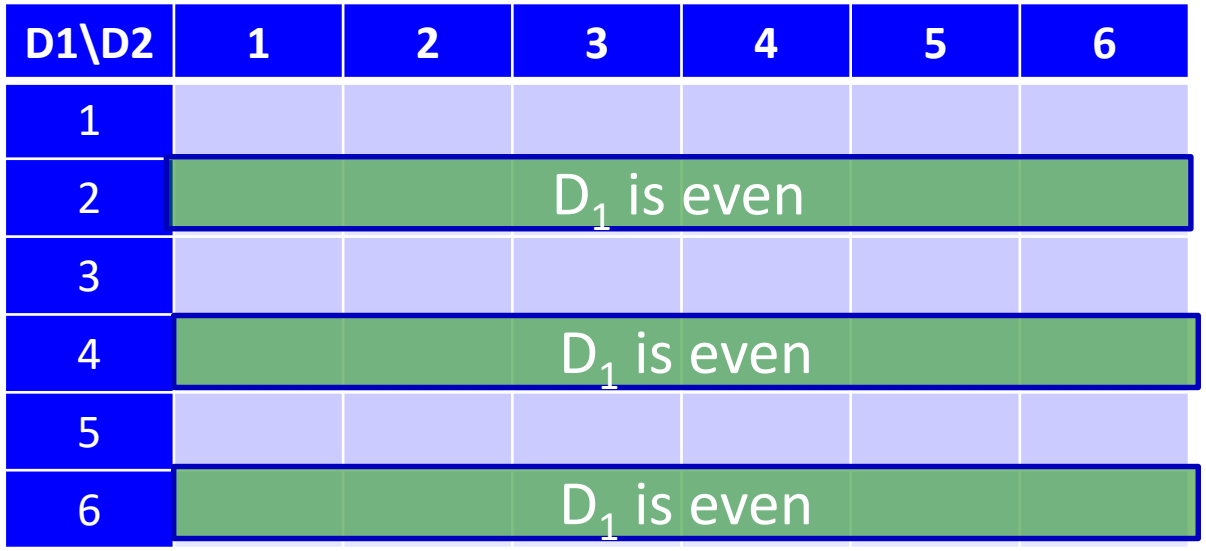
Calculating Basic Probabilities

- Probability of event ‘A’ is ratio:
  - p(A) = Area(A)/TotalArea.
  - “Likelihood” that ‘A’ happens.
- Examples:
  - p(Survive) = 12/36 = 1/3.
  - p(Pizza) = 6/36 = 1/6.
  - p(~Survive) = 1 – p(Survive) = 2/3.



Calculating Basic Probabilities

- Probability of event ‘A’ is ratio:
  - p(A) = Area(A)/TotalArea.
  - “Likelihood” that ‘A’ happens.
- Examples:
  - p(Survive) = 12/36 = 1/3.
  - p(Pizza) = 6/36 = 1/6.
  - p(¬Survive) = 1 – p(Survive) = 2/3.
  - p(D_1 is even) = 18/36 = ½.



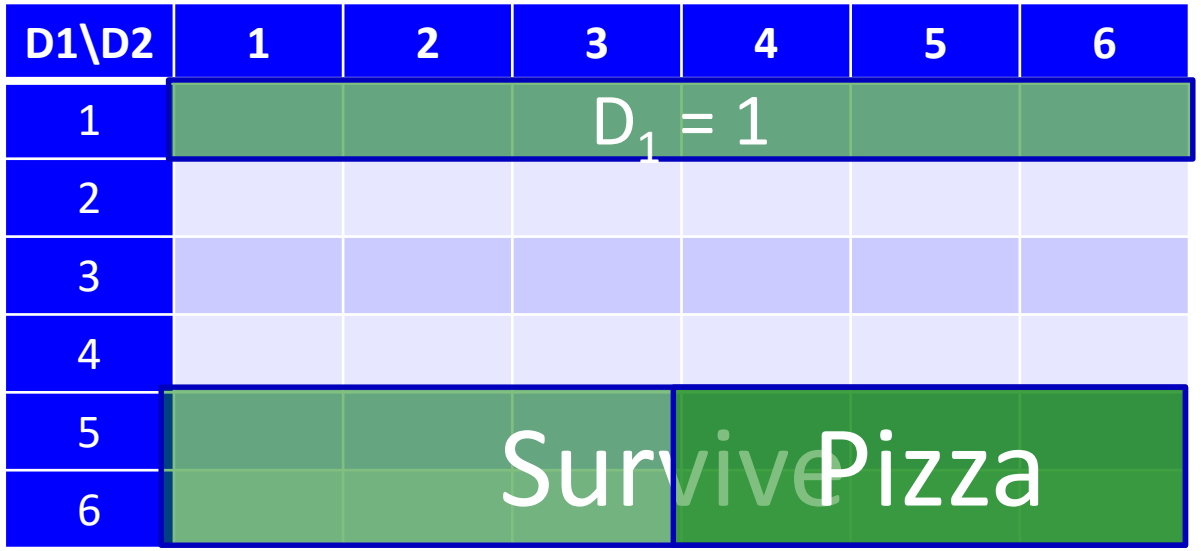
Random Variables and 'Sum to 1' Property

- Random variable: variable whose value depends on probability.
- Example: event (D_1 = x) depends on random variable D_1.
- Convention:
  - We'll use p(x) to mean p(X = x), when random variable X is obvious.
- Sum of probabilities of random variable over entire domain is 1:
  - $\sum_x p(x) = 1$.
  - E.g, $\sum_i p(D_1 = i)$ = 1/6+1/6 + …
    = 1.

D1\D2	1	2	3	4	5	6
1	D_1 =1					
2	D_1 =2					
3	D_1 = 3					
4	D_1 = 4					
5	D_1 = 5					
6	D_1 = 6					

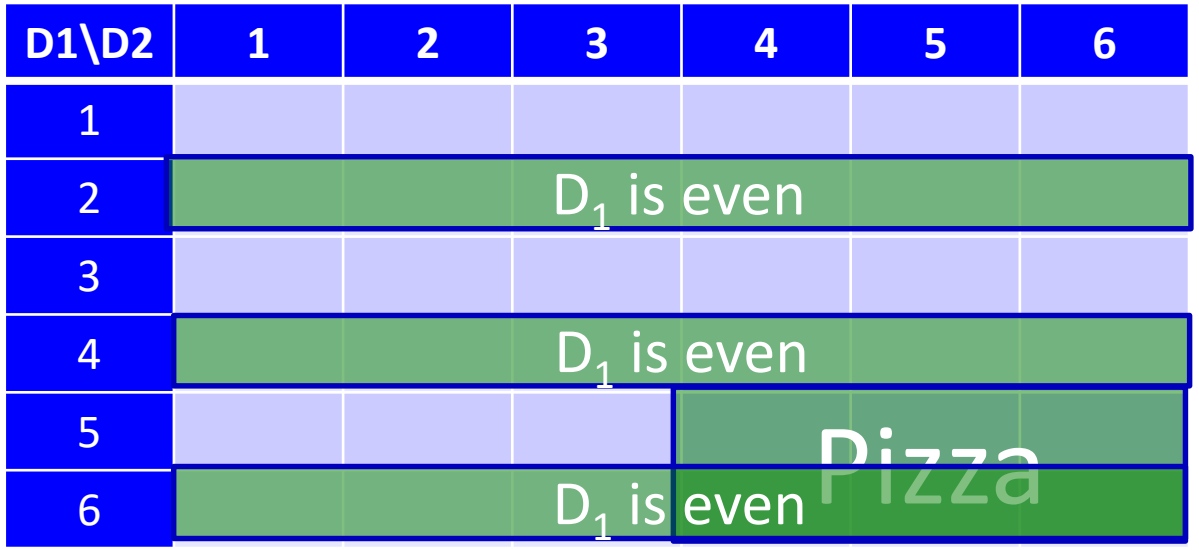
Joint Probability

- Joint probability: probability that A and B happen, written ‘p(A,B)’.
  - Intersection of Area(A) and Area(B).
- Examples:
  - $p(D_1 = 1, \text{Survive}) = 0$.
  - $p(\text{Survive}, \text{Pizza}) = 6/36 = 1/6$.



Joint Probability

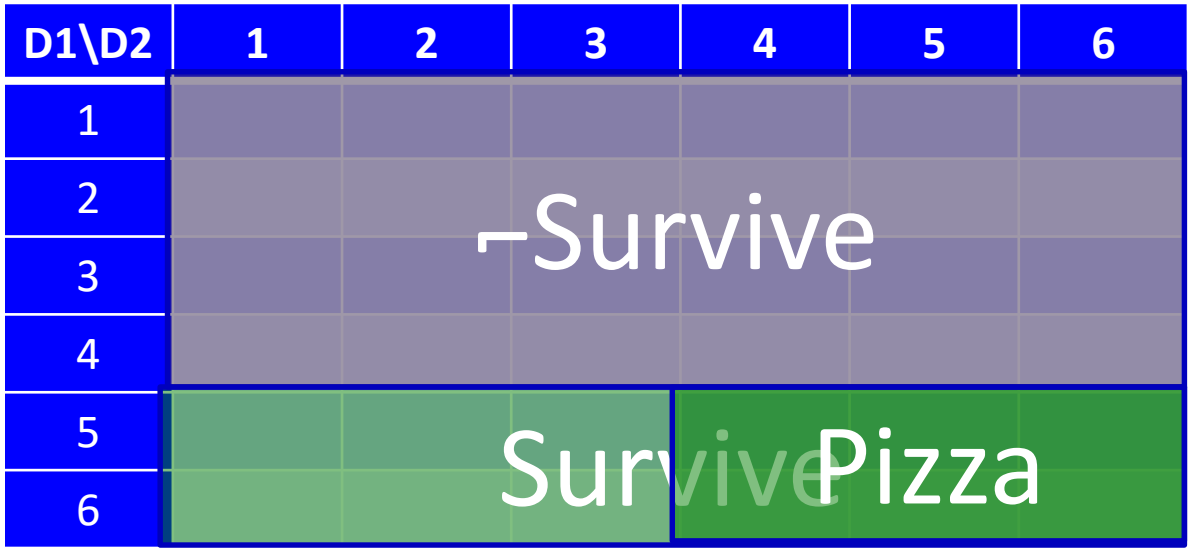
- Joint probability: probability that A and B happen, written ‘p(A,B)’.
  - Intersection of Area(A) and Area(B).
- Examples:
  - $p(D_1 = 1, \text{Survive}) = 0$.
  - $p(\text{Survive}, \text{Pizza}) = 6/36 = 1/6$.
  - $p(D_1 \text{ even}, \text{Pizza}) = 3/36 = 1/12$.



- Note: order of A and B does not matter

Marginalization Rule

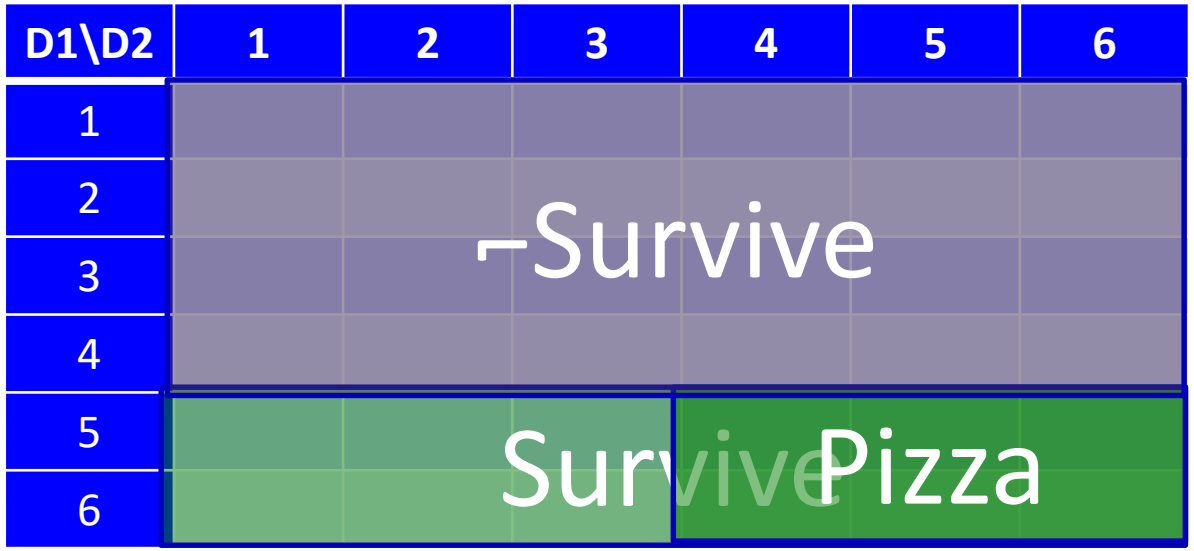
- Marginalization rule:
  - $P(A) = \sum_x P(A, X = x)$.
  - Summing joint over all values of one variable gives probability of the other.
  - Example: $P(Pizza) = P(Pizza, Survive) + P(Pizza, \neg Survive) = \frac{1}{6}$.



  - Applying rule twice: $\sum_x \sum_y p(Y = y, X = x) = 1$.

Conditional Probability

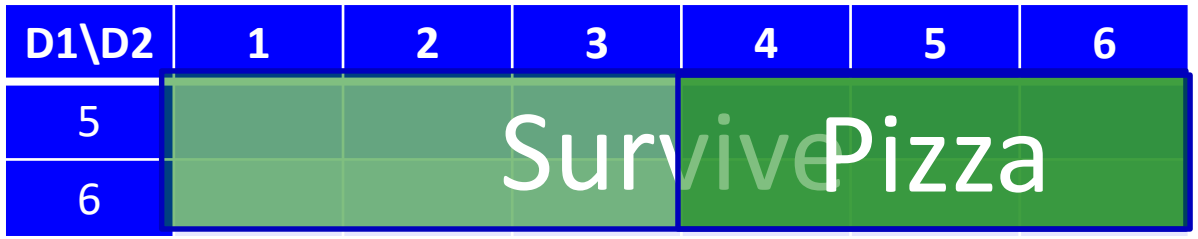
- Conditional probability:
  - probability that A will happen *if we know* that B happens.
  - "probability of A *restricted* to scenarios where B happens".
  - Written p(A|B), said "probability of A given B".
- Calculation:
  - Within area of B:
    - Compute Area(A)/TotalArea.
  - p(Pizza | Survive) =



Conditional Probability

- Conditional probability:
  - probability that A will happen *if we know* that B happens.
  - "probability of A *restricted* to scenarios where B happens".
  - Written p(A|B), said "probability of A given B".
- Calculation:
  - Within area of B:
    - Compute Area(A)/TotalArea.
  - p(Pizza | Survive) =
    p(Pizza, Survive)/p(Survive) = 6/12 = ½.
  - Higher than p(Pizza, Survive) = 6/36 = 1/6.
  - More generally, p(A | B) = p(A,B)/p(B).

Geometrically: compute area of A on new space where B happened.



‘Sum to 1’ Properties and Bayes Rule.

- Conditional probability P(A | B) sums to one over all A:
  - $\sum_x P(x \mid B) = 1$.
  - P(Pizza | Survive) + P(~ Pizza | Survive) = 1.
  - P(Pizza | Survive) + P(Pizza | ~Survive) ≠ 1.
- Product rule: p(A,B) = p(A | B)p(B).
- Bayes Rule: $$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)}$$
  - Allows you to “reverse” the conditional probability.
- Example:
  - P(Pizza | Survive) = P(Survive | Pizza)P(Pizza)/P(Survive)
    = (1) * (1/6) / (1/3) = ½.
  - http://setosa.io/ev/conditional-probability

Independence of Random Variables

- Events A and B are independent if $p(A,B) = p(A)p(B)$.
  - Equivalently: $p(A|B) = p(A)$.
  - "Knowing B happened tells you nothing about A".
  - We use the notation:

$$A \perp B$$

- Random variables are independent if $p(x,y) = p(x)p(y)$ for all x and y.
  - Flipping two coins:

$$p(C_1 = \text{'heads'}, C_2 = \text{'heads'}) = p(C_1 = \text{'heads'})p(C_2 = \text{'heads'}).$$
$$p(C_1 = \text{'tails'}, C_2 = \text{'heads'}) = p(C_1 = \text{'tails'})p(C_2 = \text{'heads'}).$$
$$\ldots$$

Conditional Independence

- A and B are conditionally independent given C if

$$p(A, B \mid C) = p(A \mid C)p(B \mid C).$$

  - Equivalently: $p(A \mid B, C) = p(A \mid C)$.
  - "Knowing C happened, also knowing B happened says nothing about A".
  - Example: $p(\text{Pizza} \mid D_1, \text{Survive}) = p(\text{Pizza} \mid \text{Survive})$.
  - Knowing you survived, dice 1 gives no information about chance of pizza.
  - We use the notation: $A \perp B \mid C$

- Semantics of $p(A, B \mid C, D)$:
  - "probability of A and B happening, if we know that C and D happened".

Conditional Independence

- Example: food poisoning
  - If food was bad, each person independently gets sick with probability 50%
  - Unconditionally, me getting and and you getting sick are NOT independent
    - If I got sick, that makes me think the food was bad, which makes it more likely that you will get sick also. So knowing my situation influences my beliefs about yours.
  - But, conditioned on knowing the food was bad (or not bad), my sickness and your sickness are independent.

More Tutorial Material

- Wikipedia's conditional probability article is good:
  - https://en.wikipedia.org/wiki/Conditional_probability

- Visual/interactive introduction to probability:
  - http://students.brown.edu/seeing-theory/basic-probability/index.html#first
  - http://students.brown.edu/seeing-theory/compound-probability/index.html#first

- "Probability Primer" (advanced, PP 1.S-5.4 are most relevant):
  - https://www.youtube.com/playlist?list=PL17567A1A3F5DB5E4

Fun with Probabilities

- Probabilities can be used for a huge variety of problems:
  - Are you the hottest person in your group?
  - Poker Odds
  - Are shy students likely to be math students?
  - Battleship
  - Should you put all your eggs/tickets in one basket/lottery?
  - The Price is Right

CPSC 340:
Machine Learning and Data Mining

Data Exploration

Fall 2019

This lecture roughly follow:
http://www-users.cs.umn.edu/~kumar/dmbook/dmslides/chap2_data.pdf

Admin

- Assignment 1 is due next Friday: start early.
  - Gradescope submission instructions will be posted soon-ish.
- Waiting list people: you should be registered soon-ish.
  - Start on the assignment now, everybody currently on the waiting list will get in.
- Bookmark the course webpage:
  - https://www.cs.ubc.ca/~schmidtm/Courses/340-F19/
- Tutorials and office hours start next week (see webpage for time).
- Sign up for the course Piazza group:
  - https://piazza.com/ubc.ca/winterterm12019/cpsc340/home
- 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.

Data Mining: Bird's Eye View

1) Collect data.
2) Data mining!
3) Profit?

Unfortunately, it's often more complicated…

Data Mining: Some Typical Steps

1) Learn about the application.
2) Identify data mining task.
3) Collect data.
4) Clean and preprocess the data.
5) Transform data or select useful subsets.
6) Choose data mining algorithm.
7) Data mining!
8) Evaluate, visualize, and interpret results.
9) Use results for profit or other goals.

(often, you'll go through cycles of the above)

Data Mining: Some Typical Steps

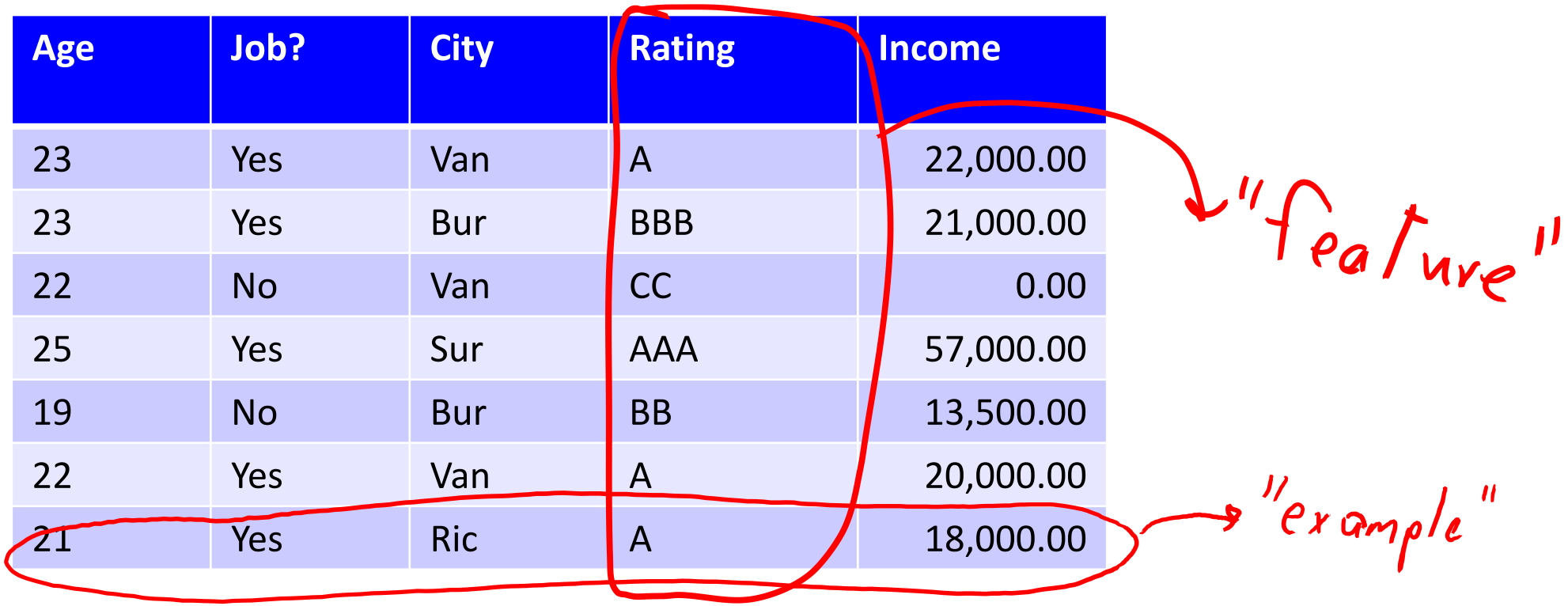
1) Learn about the application.
2) Identify data mining task.
3) Collect data.
4) Clean and preprocess the data.
5) Transform data or select useful subsets.
6) Choose data mining algorithm.
7) Data mining!
8) Evaluate, visualize, and interpret results.
9) Use results for profit or other goals.

(often, you'll go through cycles of the above)

What is Data?

- We'll define data as a collection of examples, and their features.

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
19	No	Bur	BB	13,500.00
22	Yes	Van	A	20,000.00
21	Yes	Ric	A	18,000.00



- Each row is an "example", each column is a "feature".
  - Examples are also sometimes called "samples".

Types of Data

- Categorical features come from an unordered set:
  - Binary: job?
  - Nominal: city.
- Numerical features come from ordered sets:
  - Discrete counts: age.
  - Ordinal: rating.
  - Continuous/real-valued: height.

Converting to Numerical Features

- Often want a real-valued example representation:

Age	City	Income
23	Van	22,000.00
23	Bur	21,000.00
22	Van	0.00
25	Sur	57,000.00
19	Bur	13,500.00
22	Van	20,000.00

Age	Van	Bur	Sur	Income
23	1	0	0	22,000.00
23	0	1	0	21,000.00
22	1	0	0	0.00
25	0	0	1	57,000.00
19	0	1	0	13,500.00
22	1	0	0	20,000.00

- This is called a "1 of k" encoding.
- We can now interpret examples as points in space:
  - E.g., first example is at (23,1,0,0,22000).

Approximating Text with Numerical Features

- Bag of words replaces document by word counts:

The **International Conference on Machine Learning** (ICML) is the leading international academic conference in machine learning

ICML	International	Conference	Machine	Learning	Leading	Academic
1	2	2	2	2	1	1

- Ignores order, but often captures general theme.
- You can compute a “distance” between documents.

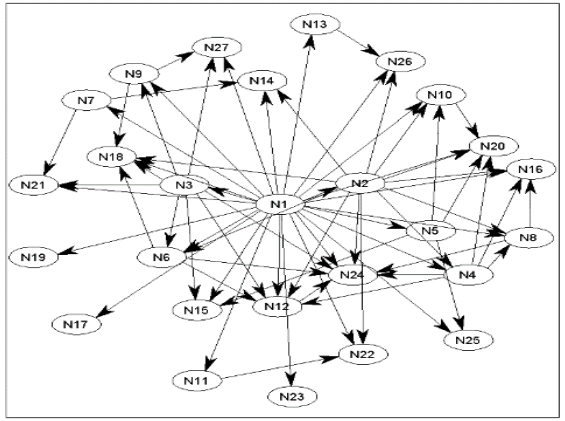
Approximating Images and Graphs

- We can think of other data types in this way:
  - Images:

graycale intensity

(1,1)	(2,1)	(3,1)	…	(m,1)	…	(m,n)
45	44	43	…	12	…	35

  - Graphs:



adjacency matrix

N1	N2	N3	N4	N5	N6	N7
0	1	1	1	1	1	1
0	0	0	1	0	1	0
0	0	0	0	0	1	0
0	0	0	0	0	0	0

Data Cleaning

- ML+DM typically assume ‘clean’ data.
- Ways that data might not be ‘clean’:
  - Noise (e.g., distortion on phone).
  - Outliers (e.g., data entry or instrument error).
  - Missing values (no value available or not applicable)
  - Duplicated data (repetitions, or different storage formats).
- Any of these can lead to problems in analyses.
  - Want to fix these issues, if possible.
  - Some ML methods are robust to these.
  - Often, ML is the best way to detect/fix these.

The Question I Hate the Most…

- How much data do we need?

- A difficult if not impossible question to answer.

- My usual answer: “more is better”.
  - With the warning: “as long as the quality doesn’t suffer”.

- Another popular answer: “ten times the number of features”.

A Simple Setting: Coupon Collecting

- Assume we have a categorical variable with 50 possible values:
  - {Alabama, Alaska, Arizona, Arkansas,…}.
- Assume each category has probability of 1/50 of being chosen:
  - How many examples do we need to see before we expect to see them all?
- Expected value is ~225.
- Coupon collector problem: O(n log n) in general.
  - Gotta Catch'em all!
- Obvious sanity check, is need more samples than categories:
  - Situation is worse if they don't have equal probabilities.
  - Typically want to see categories more than once to learn anything.

Feature Aggregation

- Feature aggregation:
  - Combine features to form new features:

Van	Bur	Sur	Edm	Cal
1	0	0	0	0
0	1	0	0	0
1	0	0	0	0
0	0	0	1	0
0	0	0	0	1
0	0	1	0	0

BC	AB
1	0
1	0
1	0
0	1
0	1
1	0

- Fewer province “coupons” to collect than city “coupons”.

Feature Selection

- Feature Selection:
  - Remove features that are not relevant to the task.

SID:	Age	Job?	City	Rating	Income
3457	23	Yes	Van	A	22,000.00
1247	23	Yes	Bur	BBB	21,000.00
6421	22	No	Van	CC	0.00
1235	25	Yes	Sur	AAA	57,000.00
8976	19	No	Bur	BB	13,500.00
2345	22	Yes	Van	A	20,000.00

  - Student ID is probably not relevant.

Feature Transformation

- Mathematical transformations:
  - Discretization (binning): turn numerical data into categorical.

Age
23
23
22
25
19
22

< 20	>= 20, < 25	>= 25
0	1	0
0	1	0
0	1	0
0	0	1
1	0	0
0	1	0

  - Only need consider 3 values.

Feature Transformation

- Mathematical transformations:
  - Discretization (binning): turn numerical data into categorical.
  - Square, exponentiation, logarithm, and so on.

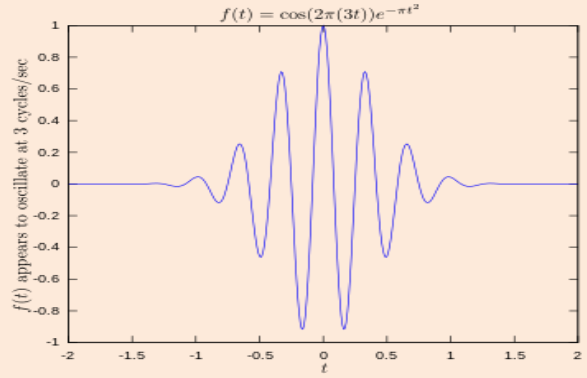
www.google.com/finance

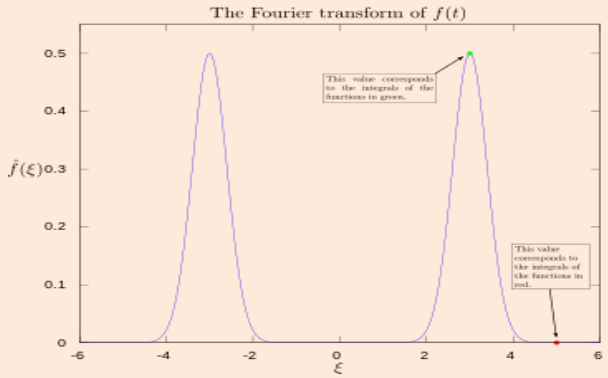
Feature Transformation

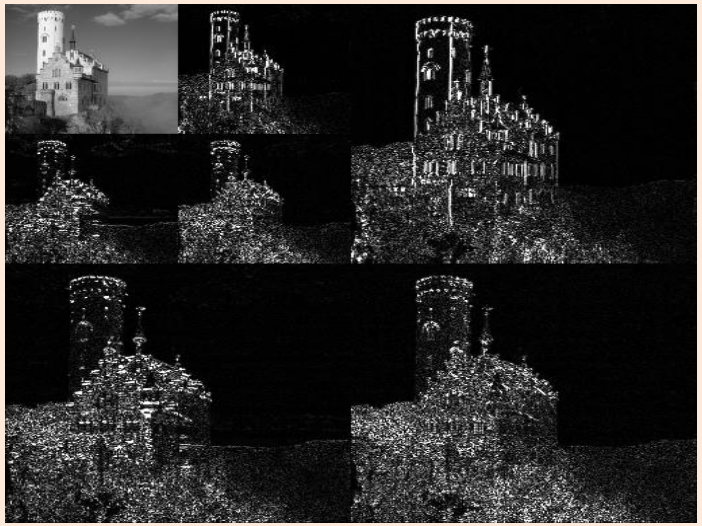
- Mathematical transformations:
  - Discretization (binning): turn numerical data into categorical.
  - Square, exponentiation, or take logarithm.
  - Scaling: convert variables to comparable scales
    (E.g., convert kilograms to grams.)

Feature Transformation

- Mathematical transformations:
  - Discretization (binning): turn numerical data into categorical.
  - Square, exponentiation, or take logarithm.
  - Scaling: convert variables to comparable scales.
  - Fourier coefficients, spectrograms, and wavelets (signal data).







https://en.wikipedia.org/wiki/Fourier_transform
https://en.wikipedia.org/wiki/Spectrogram
https://en.wikipedia.org/wiki/Discrete_wavelet_transform

Links to figure sources will be here.

(pause)

Exploratory Data Analysis

- You should always ‘look’ at the data first.

- But how do you ‘look’ at features and high-dimensional examples?
  - Summary statistics.
  - Visualization.
  - ML + DM (later in course).

Categorical Summary Statistics

- Summary statistics for a categorical feature:
  - Frequencies of different classes.
  - Mode: category that occurs most often.
  - Quantiles: categories that occur more than t times.

Population by year, by province and territory (Number)

	2014
Canada	**35,540.4**
Newfoundland and Labrador	527.0
Prince Edward Island	146.3
Nova Scotia	942.7
New Brunswick	753.9
Quebec	8,214.7
Ontario	13,678.7
Manitoba	1,282.0
Saskatchewan	1,125.4
Alberta	4,121.7
British Columbia	4,631.3
Yukon	36.5
Northwest Territories	43.6
Nunavut	36.6

Frequency: 13.3% of Canadian residents live in BC.
Mode: Ontario has largest number of residents (38.5%)
Quantile: 6 provinces have more than 1 million people.

Continuous Summary Statistics

- Measures of location for continuous features:
  - Mean: average value.
  - Median: value such that half points are larger/smaller.
  - Quantiles: value such that ‘k’ fraction of points are larger.

- Measures of spread for continuous features:
  - Range: minimum and maximum values.
  - Variance: measures how far values are from mean.
    - Square root of variance is “standard deviation”.
  - Intequantile ranges: difference between quantiles.

Continuous Summary Statistics

- Data: [0 1 2 3 3 5 7 8 9 10 14 15 17 200]
- Measures of location:
  - Mean(Data) = 21
  - Mode(Data) = 3
  - Median(Data) = 7.5
  - Quantile(Data,0.5) = 7.5
  - Quantile(Data,0.25) = 3
  - Quantile(Data,0.75) = 14
- Measures of spread:
  - Range(Data) = [0 200].
  - Std(Data) = 51.79
  - IQR(Data,.25,.75) = 11
- Notice that mean and std are more sensitive to extreme values ("outliers").

Entropy as Measure of Randomness

- Another common summary statistic is entropy.
  - Entropy measures "randomness" of a set of variables.
    - Roughly, another measure of the "spread" of values.
    - Formally, "how many bits of information are encoded in the average example".
  - For a categorical variable that can take 'k' values, entropy is defined by:

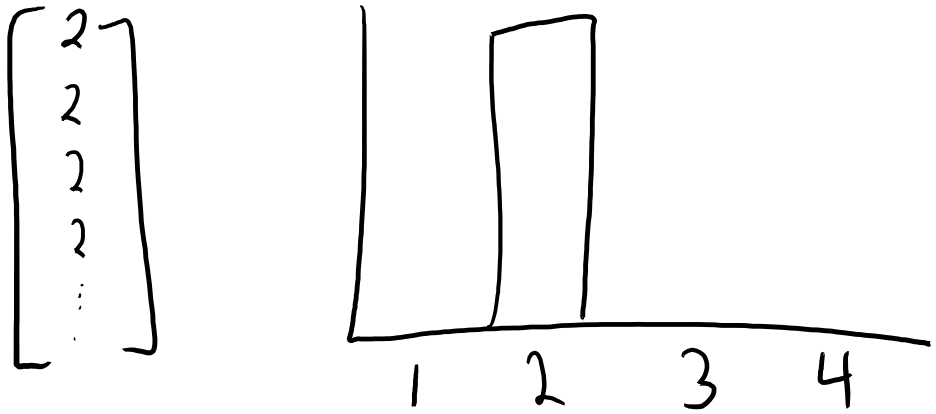
$$\text{entropy} = -\sum_{c=1}^{k} p_c \log p_c$$

where p_c is the proportion of times you have value 'c'.

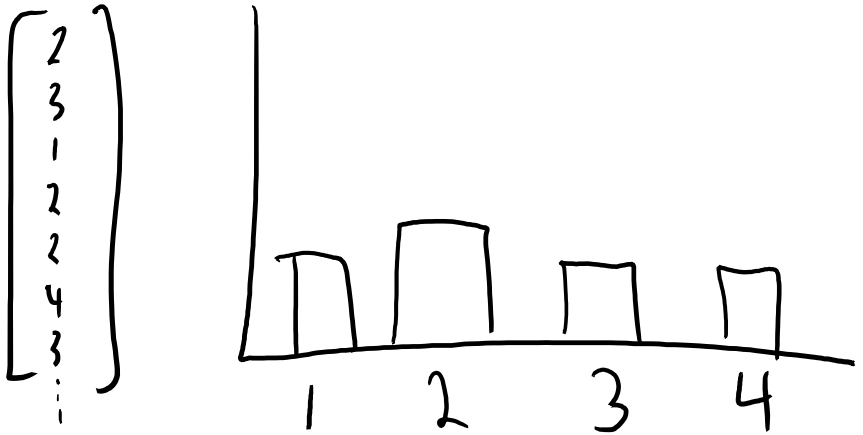
  - Low entropy means "very predictable".
  - High entropy means "very random".
  - Minimum value is 0, maximum value is log(k).
    - We use the convention that 0 log 0 = 0.

Entropy as Measure of Randomness

Low entropy means “very predictable”



High entropy means “very random”



- For categorical features: uniform distribution has highest entropy.
- For continuous densities with fixed mean and variance:
  - Normal distribution has highest entropy (not obvious).
- Entropy and Dr. Seuss (words like “snunkoople” increase entropy).

Distances and Similarities

- There are also summary statistics between features ‘x’ and ‘y’.
  - Hamming distance:
    - Number of elements in the vectors that aren’t equal.
  - Euclidean distance:
    - How far apart are the vectors?
  - Correlation:
    - Does one increase/decrease linearly as the other increases?
    - Between -1 and 1.

x	y
0	0
0	0
1	0
0	1
0	1
1	1
0	0
0	1
0	1

Distances and Similarities

- There are also summary statistics between features ‘x’ and ‘y’.
  - Rank correlation:
    - Does one increase/decrease as the other increases?
      - Not necessarily in a linear way.

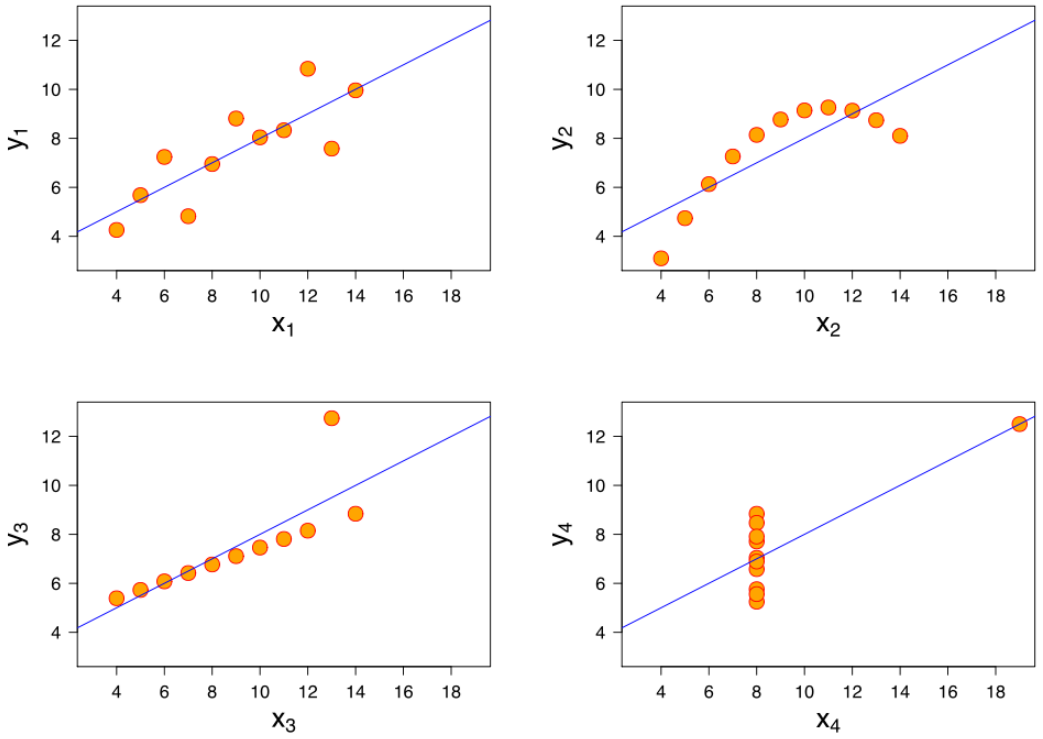
x	y
0	0
0	0
1	0
0	1
0	1
1	1
0	0
0	1
0	1

- Distances/similarities between other types of data:
  - Jaccard coefficient (distance between sets):
    - (size of intersection of sets) / (size of union of sets)
  - Edit distance (distance between strings):
    - How many characters do we need to change to go from x to y?
    - Computed using dynamic programming (CPSC 320).

Limitations of Summary Statistics

- On their own summary statistic can be misleading.
- Why not to trust statistics

- Amcomb's quartet:
  - Almost same means.
  - Almost same variances.
  - Almost same correlations.
  - Look completely different.
- Datasaurus dozen.



https://en.wikipedia.org/wiki/Anscombe%27s_quartet

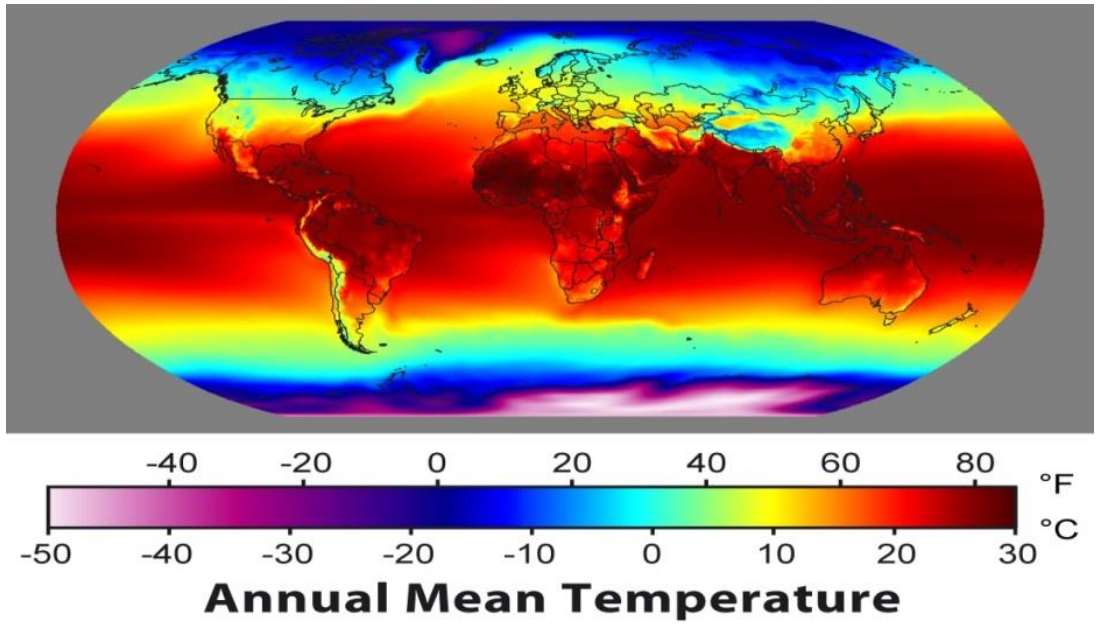
(pause)

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
…	…	…

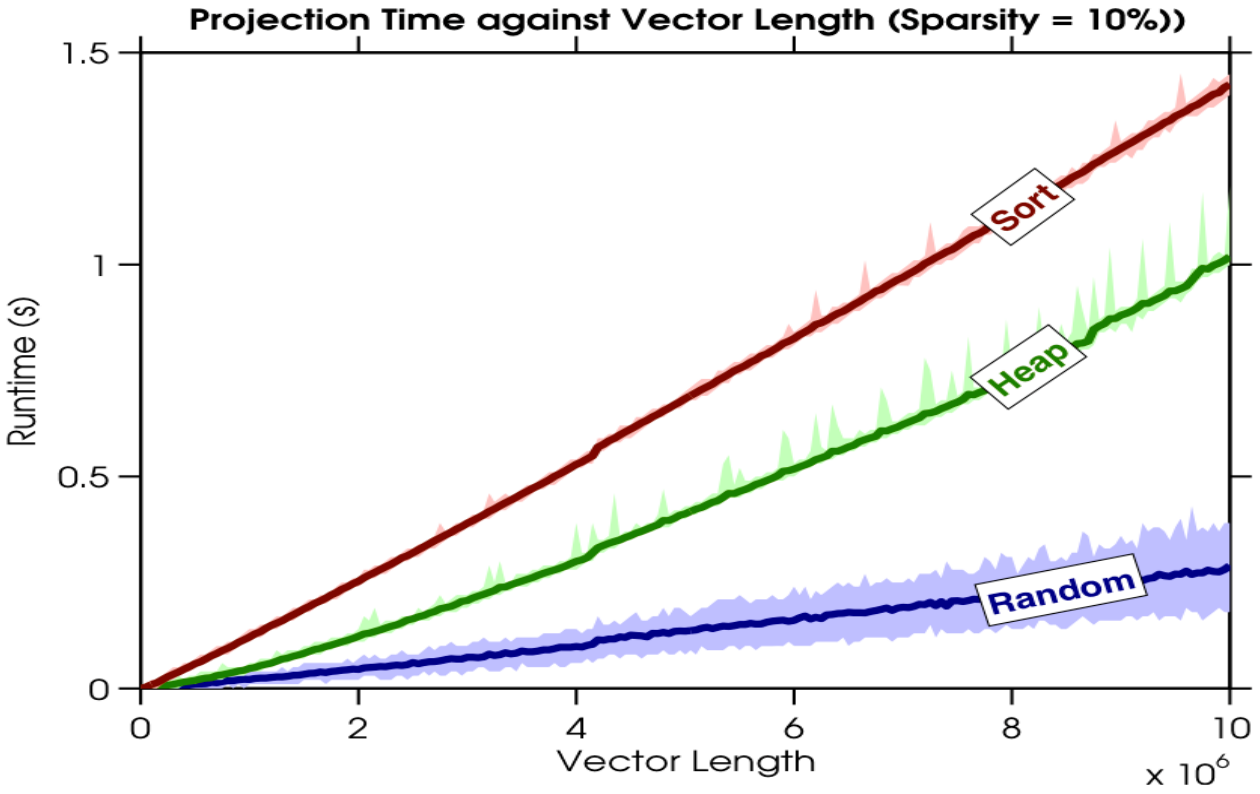
vs.

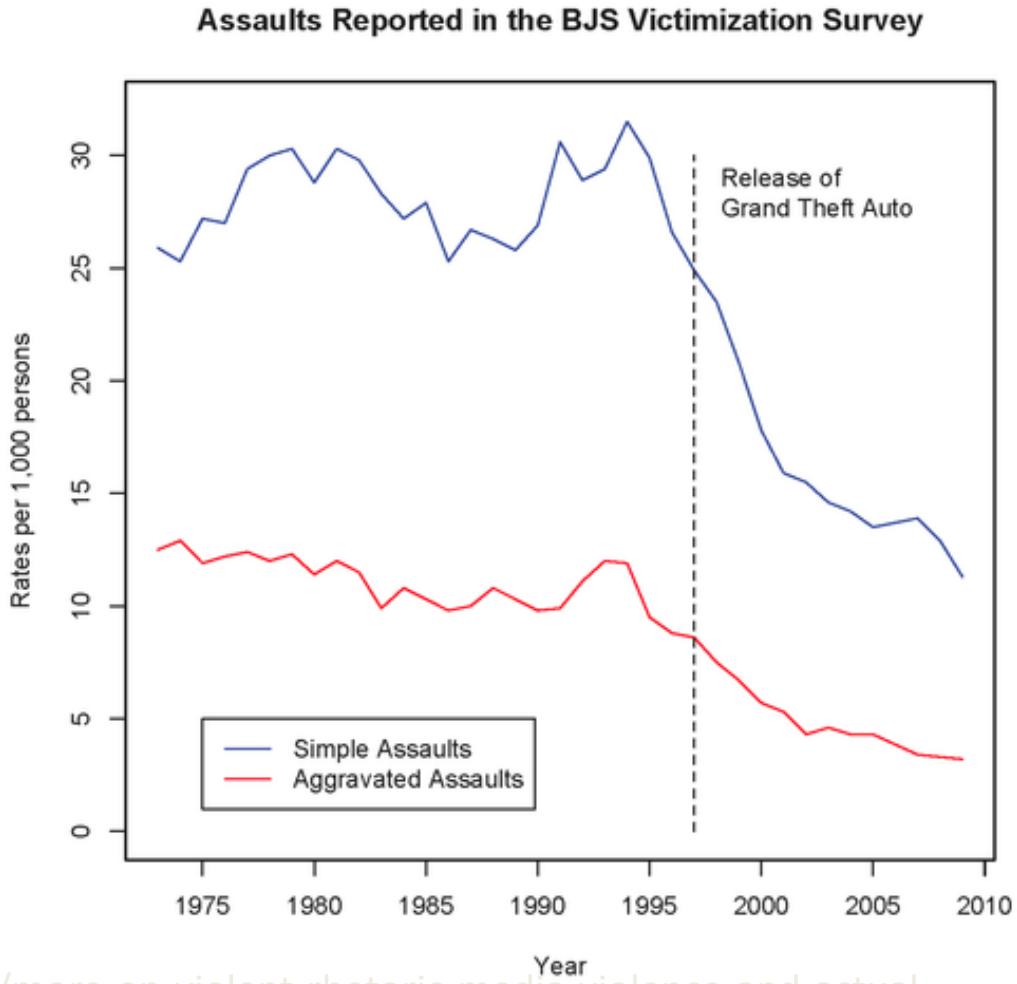


https://en.wikipedia.org/wiki/Temperature

Basic Plot

- Visualize one variable as a function of another.

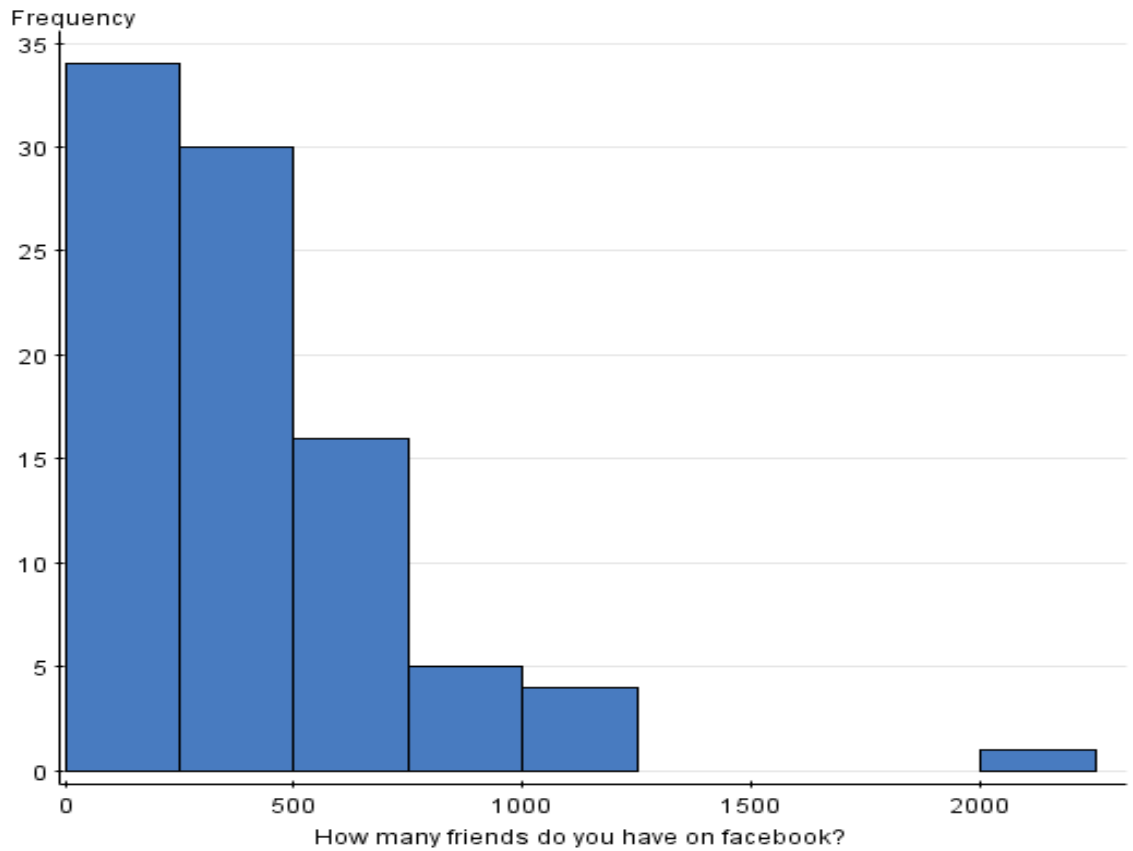




- [Fun with plots](http://notunlikeresearch.typepad.com/something-not-unlike-rese/2011/01/more-on-violent-rhetoric-media-violence-and-actual-).

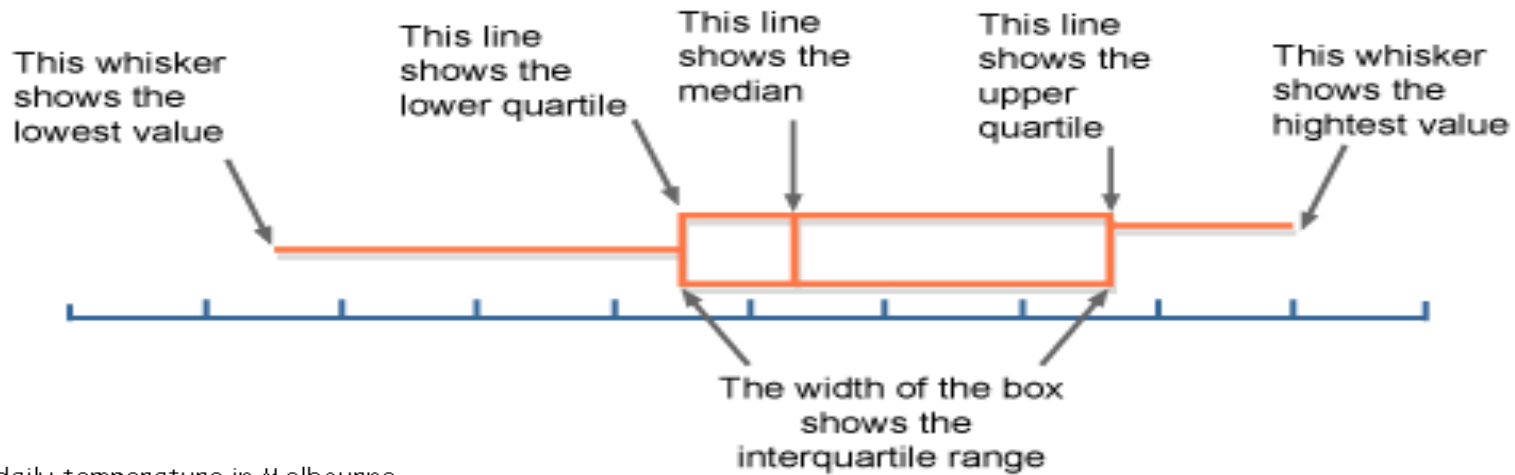
Histogram

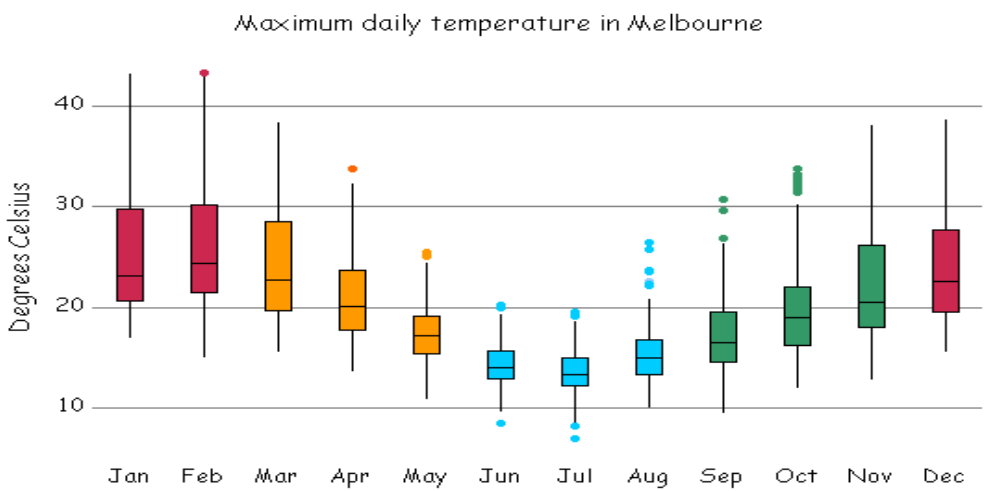
- Histograms display distribution of a variable.

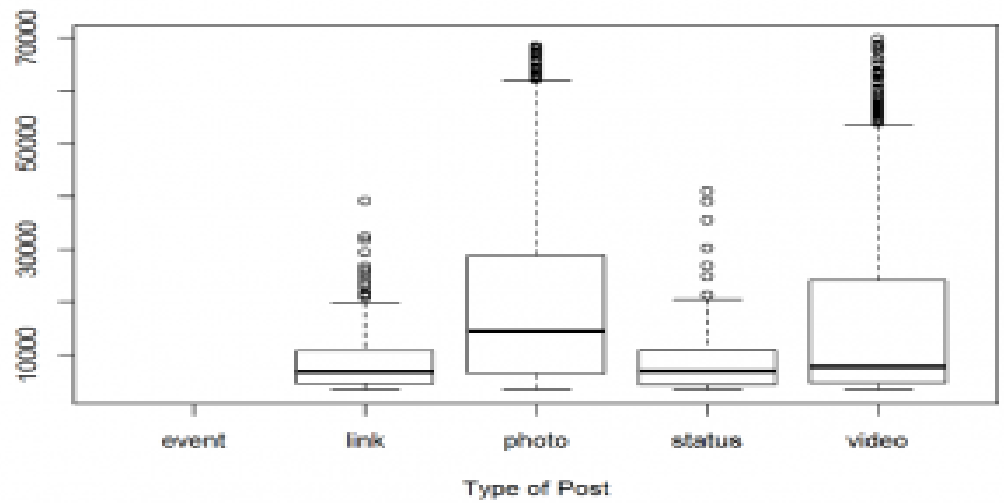


http://www.statcrunch.com/5.0/viewresult.php?resid=1024581

Box Plot







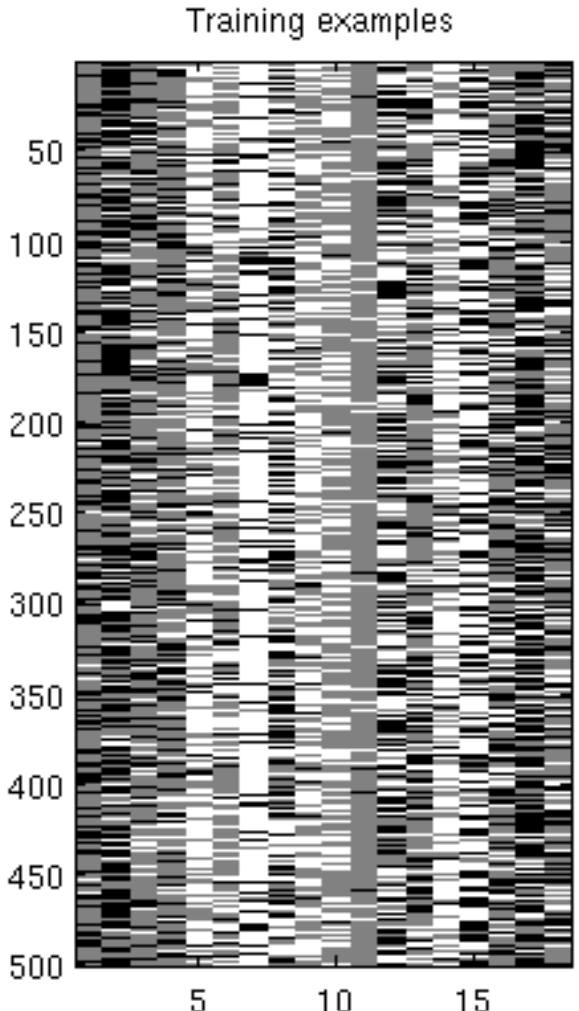
http://www.bbc.co.uk/schools/gcsebitesize/maths/statistics/representingdata3hi
http://www.scc.ms.unimelb.edu.au/whatisstatistics/weather.html
http://r.ramganalytics.com/r/facebook-likes-and-analytics/

Box Plot

- Photo from CTV Olympic coverage in 2010:

Matrix Plot

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



Matrix Plot

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

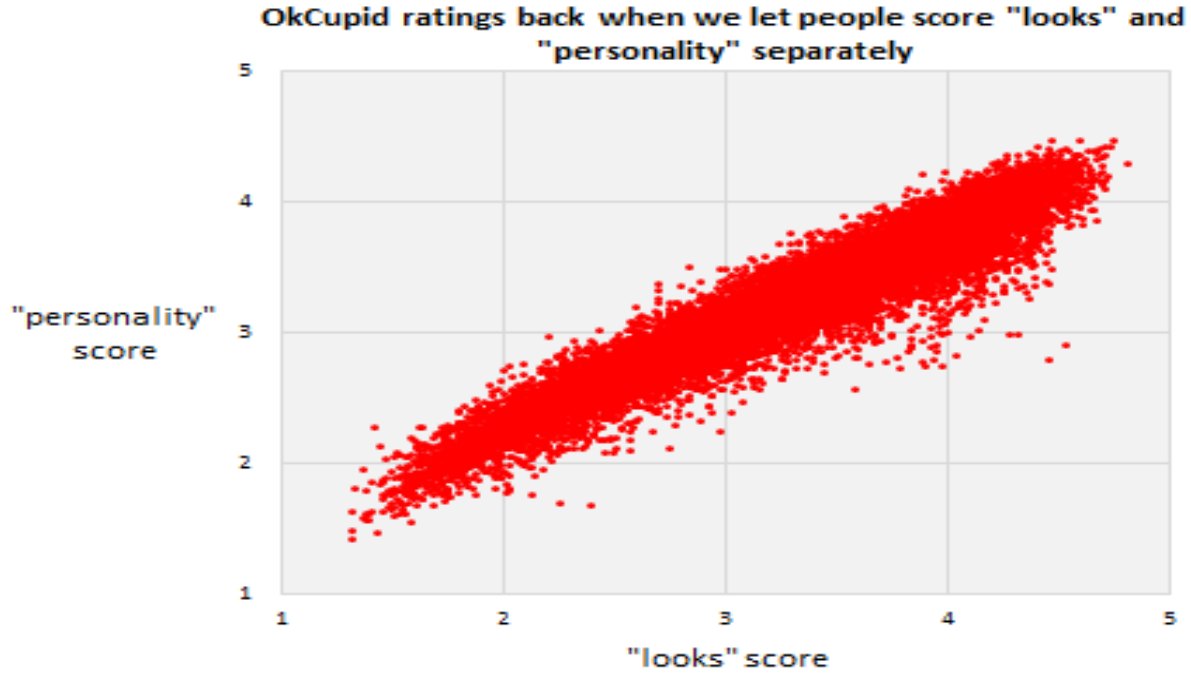
	BTC	ETH	XRP	XEM	ETC	LTC	DASH	XMR
BTC	1.00	0.61	0.36	0.51	0.60	0.56	0.55	0.66
ETH	0.61	1.00	0.28	0.49	0.68	0.43	0.70	0.64
XRP	0.36	0.28	1.00	0.48	0.08	0.35	0.40	0.44
XEM	0.51	0.49	0.48	1.00	0.40	0.43	0.47	0.52
ETC	0.60	0.68	0.08	0.40	1.00	0.47	0.56	0.53
LTC	0.56	0.43	0.35	0.43	0.47	1.00	0.59	0.67
DASH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

"Correlation plot"

https://www.crypto-news.net/portfolio-diversification-with-low-correlation-cryptos/

Scatterplot

- Look at distribution of two features:
  - Feature 1 on x-axis.
  - Feature 2 on y-axis.
  - Basically a “plot without lines” between the points.

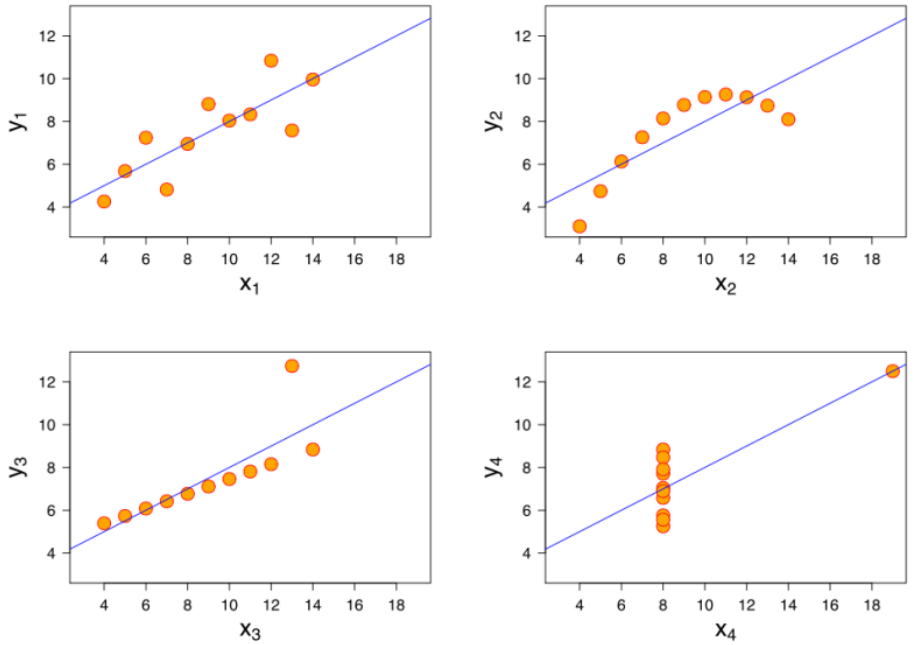


- Shows correlation between “personality” score and “looks” score.

http://cdn.okccdn.com/blog/humanexperiments/looks-v-personality.png

Scatterplot

- Look at distribution of two features:
  - Feature 1 on x-axis.
  - Feature 2 on y-axis.
  - Basically a “plot without lines” between the points.



- Shows correlation between “personality” score and “looks” score.
- But scatterplots let you see more complicated patterns.

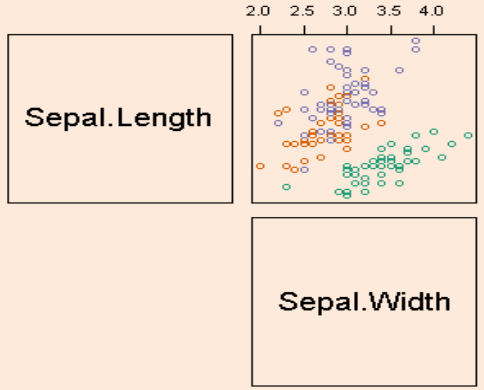
https://en.wikipedia.org/wiki/Anscombe%27s_quartet

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.

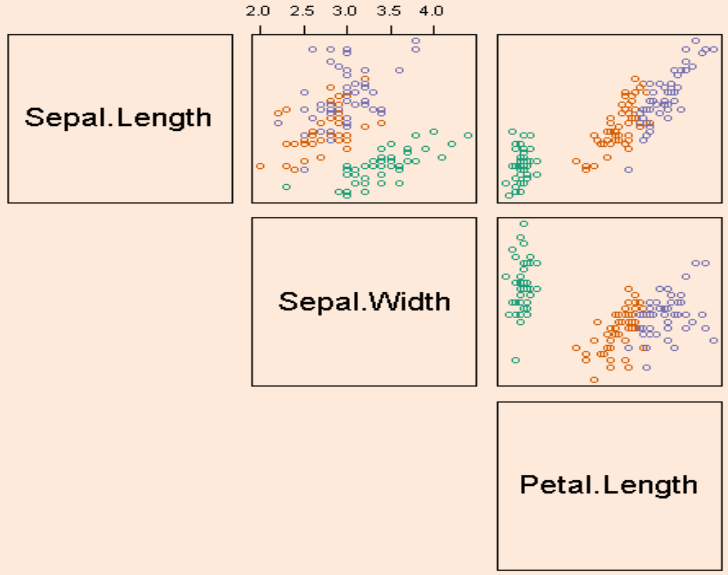
https://en.wikipedia.org/wiki/Iris_flower_data_set
http://www.ats.ucla.edu/stat/r/pages/layout.htm

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.

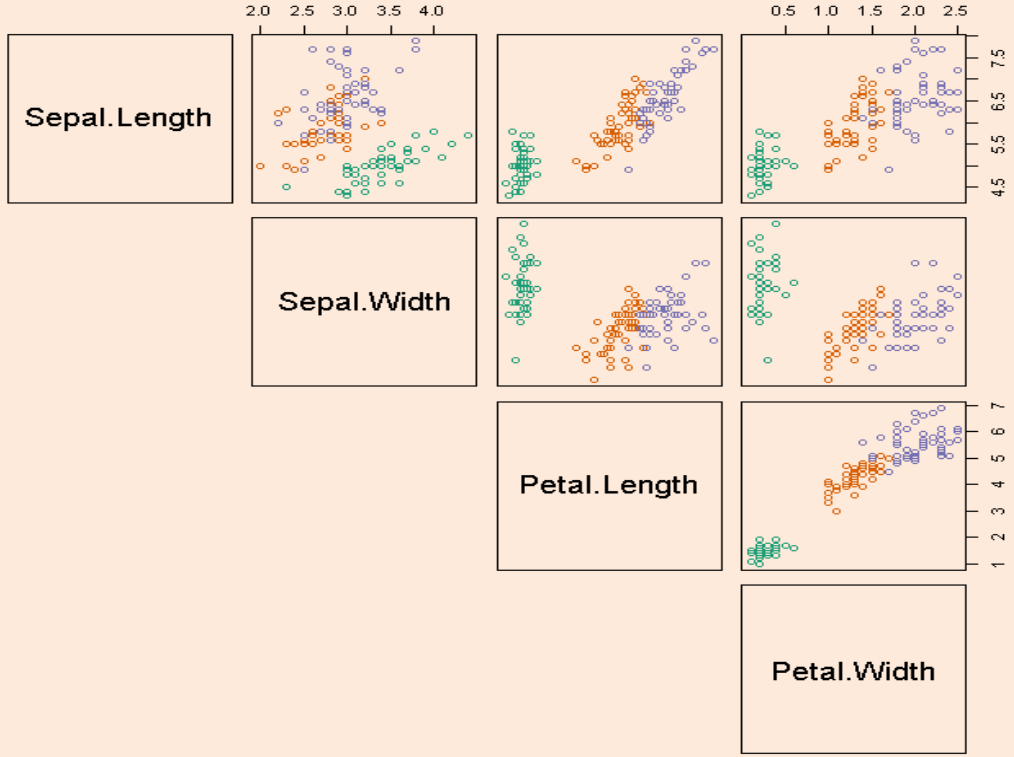
https://en.wikipedia.org/wiki/Iris_flower_data_set
http://www.ats.ucla.edu/stat/r/pages/layout.htm

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.

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

“Why Not to Trust Plots”

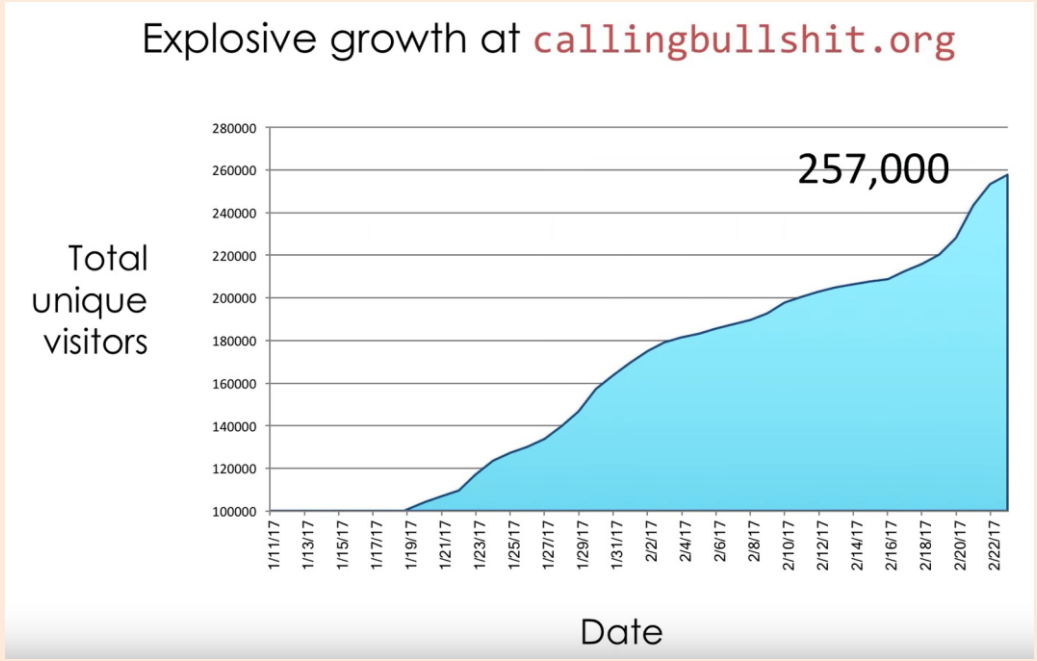
- We’ve seen how summary statistics can be mis-leading.
- Note that plots can also be mis-leading, or can be used to mis-lead.

- 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-4CSZD7bUI4gSPS

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

Mis-Leading Axes

- This plot seems to show amazing recent growth:

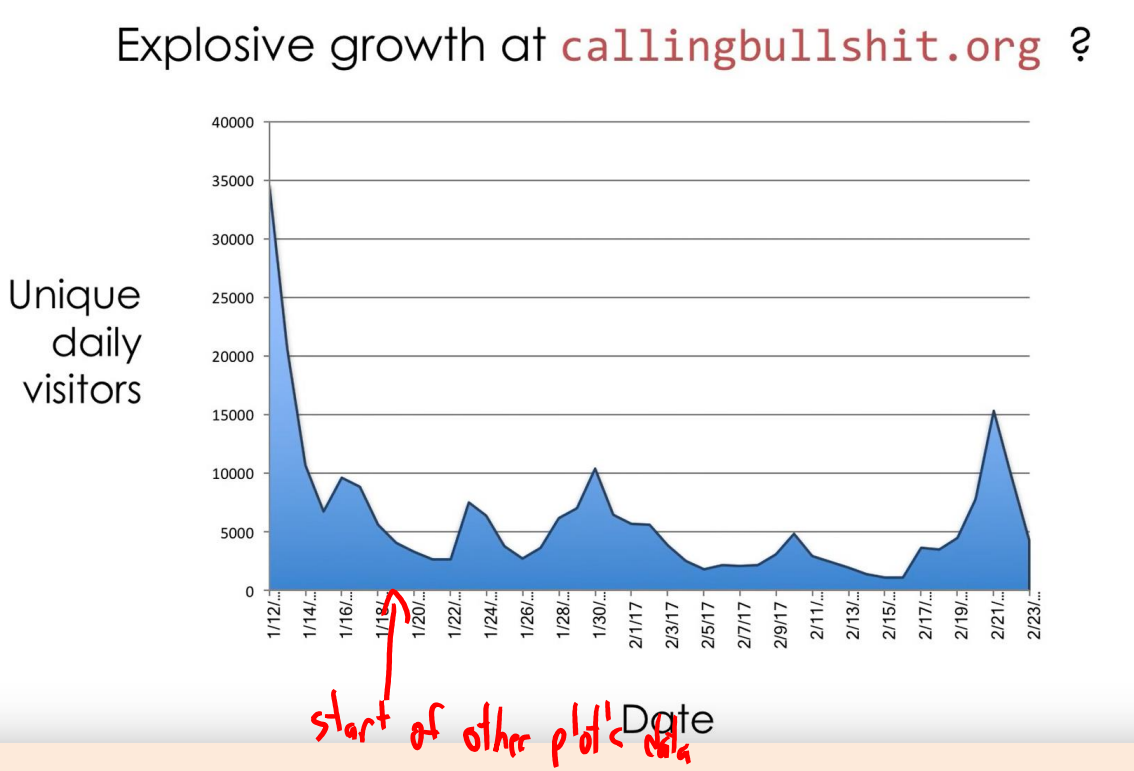


- But notice y-axis starts at 100,000 (so ~40% of growth was earlier).
- And it plots "total" users (which necessarily goes up).

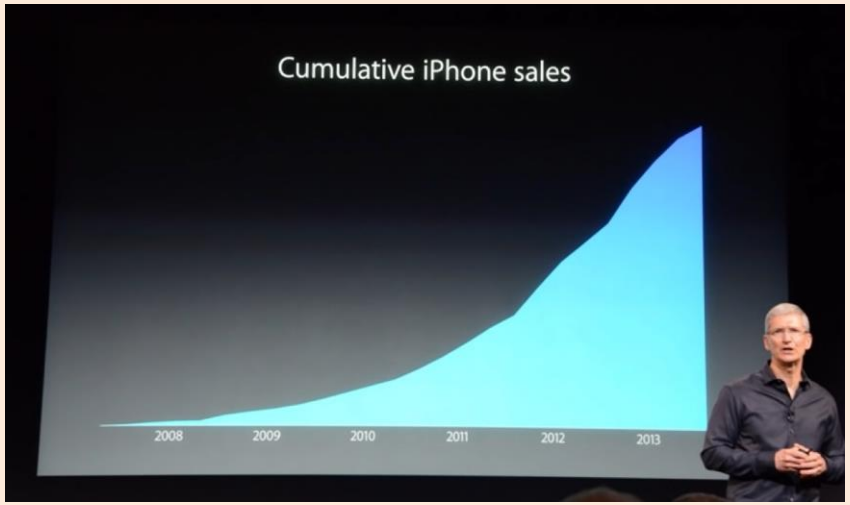
https://www.youtube.com/watch?v=q94VJ3KToK8

Mis-Leading Axes

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



- People can mis-lead to push agendas/stories:



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

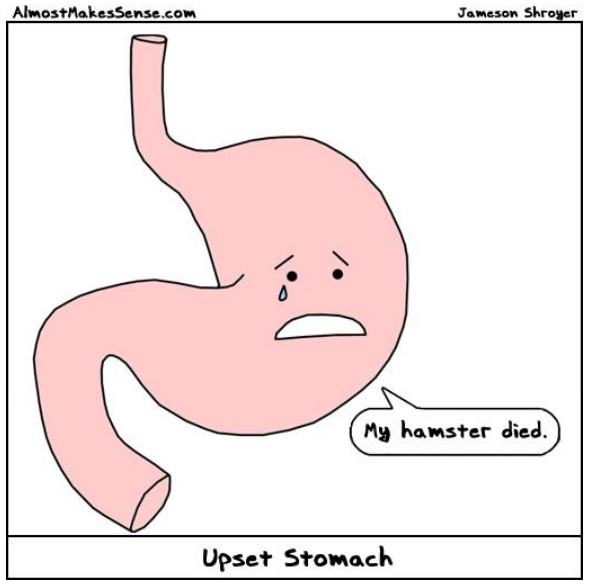
Mis-Leading Axes

- 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.

(pause)

Motivating Example: Food Allergies

- You frequently start getting an upset stomach



- You suspect an adult-onset food allergy.

http://www.cliparthut.com/upset-stomach-clipart-cn48e5.html

Motivating Example: Food Allergies

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

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

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

Supervised Learning

- We can formulate this as supervised learning:

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

- Input for an 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.
  - Model predicts whether foods will make you sick (even with new combinations).

Supervised Learning

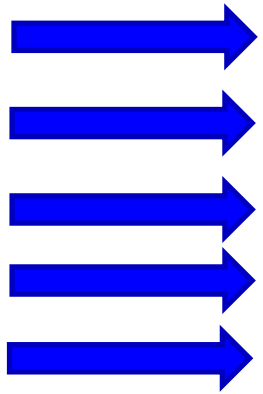
- General supervised learning problem:
  - Take features of examples and corresponding labels as inputs.
  - Find a model that can accurately ***predict the labels of new examples***.

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

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

Naïve Supervised Learning: "Predict Mode"

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



- A very naïve supervised learning method:
  - Count how many times each label occurred in the data (4 vs. 1 above).
  - Always predict the most common label, the "mode" ("sick" above).
- This ignores the features, so is only accurate if we only have 1 label.
- We want to use the features, and there are MANY ways to do this.
  - Next time we'll consider a classic way known as decision tree learning.

Summary

- Typical data mining steps:
  - Involves data collection, preprocessing, analysis, and evaluation.
- Example-feature representation and categorical/numerical features.
  - Transforming non-vector examples to vector representations.
- Feature transformations:
  - To address coupon collecting or simplify relationships between variables.
- Exploring data:
  - Summary statistics and data visualization.
- Supervised learning:
  - Using data to write a program based on input/output examples.

- Post-lecture bonus slides: other visualizations, parallel/distributed calculations.

- Next week: let's start some machine learning…

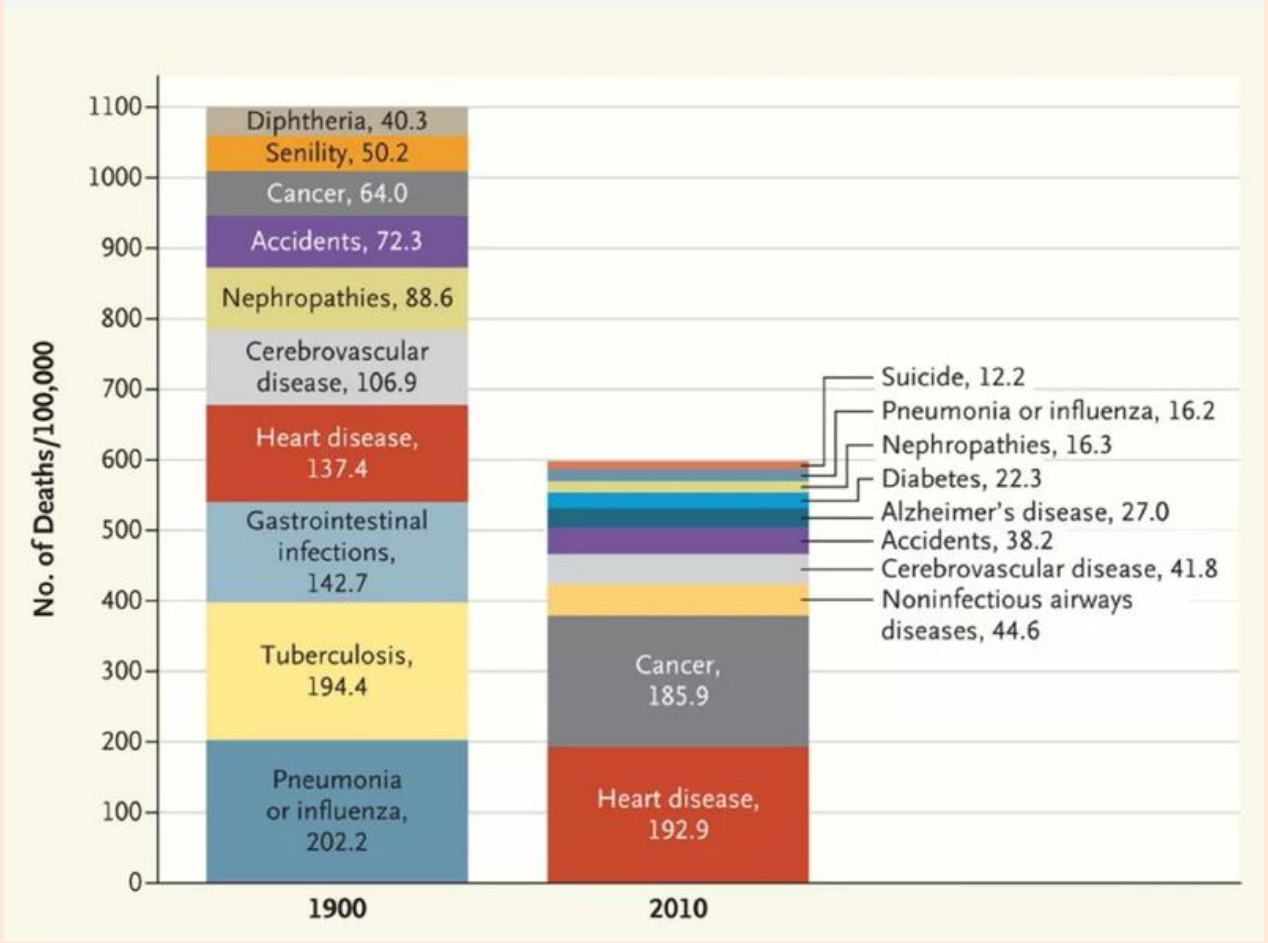
Data Cleaning and the Duke Cancer Scandal

- See the Duke cancer scandal:
  - http://www.nytimes.com/2011/07/08/health/research/08genes.html?_r=2&hp

- Basic sanity checks for data cleanliness show problems in these (and many other) studies:
  - E.g., flipped labels, off-by-one mistakes, switched columns etc.
  - https://arxiv.org/pdf/1010.1092.pdf

Histogram

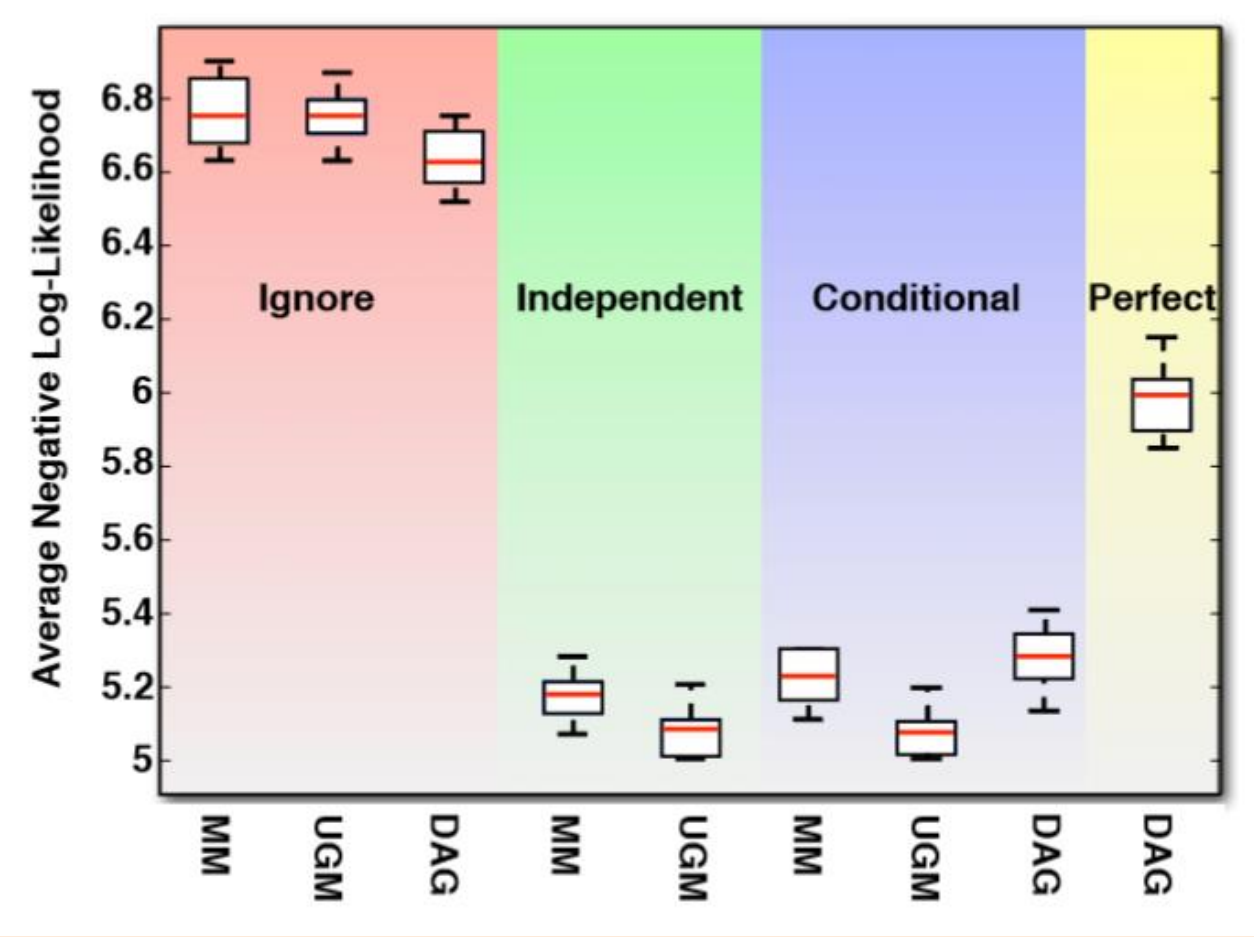
- Histogram with grouping:



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

Box Plots

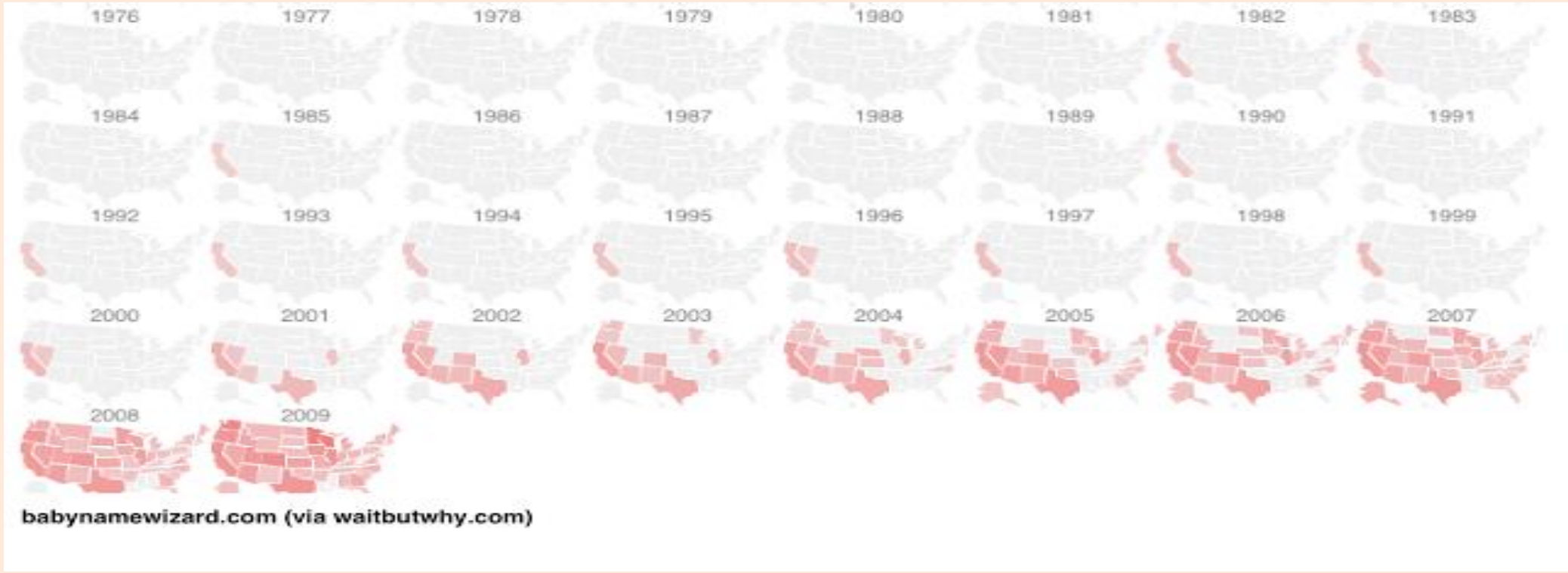
- Box plot with grouping:



Map Coloring

- Color/intensity can represent feature of region.

Popularity of naming baby “Evelyn” over time:



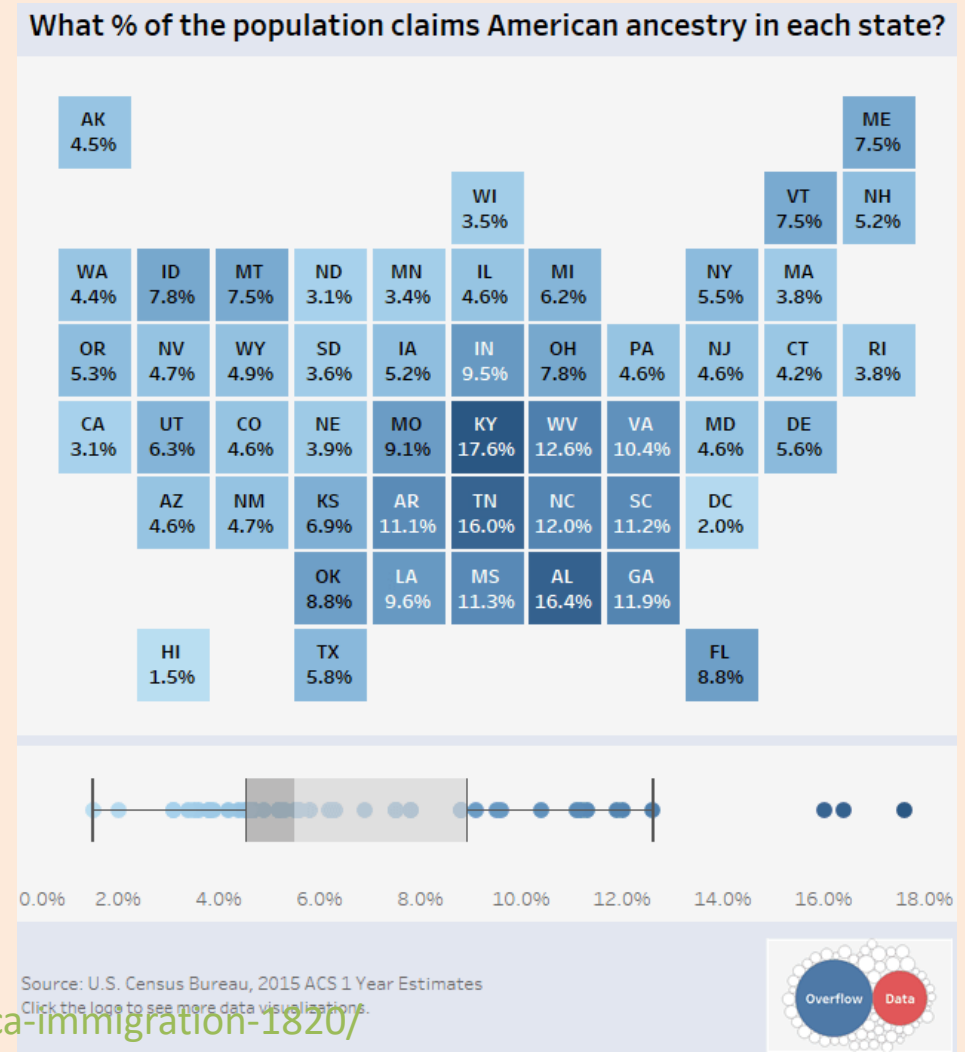
But not very good if some regions are very small.

[Canadian Income Mobility]

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

Map Coloring

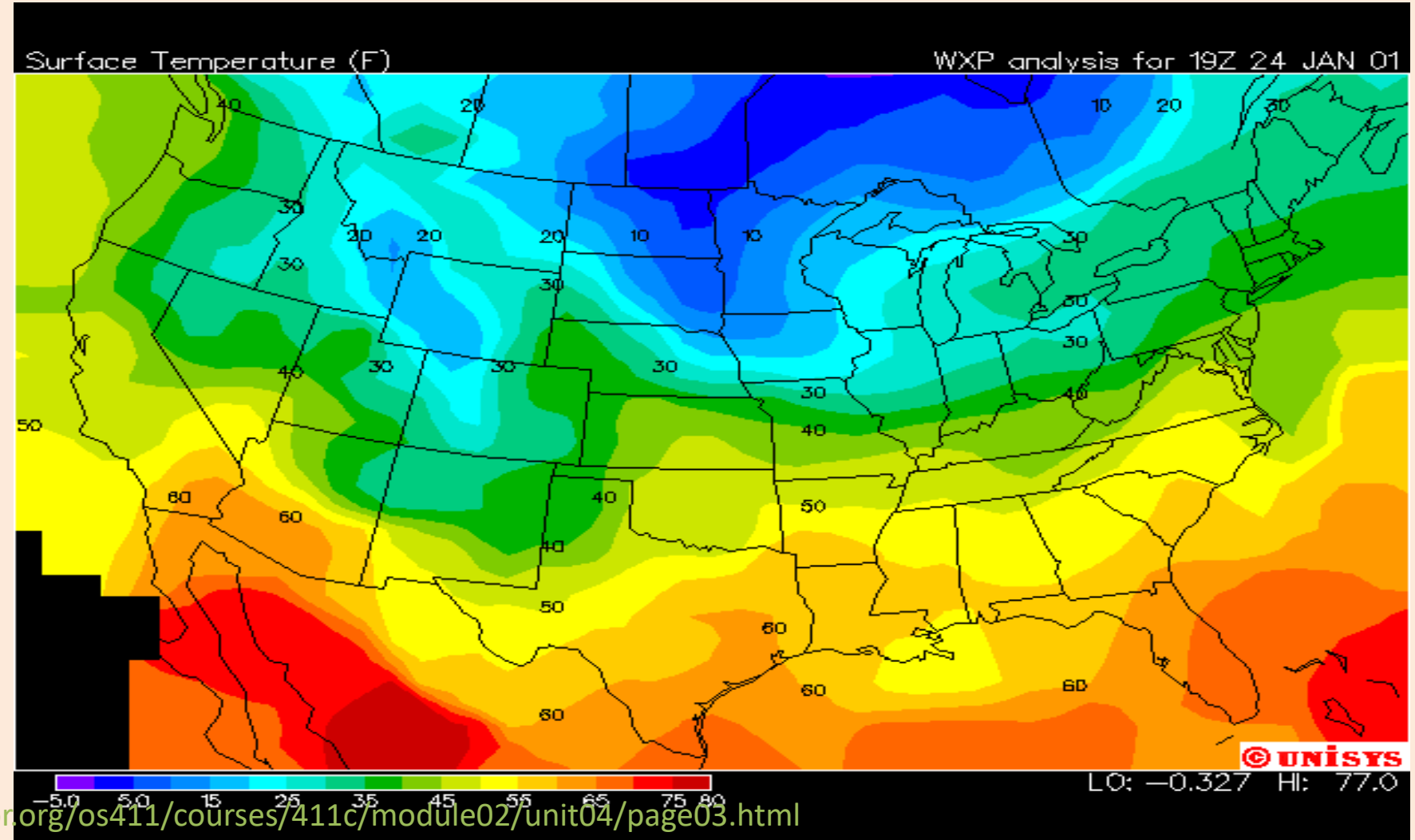
- Variation just uses fixed-size blocks and tries to arrange geographically:



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

Contour Plot

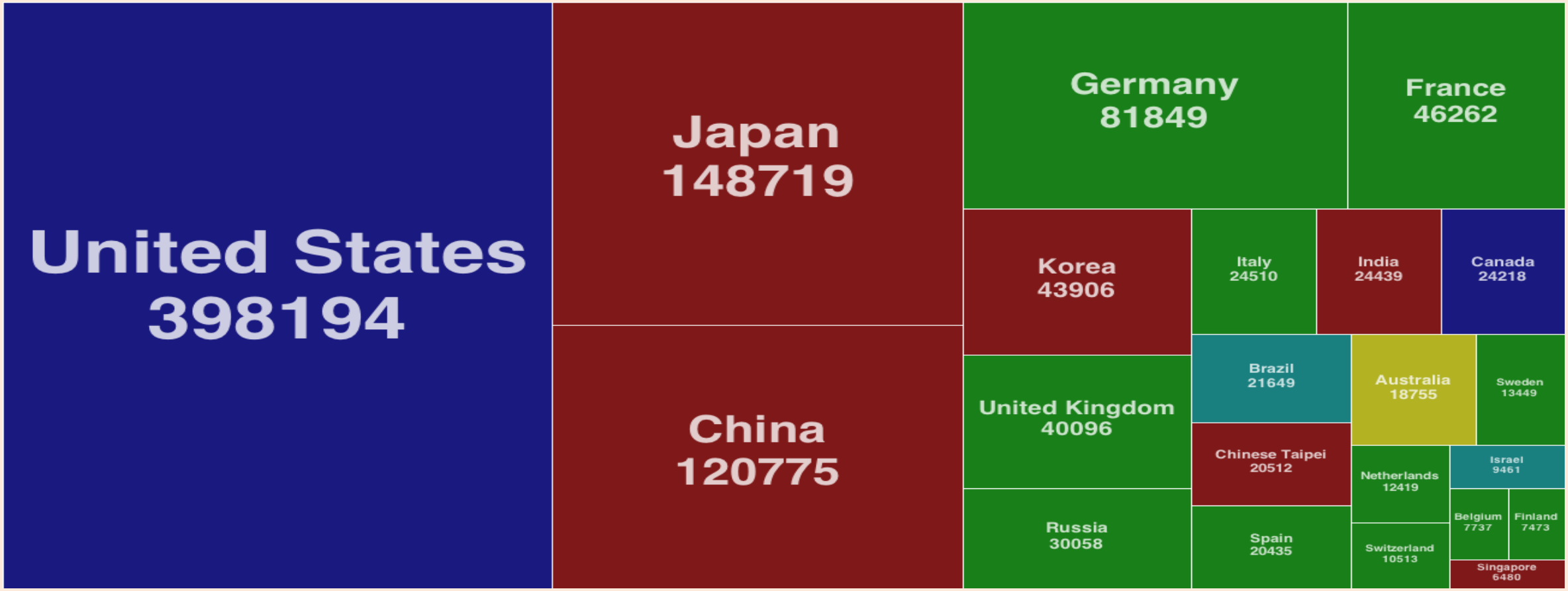
- Colour visualizes ‘z’ as we vary ‘x’ and ‘y’.



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

Treemaps

- Area represents attribute value:



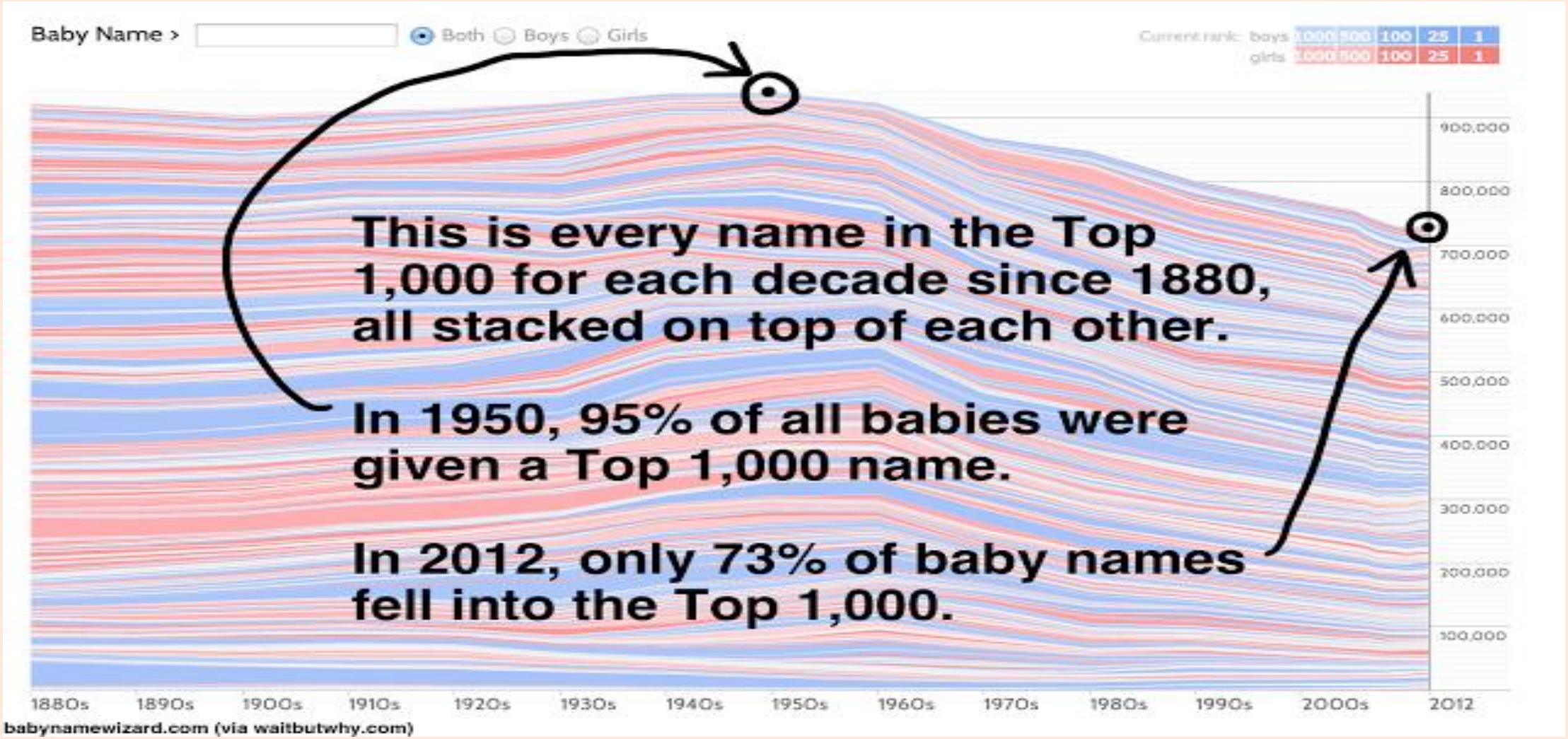
http://mjwest.org/vis/treemaps/

Cartogram

- Fancier version of treemaps:

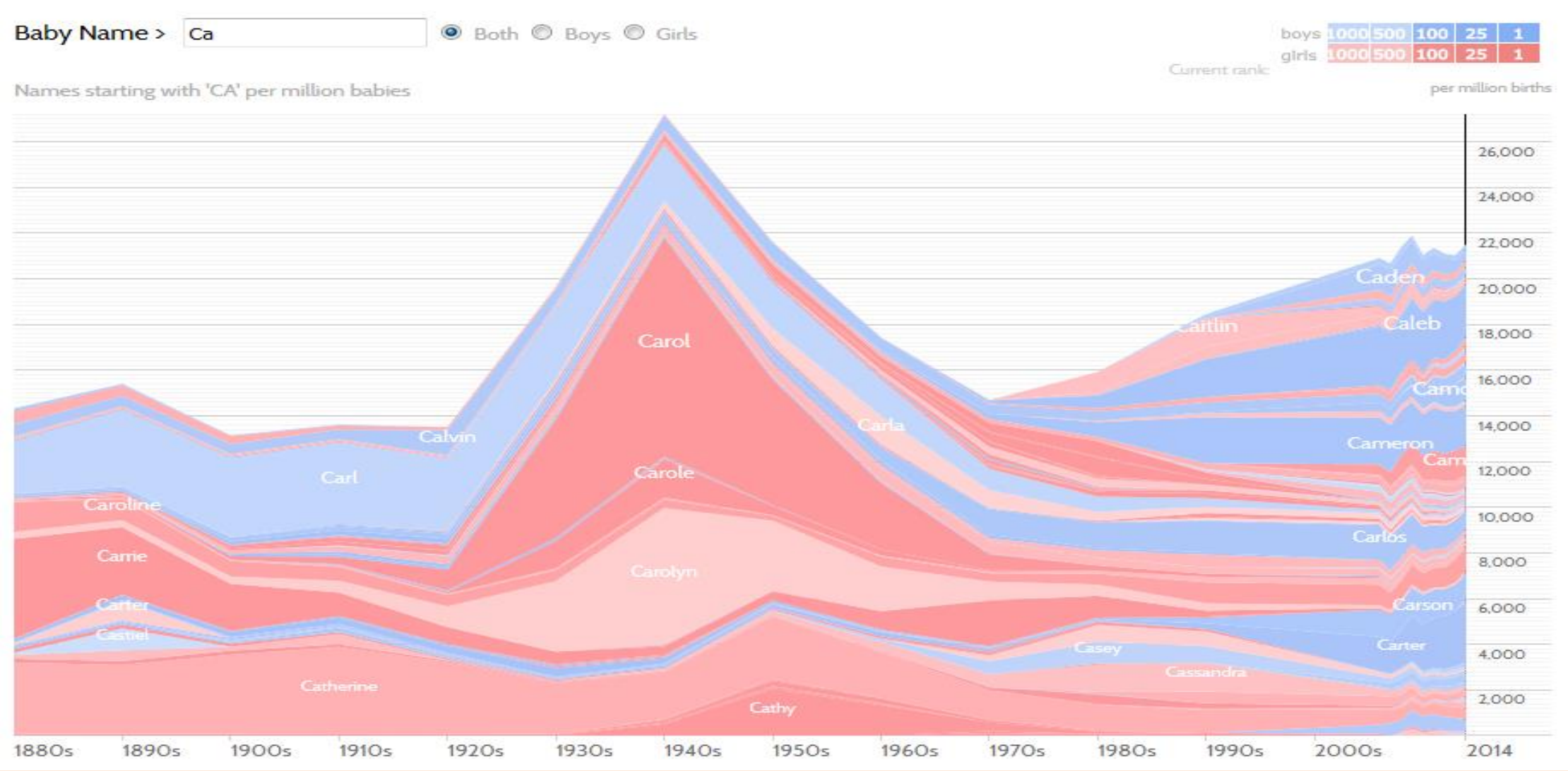
http://www-personal.umich.edu/~mejn/cartograms/

Stream Graph



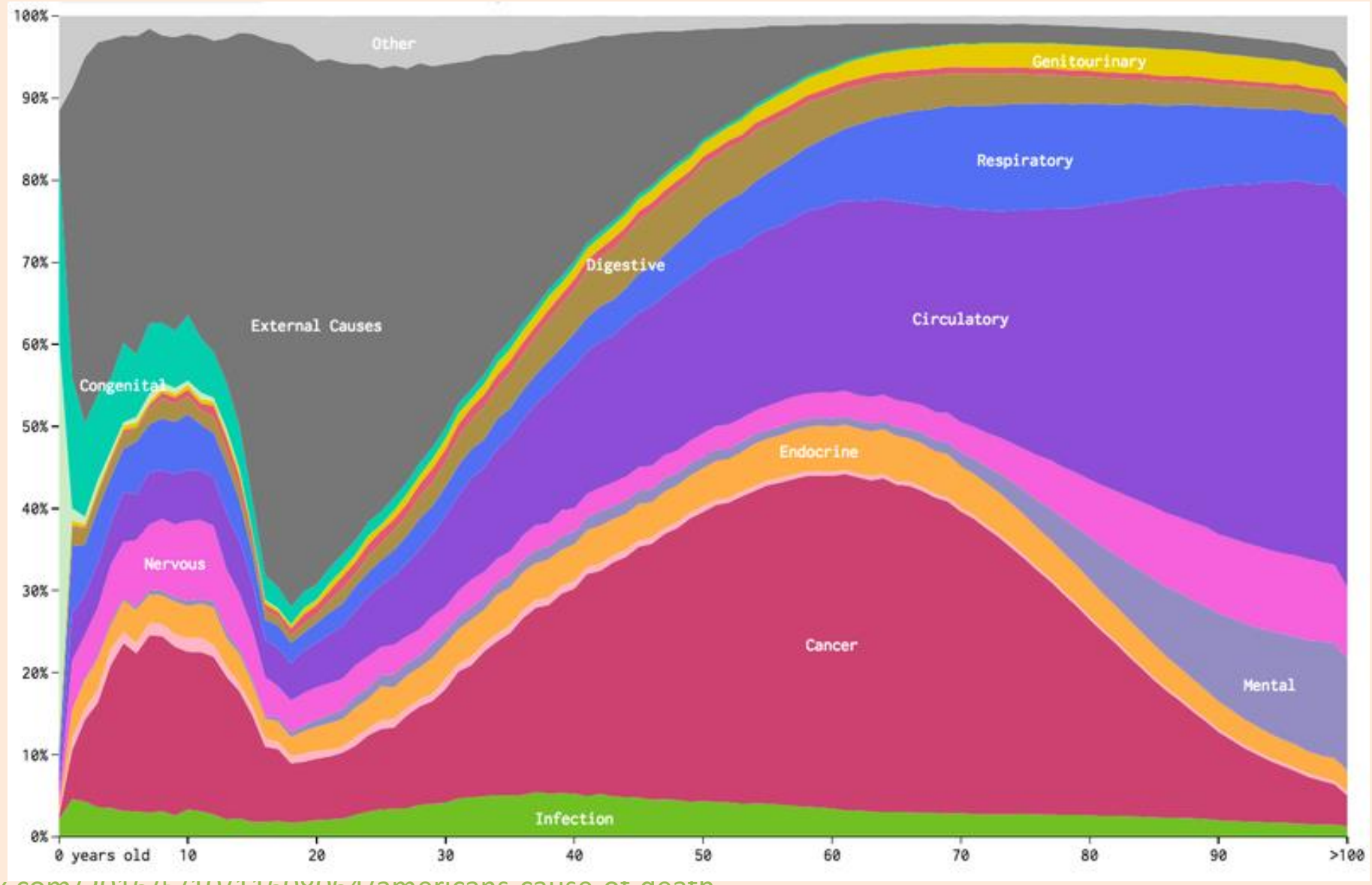
http://waitbutwhy.com/2013/12/how-to-name-baby.html

Stream Graph



http://www.babynamewizard.com/d3js-voyager/popup.html#prefix=ca&sw=both&exact=false

Stream Graph



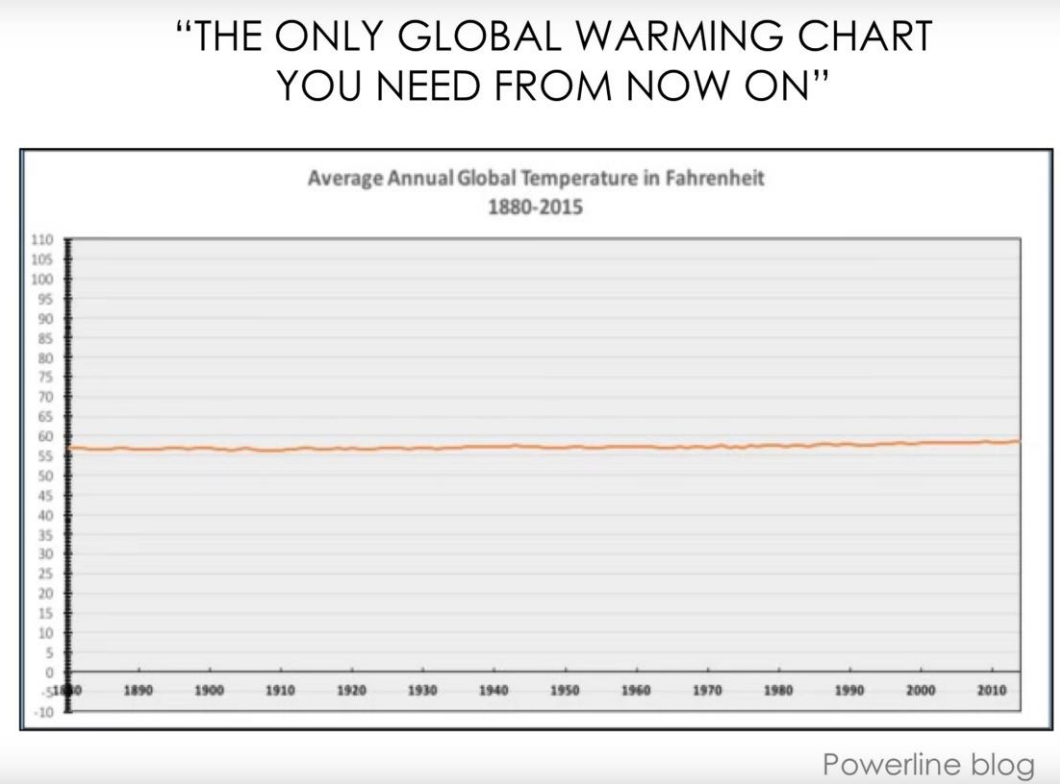
http://www.vox.com/2016/5/10/11608064/americans-cause-of-death

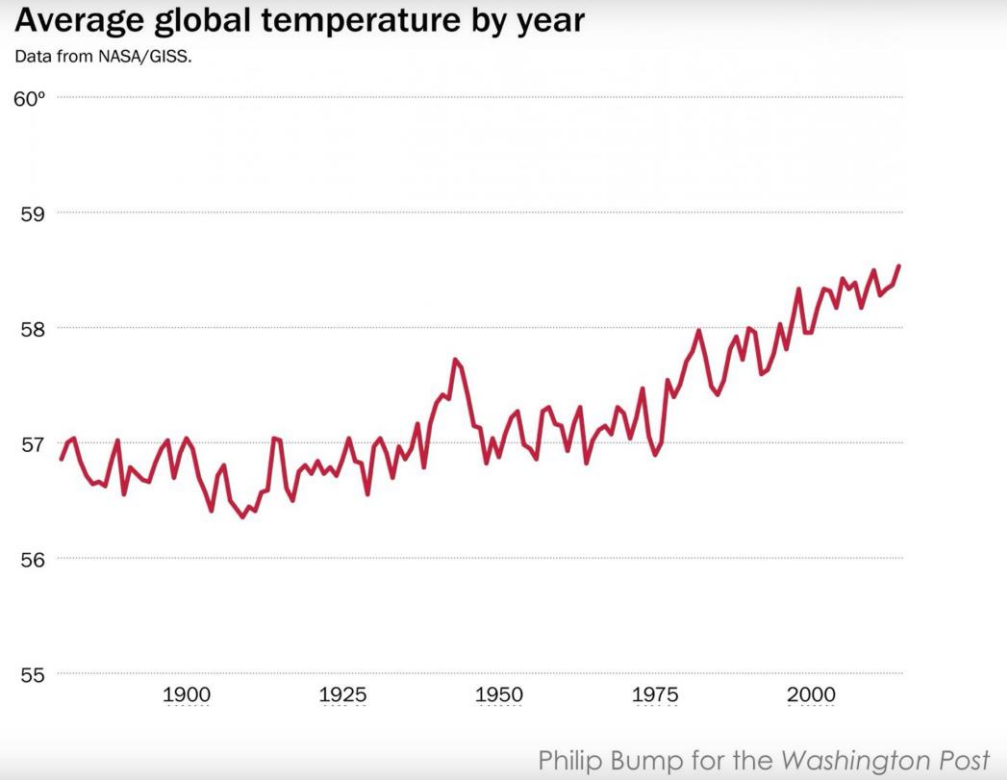
Videos and Interactive Visualizations

- 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.

More Mis-Leading Axes from “Calling Bullshit”





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

More Mis-Leading Axes from “Calling Bullshit”

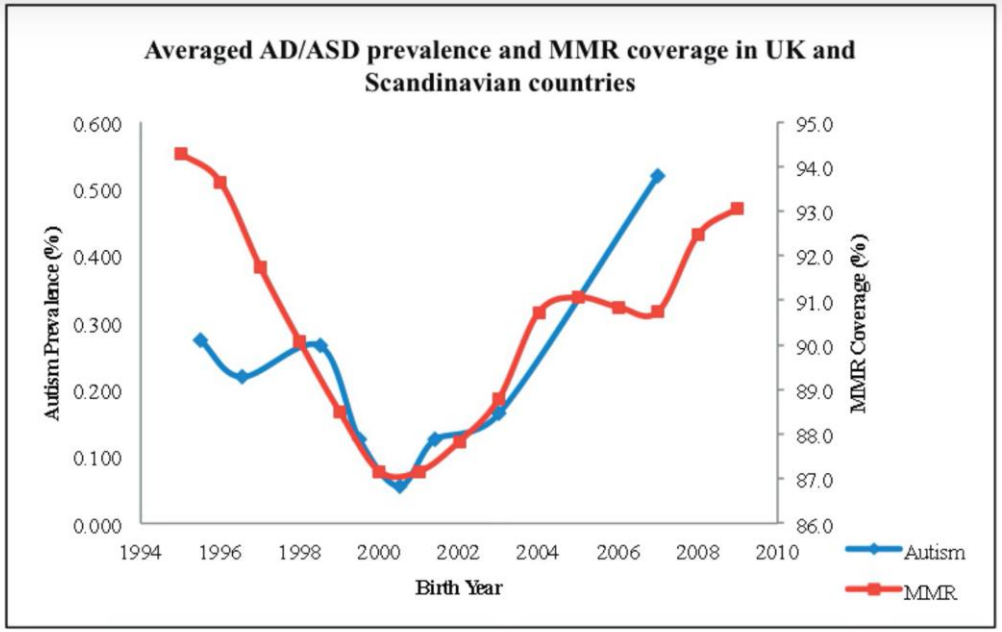
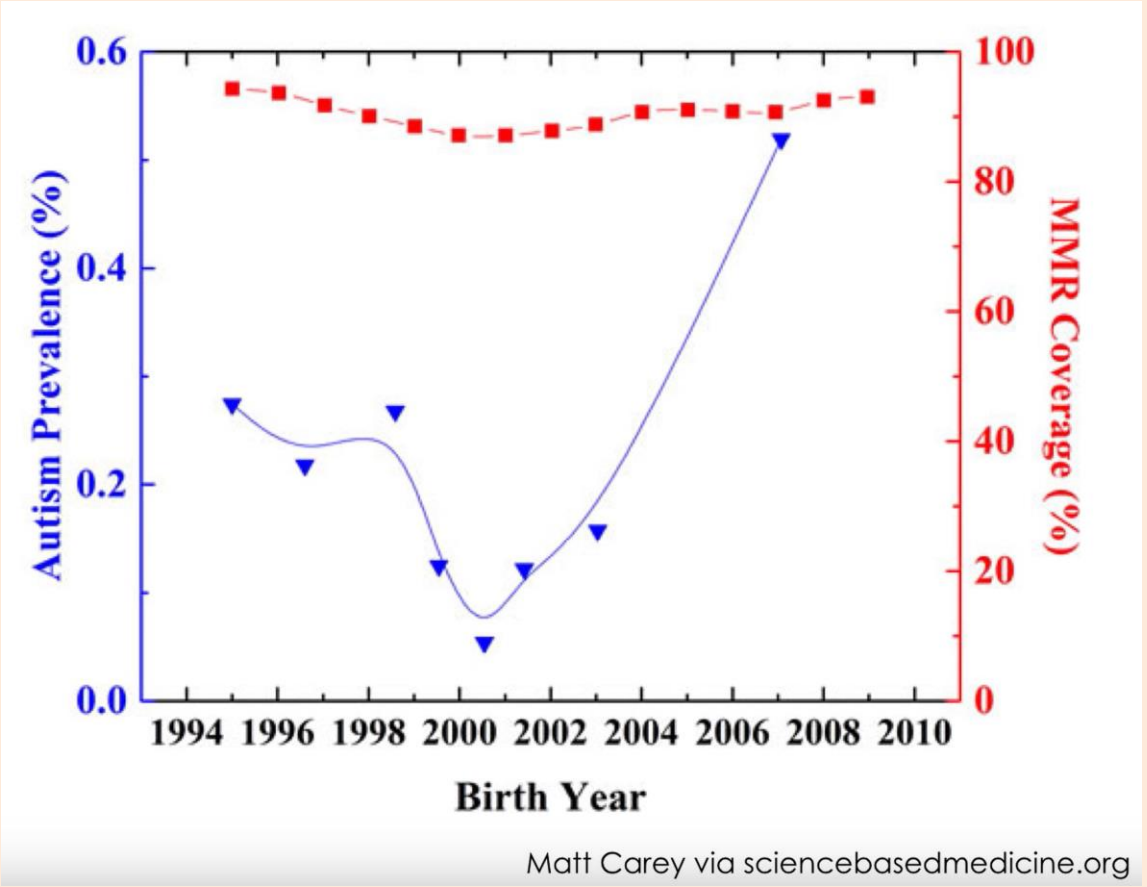


Figure 1-Averaged AD/ASD prevalence and MMR coverage in UK, Norway and Sweden. Both MMR and AD/ASD data are normalized to the maximum coverage/prevalence during the time period of this analysis.

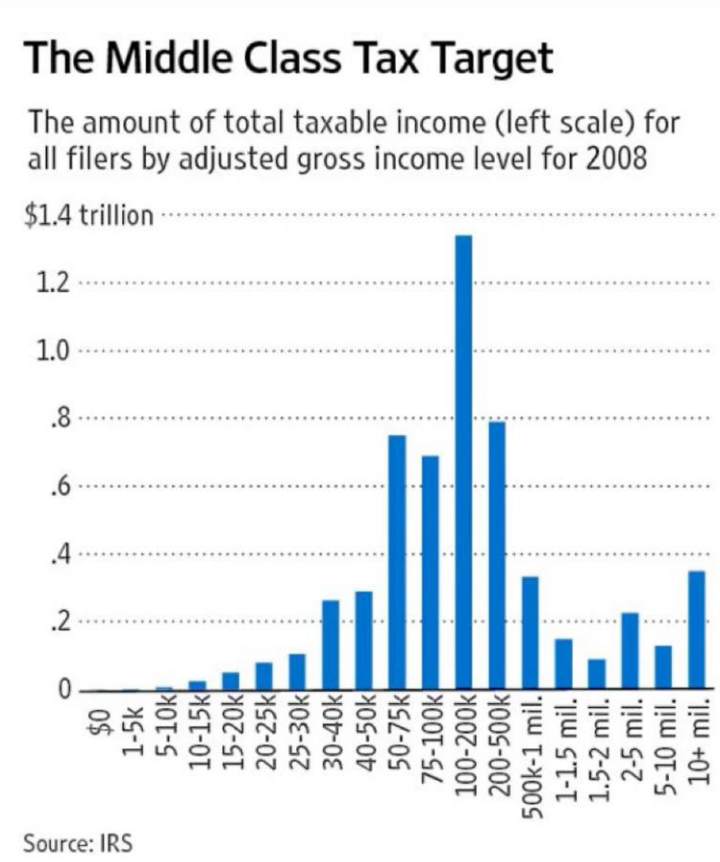
Diesher *et al.* 2015 *Issues in Law and Medicine*



Matt Carey via sciencebasedmedicine.org

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

More Mis-Leading Axes from “Calling Bullshit”



“The 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.”

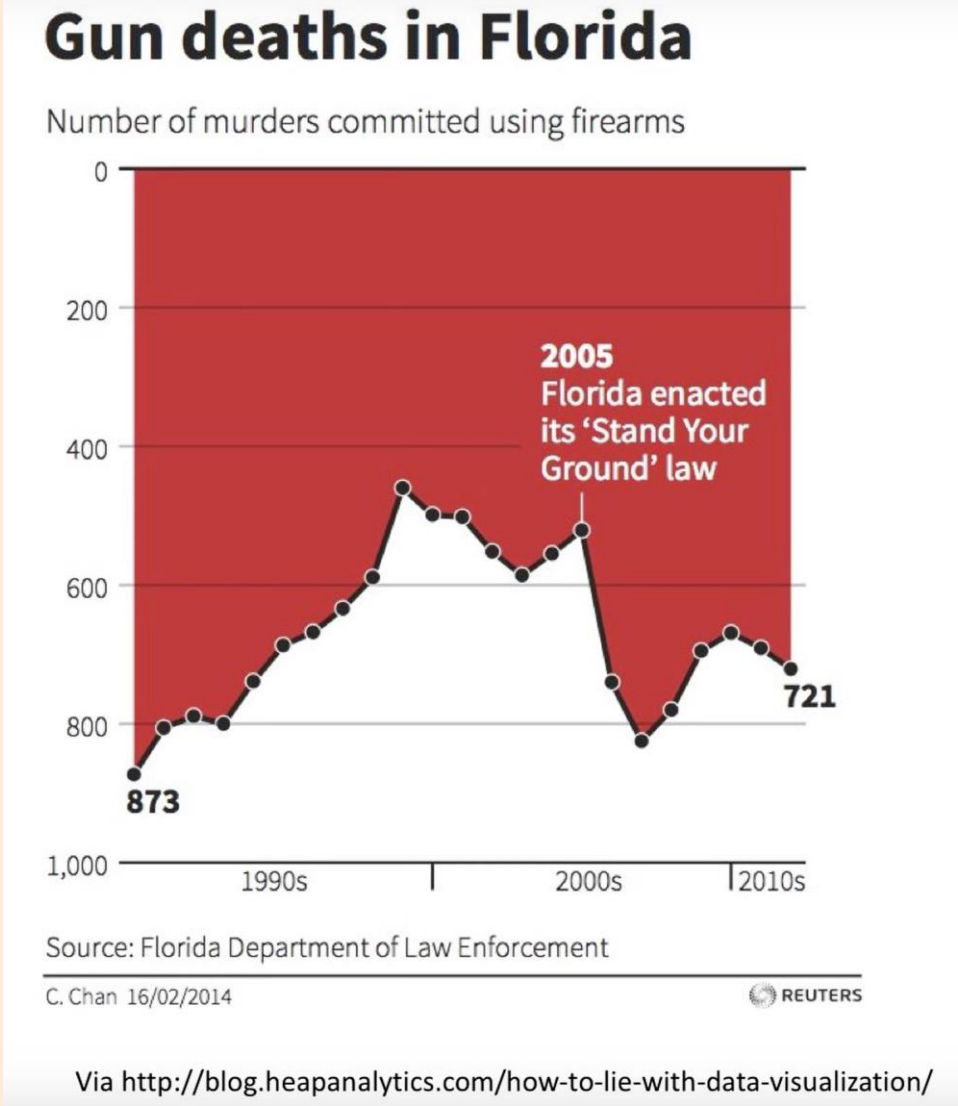
-The Wall Street Journal
April 17, 2011

- Look at the histogram bin widths.

https://www.youtube.com/watch?v=zAg1wsYfwsM

More Mis-Leading Axes from “Calling Bullshit”

- Axis is upside down.
- Looks like law makes murder go down, but number of murders go up!



Via http://blog.heapanalytics.com/how-to-lie-with-data-visualization/

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

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.

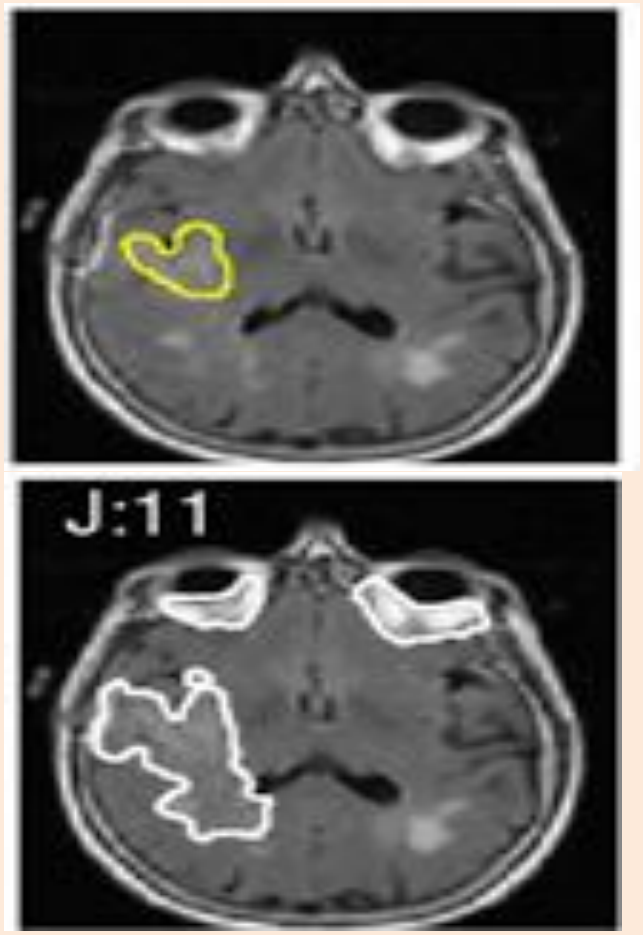
https://www.youtube.com/watch?v=-Mtmi8smpfo

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:

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

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:

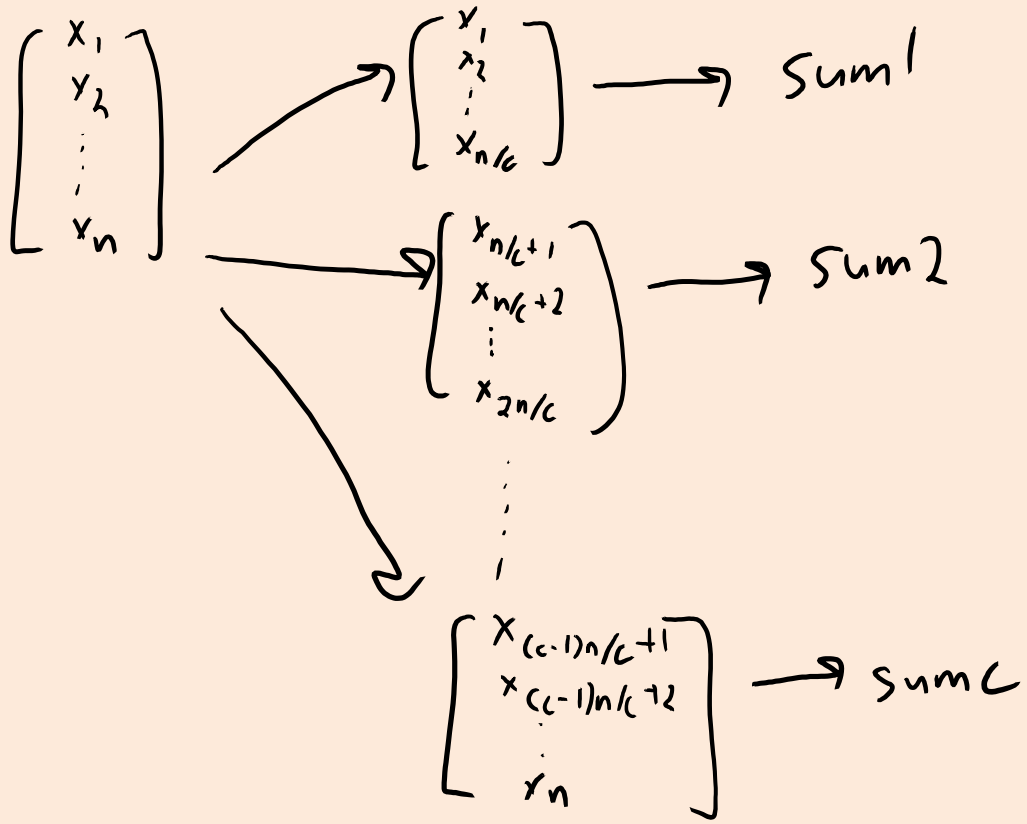
$$\text{mean}(x_1, x_2, x_3, \cdots, x_n) = \frac{x_1 + x_2 + x_3 + \cdots + x_n}{n}$$

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

- But if ‘n’ is really big, we can go even faster with parallel computing…

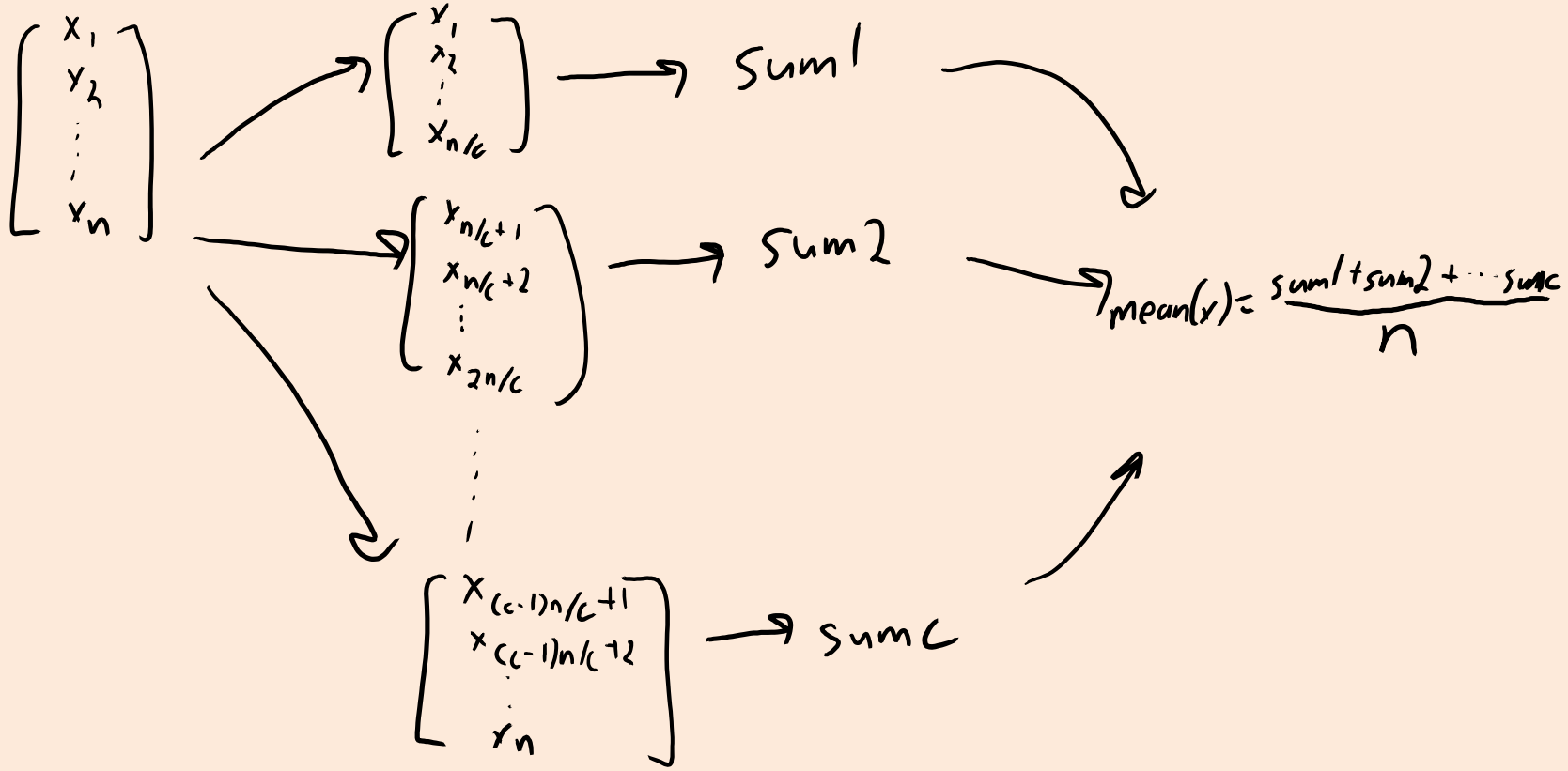
Huge Datasets and Parallel/Distributed Computation

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



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.



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

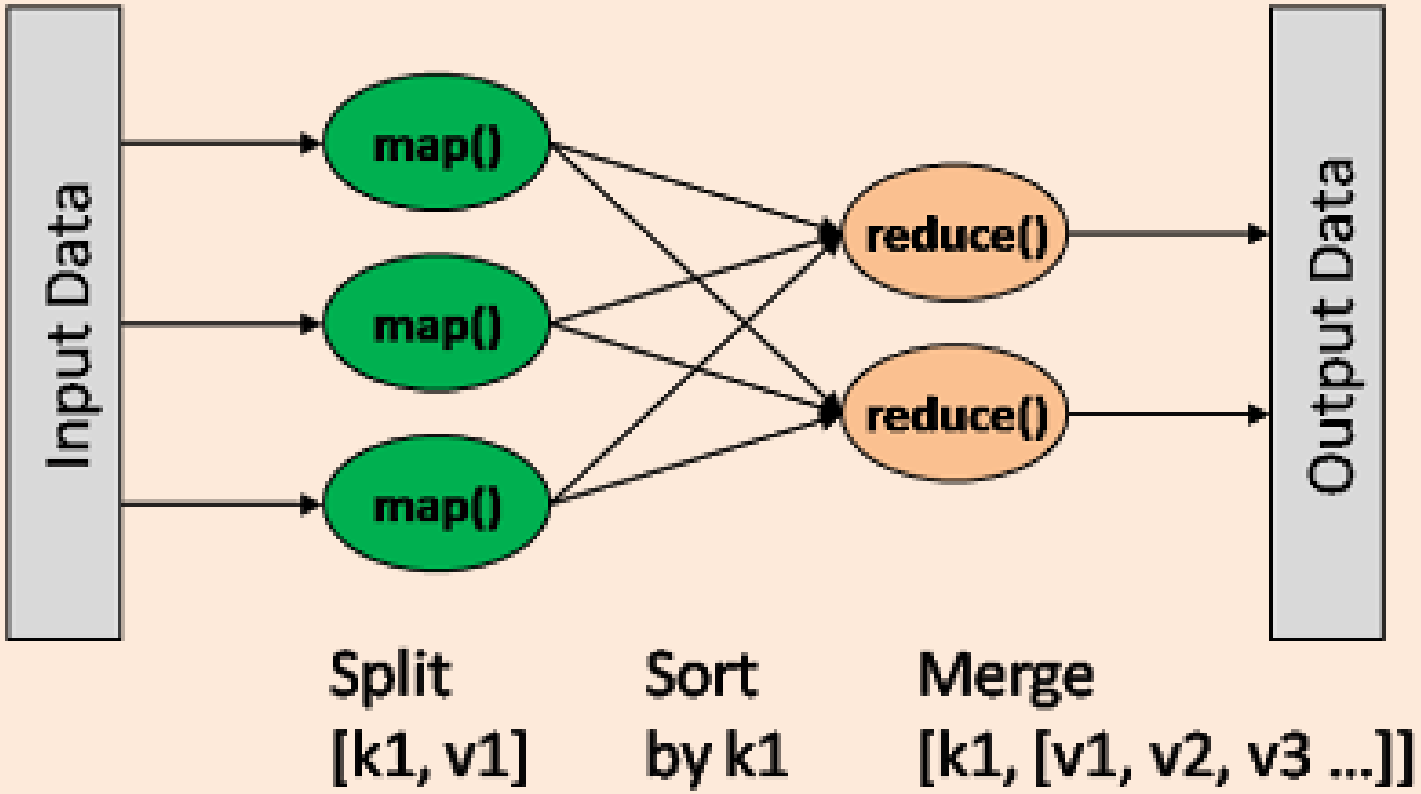
- Sometimes ‘n’ is so big that data can’t fit on one computer.

- 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.



https://dzone.com/articles/how-hadoop-mapreduce-works

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.

CPSC 340:
Machine Learning and Data Mining

Ensemble Methods

Fall 2019

Admin

- Welcome to the course!

- Course webpage:
  - https://www.cs.ubc.ca/~schmidtm/Courses/340-F19/

- Assignment 1:
  - 2 late days to hand in tonight.

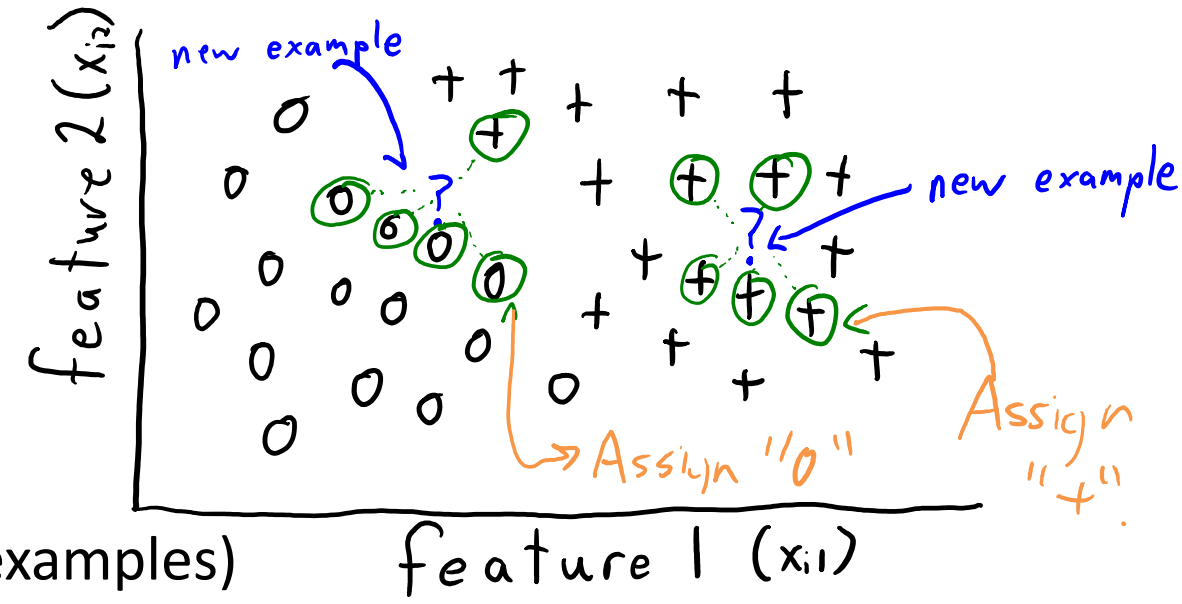
- Assignment 2 is out.
  - Due Friday of next week. It's long so start early.

Last Time: K-Nearest Neighbours (KNN)

- K-nearest neighbours algorithm for classifying $\tilde{x}_i$:
  - Find 'k' values of x_i that are most similar to $\tilde{x}_i$.
  - Use mode of corresponding y_i.

- Lazy learning:
  - To "train" you just store X and y.

- Non-parametric:
  - Size of model grows with 'n' (number of examples)
  - Nearly-optimal test error with infinite data.



- But high prediction cost and may need large 'n' if 'd' is large.

Defining "Distance" with "Norms"

- A common way to define the "distance" between examples:
  - Take the "norm" of the difference between feature vectors.

$$\|x_i - \tilde{x}_i\|_2 = \sqrt{\sum_{j=1}^{d} (x_{ij} - \tilde{x}_{ij})^2}$$

train example — test example — "L_2-norm"

- Norms are a way to measure the "length" of a vector.
  - The most common norm is the "L2-norm" (or "Euclidean norm"):

$$\|r\|_2 = \sqrt{\sum_{j=1}^{d} r_j^2}$$

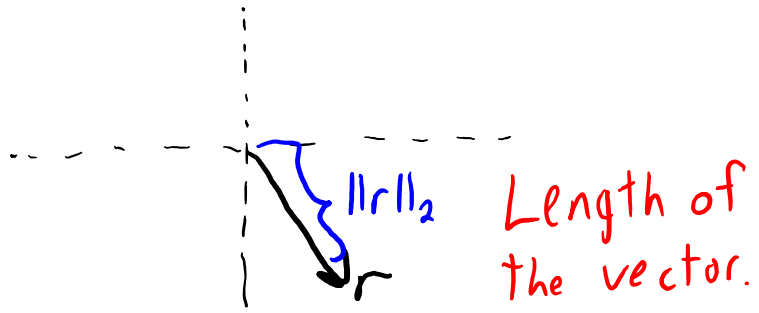
  - Here, the "norm" of the difference is the standard Euclidean distance.

L2-norm, L1-norm, and L∞-Norms.

- The three most common norms: L2-norm, L1-norm, and L∞-norm.
  - Definitions of these norms with two-dimensions:

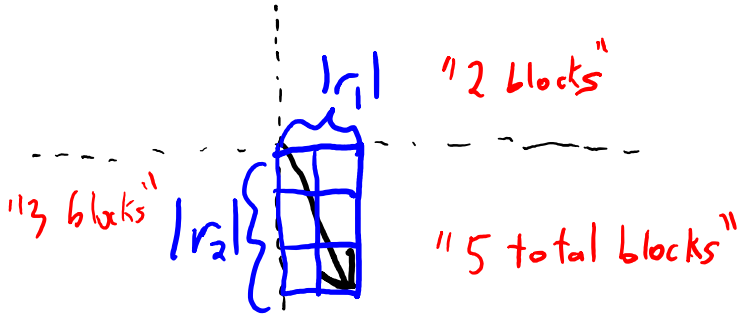
L_2 or "Euclidean" norm.

$$\|r\|_2 = \sqrt{r_1^2 + r_2^2}$$



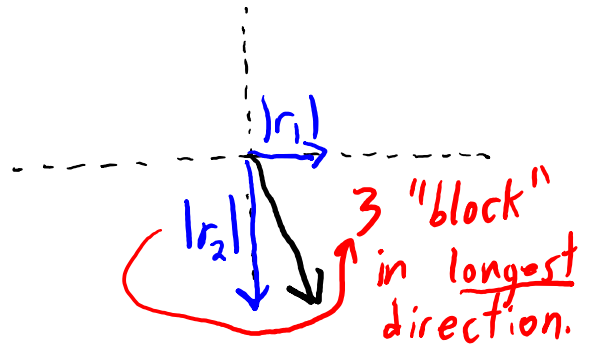
L_1 or "Manhattan" norm:

$$\|r\|_1 = |r_1| + |r_2|$$



L_∞ or "max" norm:

$$\|r\|_\infty = \max\{|r_1|, |r_2|\}$$



  - Definitions of these norms in d-dimensions.

L_2: $\|r\|_2 = \sqrt{\sum_{j=1}^{d} r_j^2}$

L_1: $\|r\|_1 = \sum_{j=1}^{d} |r_j|$

L_∞: $\max_j \{|r_j|\}$

Infinite Series Video

Norm and Norm^p Notation (MEMORIZE)

- Notation:
  - We often leave out the "2" for the L2-norm: We use $\|r\|$ for $\|r\|_2$
  - We use superscripts for raising norms to powers: We use $\|r\|^2$ for $(\|r\|)^2$
  - You should understand why all of the following quantities are equal:

$$\|r\|^2 = \|r\|_2^2 = (\|r\|_2)^2 = \left(\sqrt{\sum_{j=1}^{d} r_j^2}\right)^2 = \sum_{j=1}^{d} r_j^2 = \sum_{j=1}^{d} r_j \cdot r_j = r^T r$$

$$= \langle r, r \rangle$$

(we'll use these later)

Norms as Measures of Distance

- By taking norm of difference, we get a "distance" between vectors:

$$\|r - s\|_2 = \sqrt{(r_1 - s_1)^2 + (r_2 - s_2)^2}$$

$= \|r - s\|$ "Euclidean distance"

$$\|r - s\|_1 = |r_1 - s_1| + |r_2 - s_2|$$

"Number of blocks you need to walk to get from r to s."

$$\|r - s\|_\infty = \max\{|r_1 - s_1|, |r_2 - s_2|\}$$

"Most number of blocks in any direction you would have to walk."

- Place different "weights" on large differences:
  - L_1: differences are equally notable.
  - L_2: bigger differences are more important (because of squaring).
  - L_∞: only biggest difference is important.

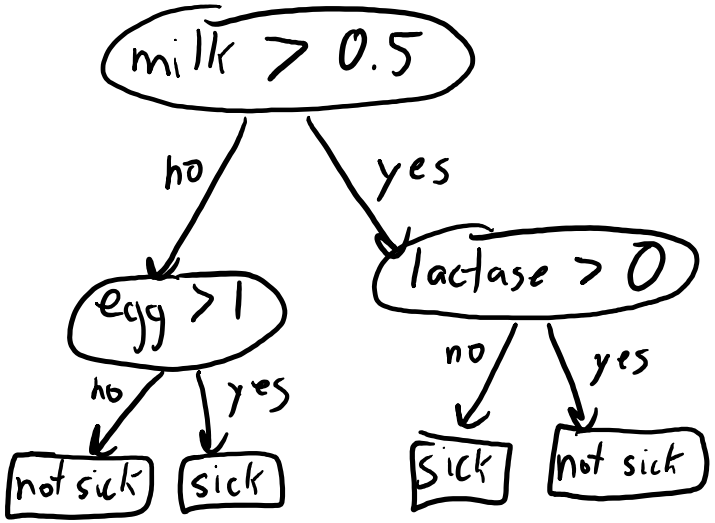
KNN Distance Functions

- Most common KNN distance functions: norm($x_i - x_j$).
  - L1-, L2-, and L∞-norm.
  - Weighted norms (if some features are more important): $\sum_{j=1}^{d} v_j |x_j|$
  - "Mahalanobis" distance (takes into account correlations).
    - See bonus slide for what functions define a "norm".

"weight" of feature j

- But we can consider other distance/similarity functions:
  - Jaccard similarity (if x_i are sets).
  - Edit distance (if x_i are strings).
  - Metric learning (*learn* the best distance function).

Decision Trees vs. Naïve Bayes vs. KNN

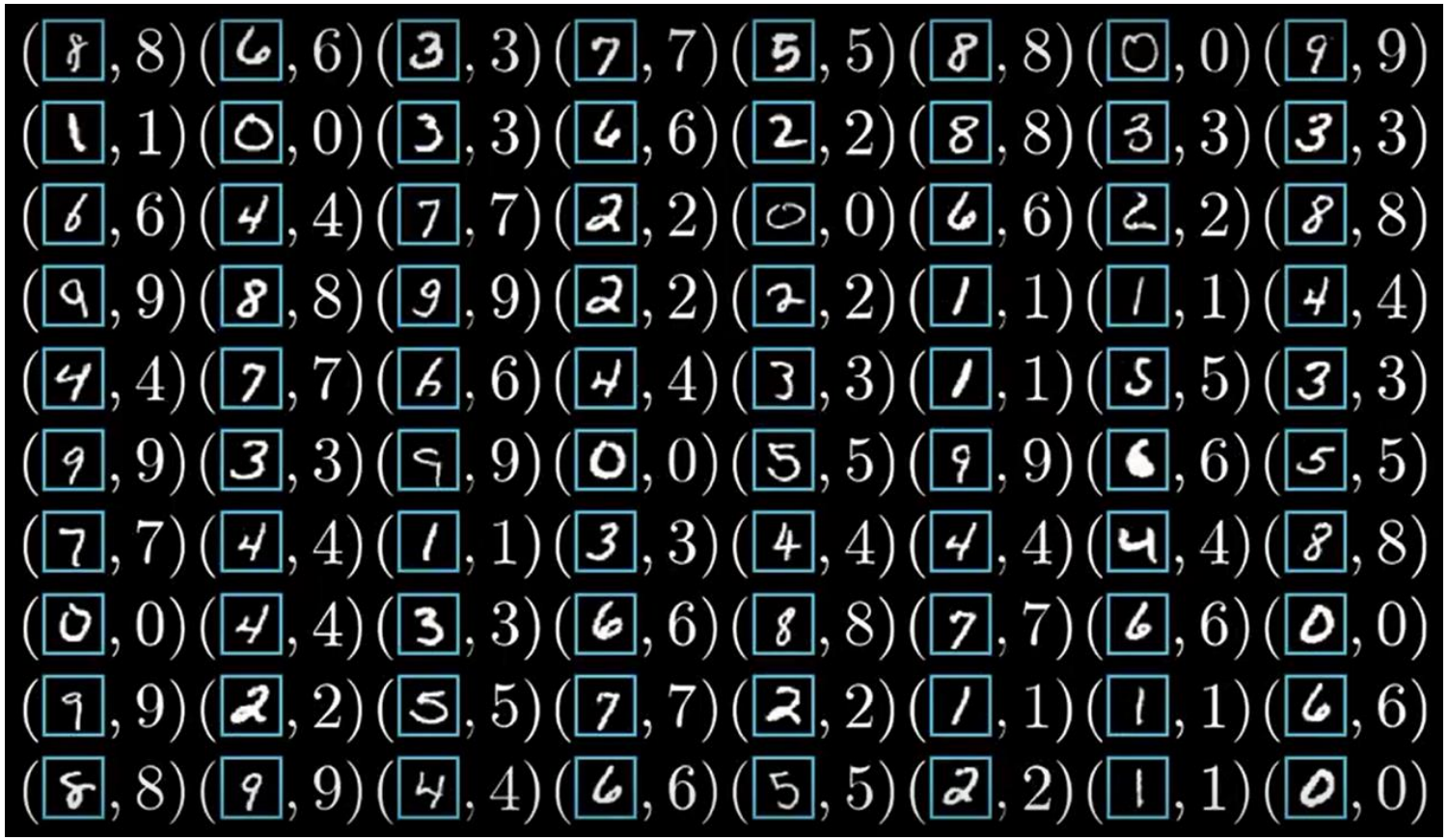


$$p(\text{sick} \mid \text{milk}, \text{egg}, \text{lactase}) \approx p(\text{milk} \mid \text{sick})\, p(\text{egg} \mid \text{sick})\, p(\text{lactase} \mid \text{sick})\, p(\text{sick})$$

(milk = 0.6, egg = 2, lactase = 0, ??) is close to
(milk = 0.7, egg = 2, lactase = 0, sick) so predict sick.

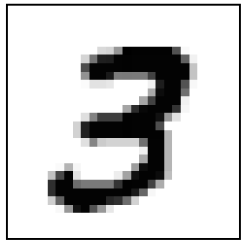
Application: Optical Character Recognition

- To scan documents, we want to turn images into characters:
  - "Optical character recognition" (OCR).



Application: Optical Character Recognition

- To scan documents, we want to turn images into characters:
  - "Optical character recognition" (OCR).



"3"

  - Turning this into a supervised learning problem (with 28 by 28 images):

$X =$

(1,1)	(2,1)	(3,1)	…	(28,1)	(1,2)	(2,2)	…	(14,14)	…	(28,28)
0	0	0		0	0	0		1		0
0	0	0		0	0	0		1		0
0	0	0		0	0	0		0		0
0	0	0		0	0	0		1		0

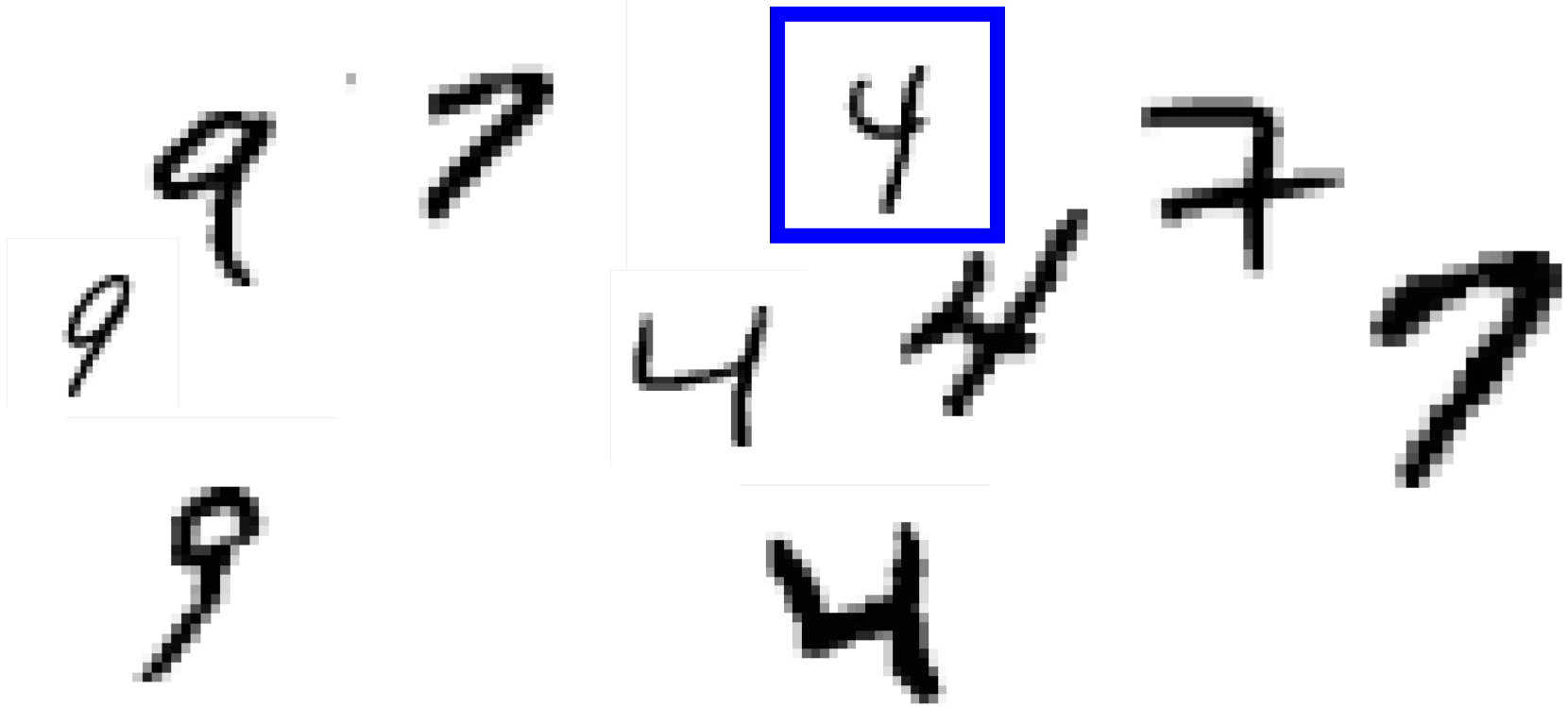
Each feature is grayscale intensity of one of the 784 pixels

$y =$

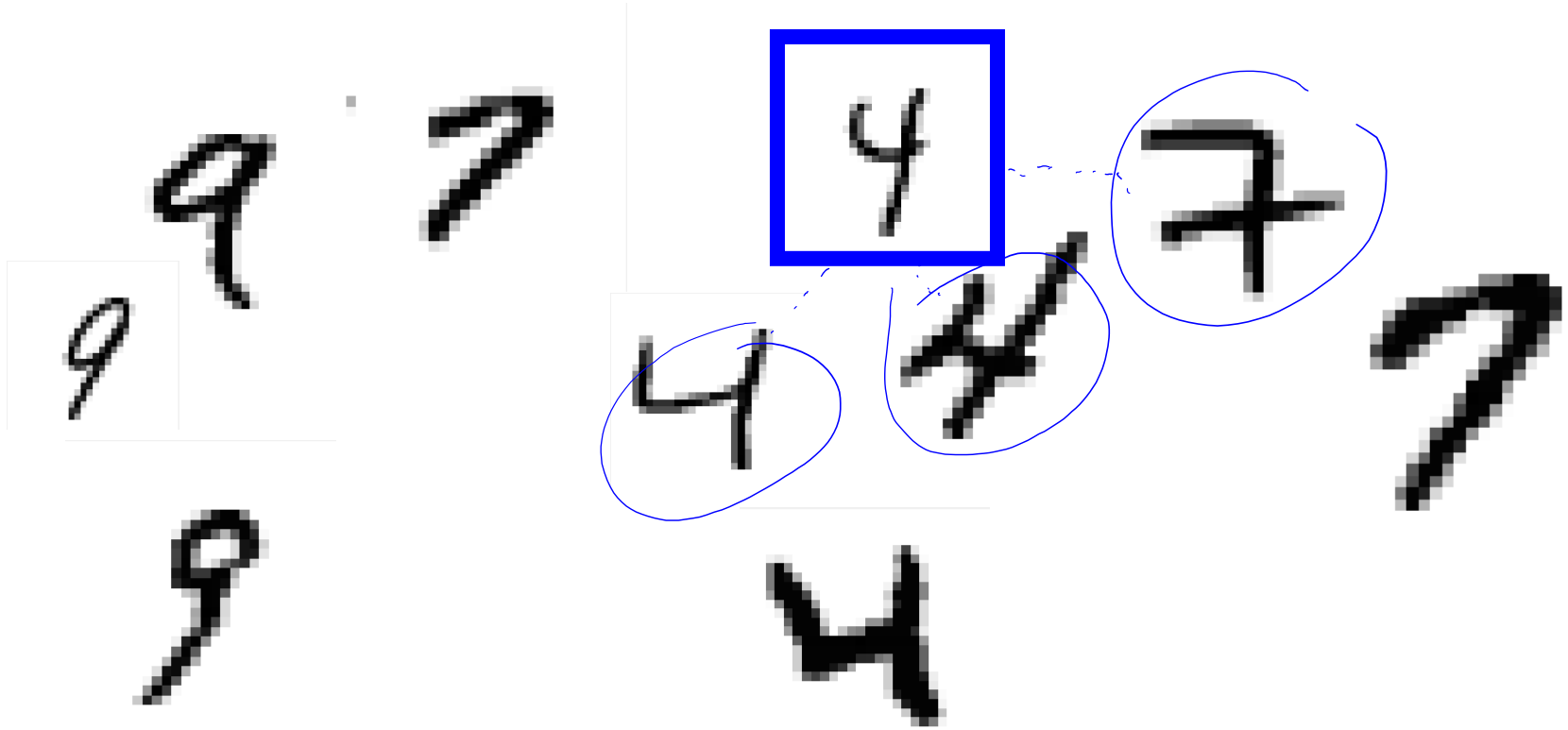
char
3
6
0
9

KNN for Optical Character Recognition

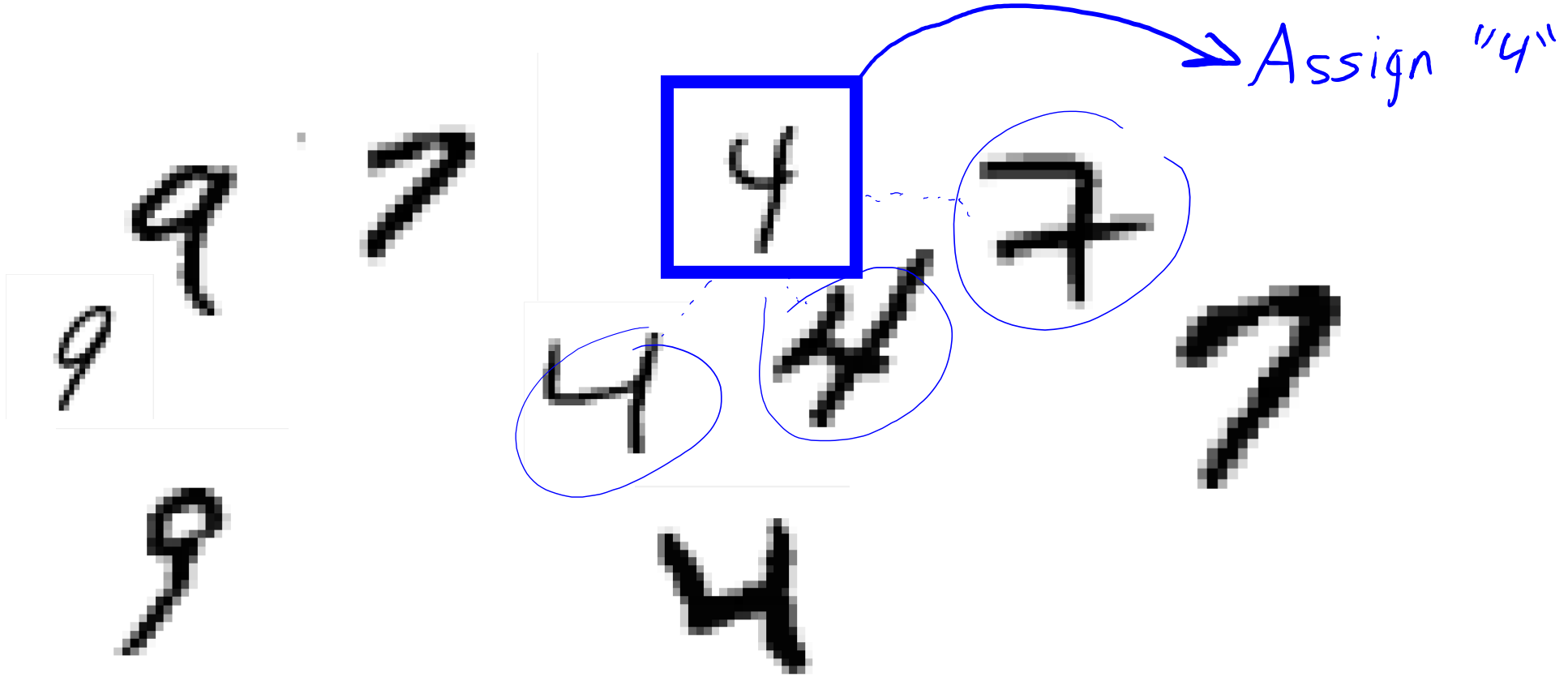
KNN for Optical Character Recognition



KNN for Optical Character Recognition



KNN for Optical Character Recognition



Human vs. Machine Perception

- There is huge difference between what we see and what KNN sees:

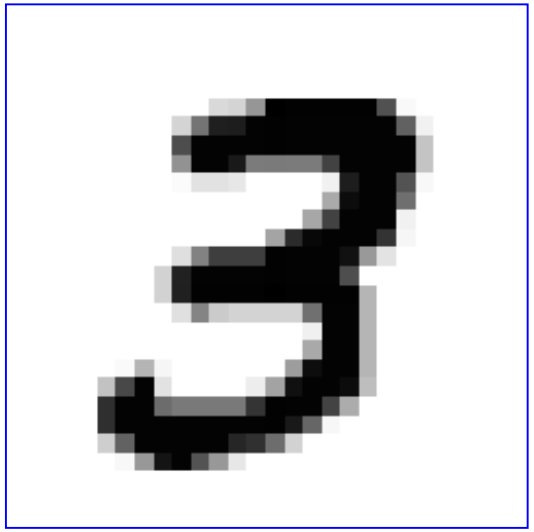
What we see:

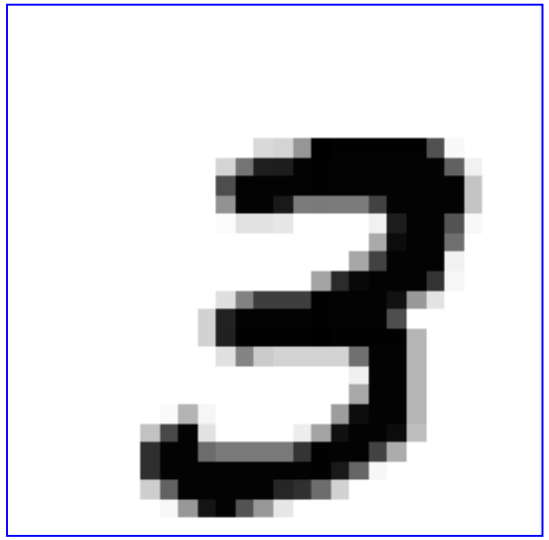
What the computer “sees”:

Actually, it’s worse:

What the Computer Sees

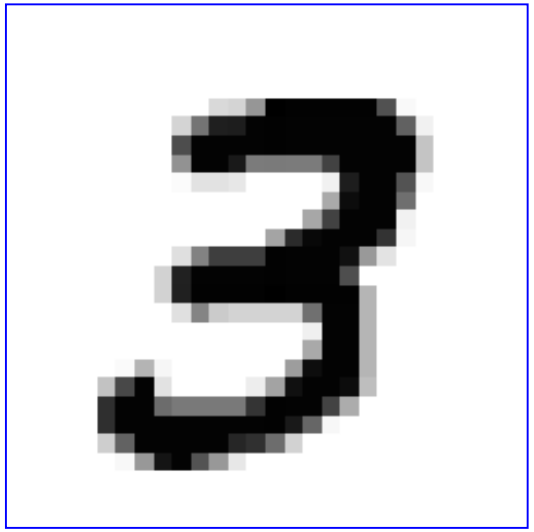
- Are these two images “similar”?

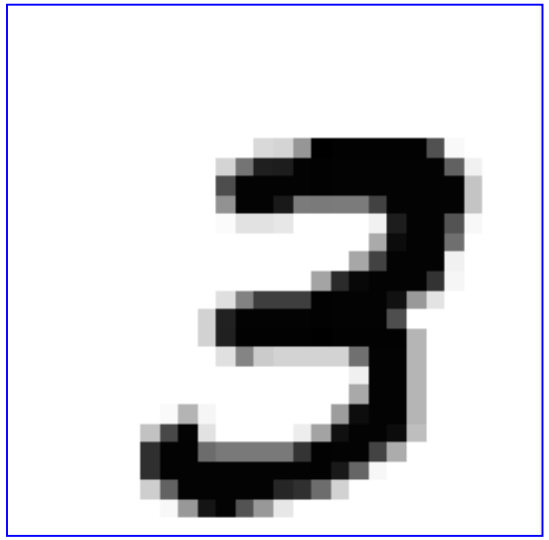




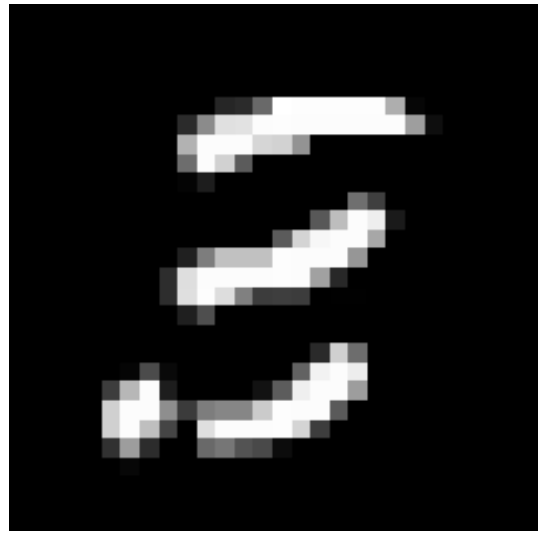
What the Computer Sees

- Are these two images “similar”?





Difference:

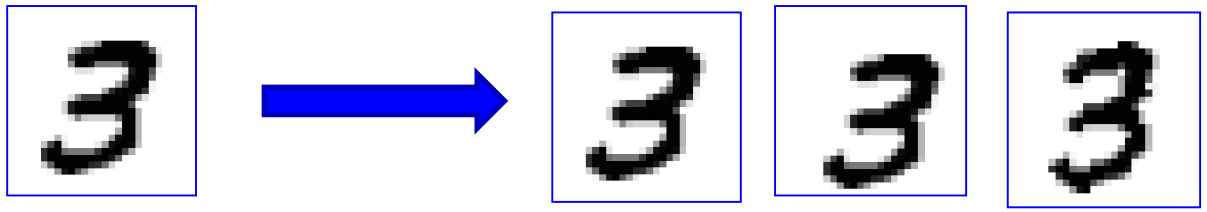


- KNN does not know that labels should be translation invariant.

Encouraging Invariance

- May want classifier to be invariant to certain feature transforms.
  - Images: translations, small rotations, changes in size, mild warping,…
- The hard/slow way is to modify your distance function:
  - Find neighbours that require the “smallest” transformation of image.

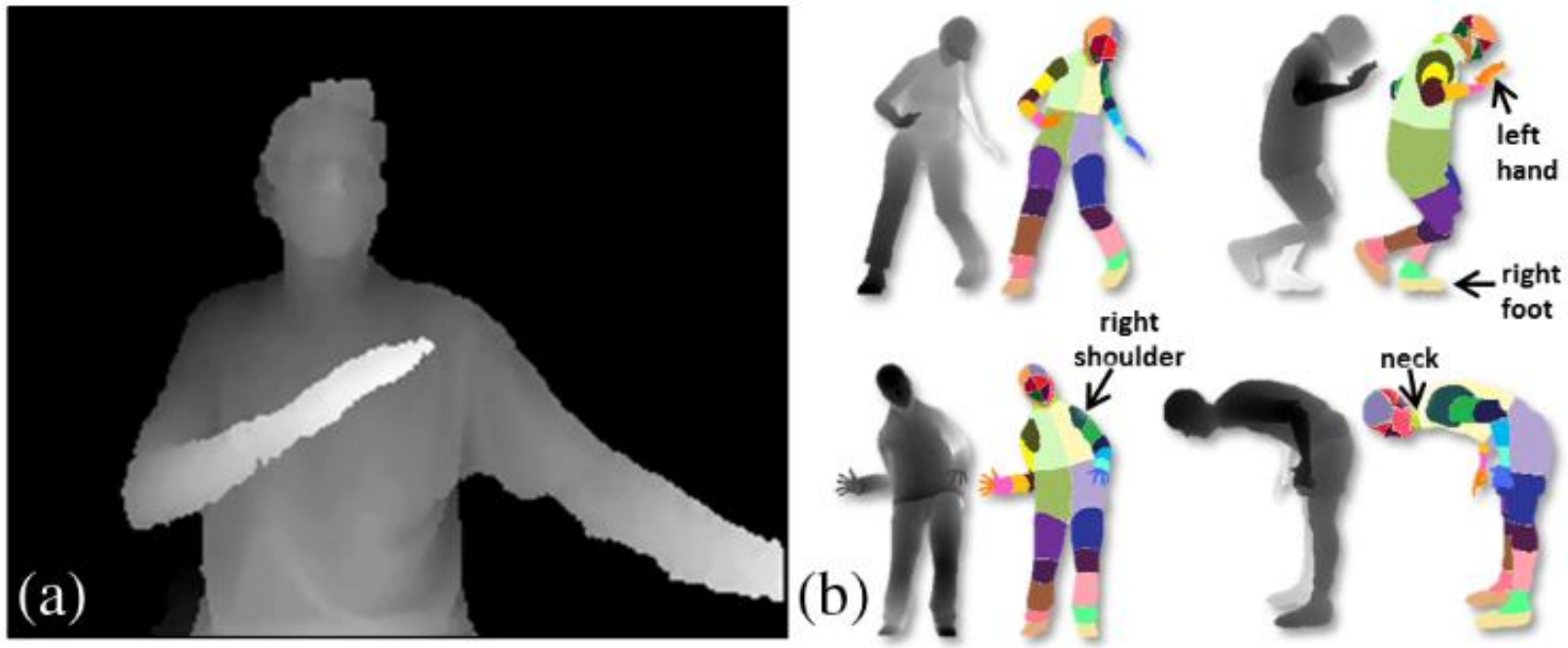
- The easy/fast way is to just add transformed data during training:
  - Add translated/rotate/resized/warped versions of training images.



  - Crucial part of many successful vision systems.
  - Also really important for sound (translate, change volume, and so on).

Application: Body-Part Recognition

- Microsoft Kinect:
  - Real-time recognition of 31 body parts from laser depth data.



- How could we write a program to do this?

http://research.microsoft.com/pubs/158806/CriminisiForests_FoundTrends_2011.pdf