

Introduction and Optimization Problems

John Guttag

MIT Department of Electrical Engineering and
Computer Science

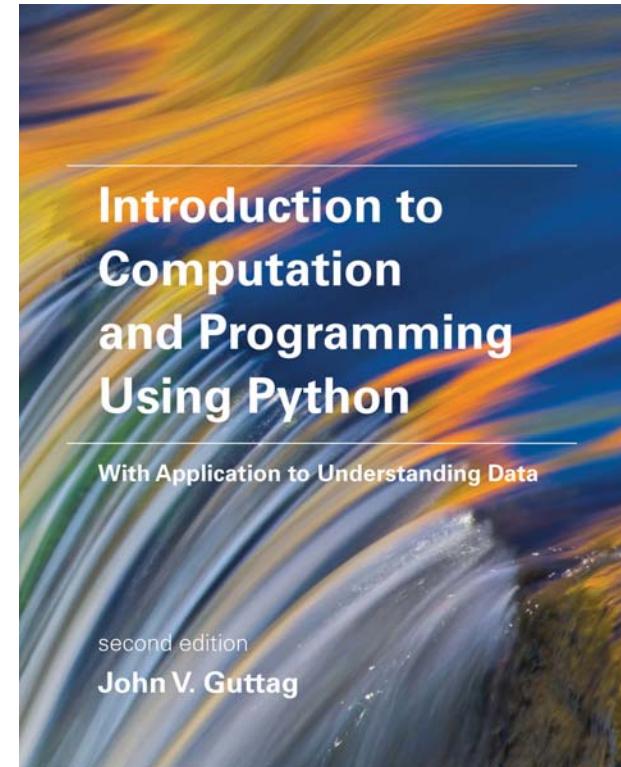
6.0002 Prerequisites

- Experience writing object-oriented programs in Python
 - Preferably in Python 3.5
- Familiarity with concepts of computational complexity
- Familiarity with some simple algorithms
- **6.0001 sufficient**

Question 1

Some Administrative Things

- Problem sets
 - Programming problems designed to
 - Improve your programming skills
 - Help you learn the conceptual material
- Finger exercises
 - Very small programming problems designed to help you learn a single programming concept
- Reading assignments in textbook
 - Another take on and more details about material covered by lectures and problem sets
- Exam: based on above



How Does It Compare to 6.0001?

- Programming assignments a bit easier
 - Focus more on the problem to be solved than on programming
- Lecture content more abstract
- Lectures will be a bit faster paced
- Less about learning to program, more about dipping your toe into data science

Honing Your Programming Skills

- A few additional bits of Python
- Software engineering
- Using packages
- How do you get to Carnegie Hall?

Computational Models

- Using computation to help understand the world in which we live
- Experimental devices that help us to understand something that has happened or to predict the future



Images © sources unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

- Optimization models
- Statistical models
- Simulation models

Relevant Reading for Today's Lecture

- Section 12.1
- Section 5.4 (lambda functions)

Computational Models

- Using computation to help understand the world in which we live
- Experimental devices that help us to understand something that has happened or to predict the future



Images © sources unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

- *Optimization models*
- Statistical models
- Simulation models

What Is an Optimization Model?

- An objective function that is to be maximized or minimized, e.g.,
 - Minimize time spent traveling from New York to Boston
- A set of constraints (possibly empty) that must be honored, e.g.,
 - Cannot spend more than \$100
 - Must be in Boston before 5:00PM



Images © sources unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

Knapsack Problems



Images © sources unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

Knapsack Problem

- You have limited strength, so there is a maximum weight knapsack that you can carry
- You would like to take more stuff than you can carry
- How do you choose which stuff to take and which to leave behind?
- Two variants
 - 0/1 knapsack problem
 - Continuous or fractional knapsack problem



Images © sources unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

My Least-favorite Knapsack Problem



Images © sources unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

0/1 Knapsack Problem, Formalized

- Each item is represented by a pair, $\langle \text{value}, \text{weight} \rangle$
- The knapsack can accommodate items with a total weight of no more than w
- A vector, L , of length n , represents the set of available items. Each element of the vector is an item
- A vector, V , of length n , is used to indicate whether or not items are taken. If $V[i] = 1$, item $I[i]$ is taken. If $V[i] = 0$, item $I[i]$ is not taken

0/1 Knapsack Problem, Formalized

Find a V that maximizes

$$\sum_{i=0}^{n-1} V[i] * I[i].value$$

subject to the constraint that

$$\sum_{i=0}^{n-1} V[i] * I[i].weight \leq w$$

Brute Force Algorithm

- 1. Enumerate all possible combinations of items. That is to say, generate all subsets of the set of items. This is called the **power set**.
- 2. Remove all of the combinations whose total units exceeds the allowed weight.
- 3. From the remaining combinations choose any one whose value is the largest.

Often Not Practical

- How big is power set?
- Recall
 - A vector, V , of length n , is used to indicate whether or not items are taken. If $V[i] = 1$, item $I[i]$ is taken. If $V[i] = 0$, item $I[i]$ is not taken
- How many possible different values can V have?
 - As many different binary numbers as can be represented in n bits
- For example, if there are 100 items to choose from, the power set is of size?
 - 1,267,650,600,228,229,401,496,703,205,376

Question 2

Are We Just Being Stupid?

- Alas, no
- 0/1 knapsack problem is inherently exponential
- But don't despair

Greedy Algorithm a Practical Alternative

- while knapsack not full
 - put “best” available item in knapsack
- But what does best mean?
 - Most valuable
 - Least expensive
 - Highest value/units

An Example

- You are about to sit down to a meal
- You know how much you value different foods, e.g., you like donuts more than apples
- But you have a calorie budget, e.g., you don't want to consume more than 750 calories
- Choosing what to eat is a knapsack problem

A Menu

Food	wine	beer	pizza	burger	fries	coke	apple	donut
Value	89	90	30	50	90	79	90	10
calories	123	154	258	354	365	150	95	195

- Let's look at a program that we can use to decide what to order

Class Food

```
class Food(object):
    def __init__(self, n, v, w):
        self.name = n
        self.value = v
        self.calories = w

    def getValue(self):
        return self.value

    def getCost(self):
        return self.calories

    def density(self):
        return self.getValue()/self.getCost()

    def __str__(self):
        return self.name + ': <' + str(self.value) \
               + ', ' + str(self.calories) + '>'
```

Build Menu of Foods

```
def buildMenu(names, values, calories):
    """names, values, calories lists of same length.
       name a list of strings
       values and calories lists of numbers
       returns list of Foods"""
    menu = []
    for i in range(len(values)):
        menu.append(Food(names[i], values[i],
                          calories[i]))
    return menu
```

Implementation of Flexible Greedy

```
def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,
       keyFunction maps elements of items to numbers"""
    itemsCopy = sorted(items, key = keyFunction, ←
                      reverse = True)
    result = []
    totalValue, totalCost = 0.0, 0.0

    for i in range(len(itemsCopy)): ←
        if (totalCost+itemsCopy[i].getCost()) <= maxCost:
            result.append(itemsCopy[i])
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()

    return (result, totalValue)
```

Algorithmic Efficiency

```
def greedy(items, maxCost, keyFunction):
    → itemsCopy = sorted(items, key = keyFunction,
                         reverse = True)
    result = []
    totalValue, totalCost = 0.0, 0.0

    for i in range(len(itemsCopy)): ←
        if (totalCost+itemsCopy[i].getCost()) <= maxCost:
            result.append(itemsCopy[i])
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()

    return (result, totalValue)
```

Question 3

Using greedy

```
def testGreedy(items, constraint, keyFunction):
    taken, val = greedy(items, constraint, keyFunction)
    print('Total value of items taken =', val)
    for item in taken:
        print('    ', item)
```

Using greedy

```
def testGreedy(maxUnits):
    print('Use greedy by value to allocate', maxUnits,
          'calories')
    testGreedy(foods, maxUnits, Food.getValue)
    print('\nUse greedy by cost to allocate', maxUnits,
          'calories')
    testGreedy(foods, maxUnits,
               lambda x: 1/Food.getCost(x)) ←
    print('\nUse greedy by density to allocate', maxUnits,
          'calories')
    testGreedy(foods, maxUnits, Food.density)

testGreedy(800) ?
```

lambda

- lambda used to create anonymous functions
 - `Lambda <id1, id2, ... idn>: <expression>`
 - Returns a function of n arguments
- Can be very handy, as here
- Possible to write amazing complicated lambda expressions
- **Don't**—use `def` instead

Using greedy

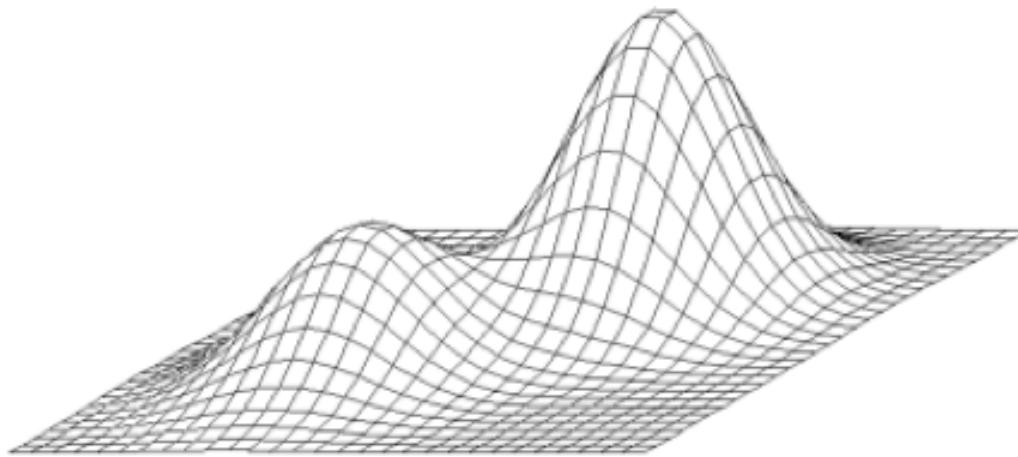
```
def testGreedy(foods, maxUnits):
    print('Use greedy by value to allocate', maxUnits,
          'calories')
    testGreedy(foods, maxUnits, Food.getValue)
    print('\nUse greedy by cost to allocate', maxUnits,
          'calories')
    testGreedy(foods, maxUnits,
               lambda x: 1/Food.getCost(x))
    print('\nUse greedy by density to allocate', maxUnits,
          'calories')
    testGreedy(foods, maxUnits, Food.density)

names = ['wine', 'beer', 'pizza', 'burger', 'fries',
         'cola', 'apple', 'donut', 'cake']
values = [89, 90, 95, 100, 90, 79, 50, 10]
calories = [123, 154, 258, 354, 365, 150, 95, 195]
foods = buildMenu(names, values, calories)
testGreedy(foods, 750)
```

[Run code](#)

Why Different Answers?

- Sequence of locally “optimal” choices don’t always yield a globally optimal solution



- Is greedy by density always a winner?
 - Try `testGreedy(foods, 1000)`

The Pros and Cons of Greedy

- Easy to implement
- Computationally efficient

- But does not always yield the best solution
 - Don't even know how good the approximation is
- In the next lecture we'll look at finding truly optimal solutions

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Optimization Problems,

John Guttag

MIT Department of Electrical Engineering and
Computer Science

Relevant Reading for Today's Lecture

- Chapter 13

The Pros and Cons of Greedy

- Easy to implement
 - Computationally efficient
-
- But does not always yield the best solution
 - Don't even know how good the approximation is

Question 1

Brute Force Algorithm

- 1. Enumerate all possible combinations of items.
- 2. Remove all of the combinations whose total units exceeds the allowed weight.
- 3. From the remaining combinations choose any one whose value is the largest.

Search Tree Implementation

- The tree is built top down starting with the root
- The first element is selected from the still to be considered items
 - If there is room for that item in the knapsack, a node is constructed that reflects the consequence of choosing to take that item. By convention, we draw that as the left child
 - We also explore the consequences of not taking that item. This is the right child
- The process is then applied **recursively** to non-leaf children
- Finally, chose a node with the highest value that meets constraints

A Search Tree Enumerates Possibilities

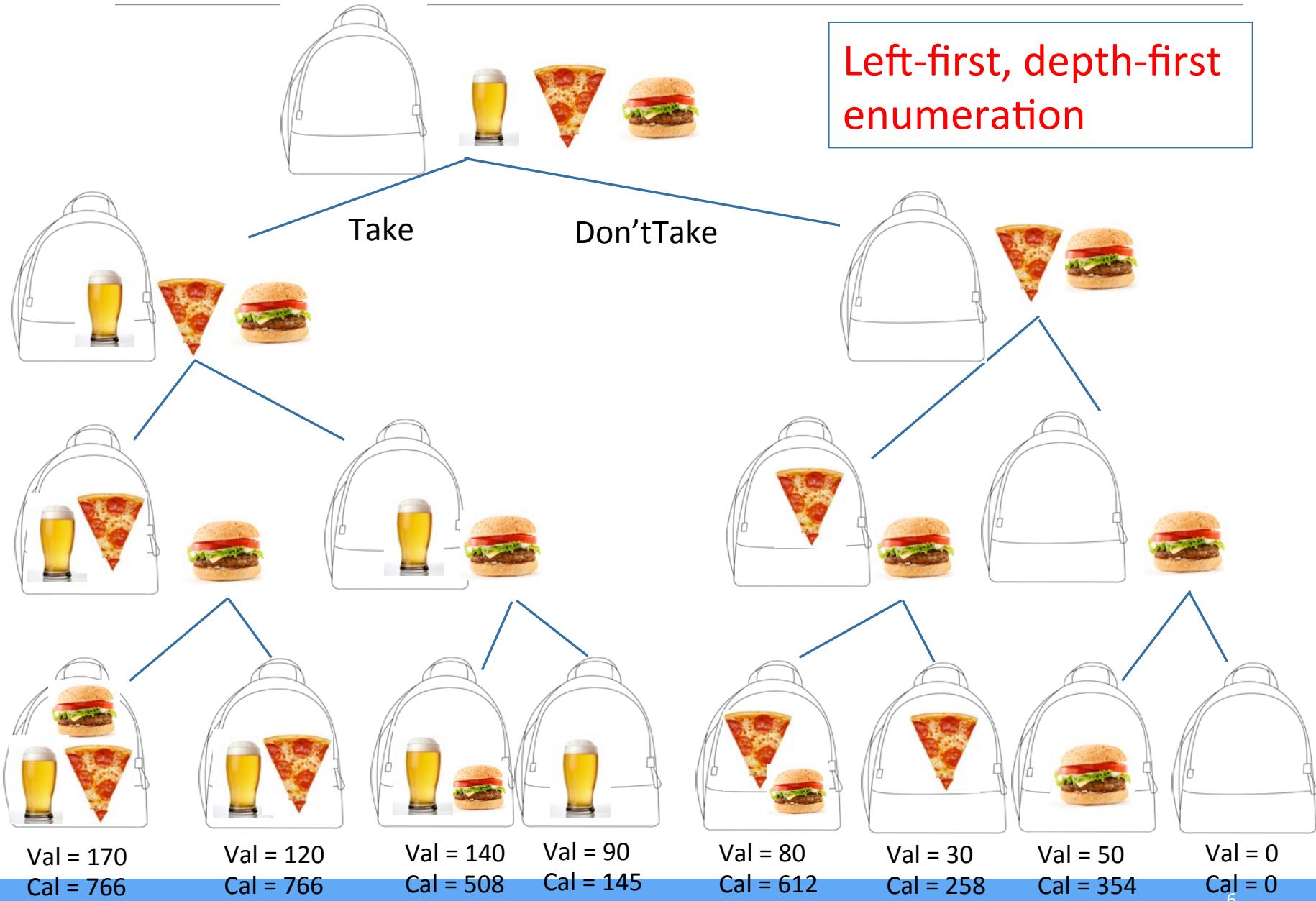




Image © source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

Computational Complexity

- Time based on number of nodes generated
- Number of levels is number of items to choose from
- Number of nodes at level i is 2^i
- So, if there are n items the number of nodes is
 - $\sum_{i=0}^{n-1} 2^i$
 - I.e., $O(2^n)$
- An obvious optimization: don't explore parts of tree that violate constraint (e.g., too many calories)
 - Doesn't change complexity
- Does this mean that brute force is never useful?
 - Let's give it a try

Header for Decision Tree Implementation

```
def maxVal(toConsider, avail):
    """Assumes toConsider a list of items,
       avail a weight
    Returns a tuple of the total value of a
    solution to 0/1 knapsack problem and
    the items of that solution"""

```

toConsider. Those items that nodes higher up in the tree (corresponding to earlier calls in the recursive call stack) have not yet considered

avail. The amount of space still available

Body of maxVal (without comments)

```
if toConsider == [] or avail == 0:
    result = (0, ())
elif toConsider[0].getUnits() > avail:
    result = maxVal(toConsider[1:], avail)
else:
    nextItem = toConsider[0]
    withVal, withToTake = maxVal(toConsider[1:],
                                    avail - nextItem.getUnits())
    withVal += nextItem.getValue()
    withoutVal, withoutToTake = maxVal(toConsider[1:], avail)
if withVal > withoutVal:
    result = (withVal, withToTake + (nextItem,))
else:
    result = (withoutVal, withoutToTake)
return result
```

Does not actually build search tree

Local variable **result** records best solution found so far

Try on Example from Lecture 1

- With calorie budget of 750 calories, chose an optimal set of foods from the menu

Food	wine	beer	pizza	burger	fries	coke	apple	donut
Value	89	90	30	50	90	79	90	10
calories	123	154	258	354	365	150	95	195

Search Tree Worked Great

- Gave us a better answer
- Finished quickly
- But 2^8 is not a large number
 - We should look at what happens when we have a more extensive menu to choose from

Code to Try Larger Examples

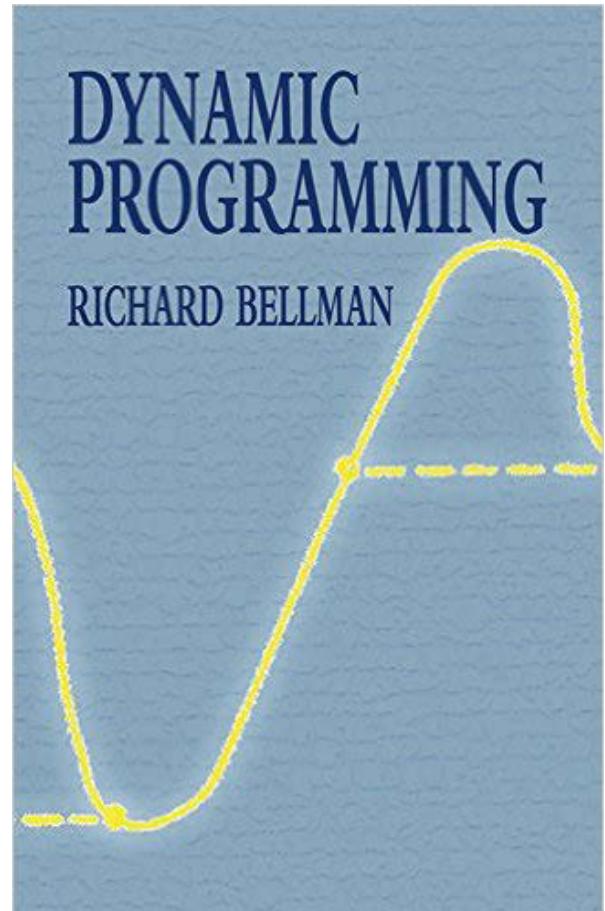
```
import random ←
```

```
def buildLargeMenu(numItems, maxVal, maxCost):
    items = []
    for i in range(numItems):
        items.append(Food(str(i),
                          random.randint(1, maxVal),
                          random.randint(1, maxCost)))
    return items

for numItems in (5,10,15,20,25,30,35,40,45,50,55,60):
    items = buildLargeMenu(numItems, 90, 250)
    testMaxVal(items, 750, False)
```

Is It Hopeless?

- In theory, yes
- In practice, no!
- Dynamic programming to the rescue



Dynamic Programming?

Sometimes a name is just a name

“The 1950s were not good years for mathematical research... I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics... What title, what name, could I choose? ... It's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

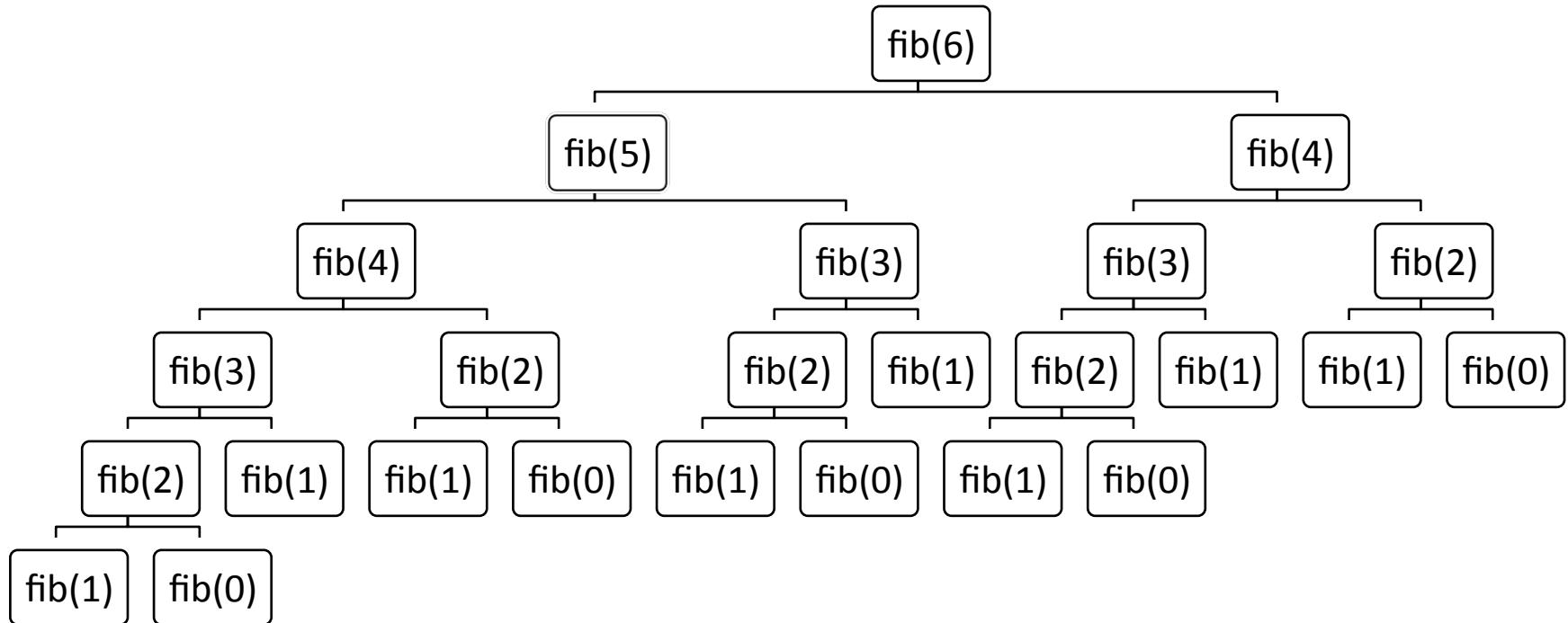
-- Richard Bellman

Recursive Implementation of Fibonnaci

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

$\text{fib}(120) = 8,670,007,398,507,948,658,051,921$

Call Tree for Recursive Fibonnaci(6) = 13



Clearly a Bad Idea to Repeat Work

- Trade a time for space
- Create a table to record what we've done
 - Before computing $\text{fib}(x)$, check if value of $\text{fib}(x)$ already stored in the table
 - If so, look it up
 - If not, compute it and then add it to table
 - Called **memoization**

Using a Memo to Compute Fibonacci

```
def fastFib(n, memo = {}):
    """Assumes n is an int >= 0, memo used only by
       recursive calls
       Returns Fibonacci of n"""
    if n == 0 or n == 1:
        return 1
    try:
        return memo[n]
    except KeyError:
        result = fastFib(n-1, memo) +\
                 fastFib(n-2, memo)
        memo[n] = result
    return result
```

When Does It Work?

- **Optimal substructure**: a globally optimal solution can be found by combining optimal solutions to local subproblems
 - For $x > 1$, $\text{fib}(x) = \text{fib}(x - 1) + \text{fib}(x - 2)$
- **Overlapping subproblems**: finding an optimal solution involves solving the same problem multiple times
 - Compute $\text{fib}(x)$ or many times

What About 0/1 Knapsack Problem?

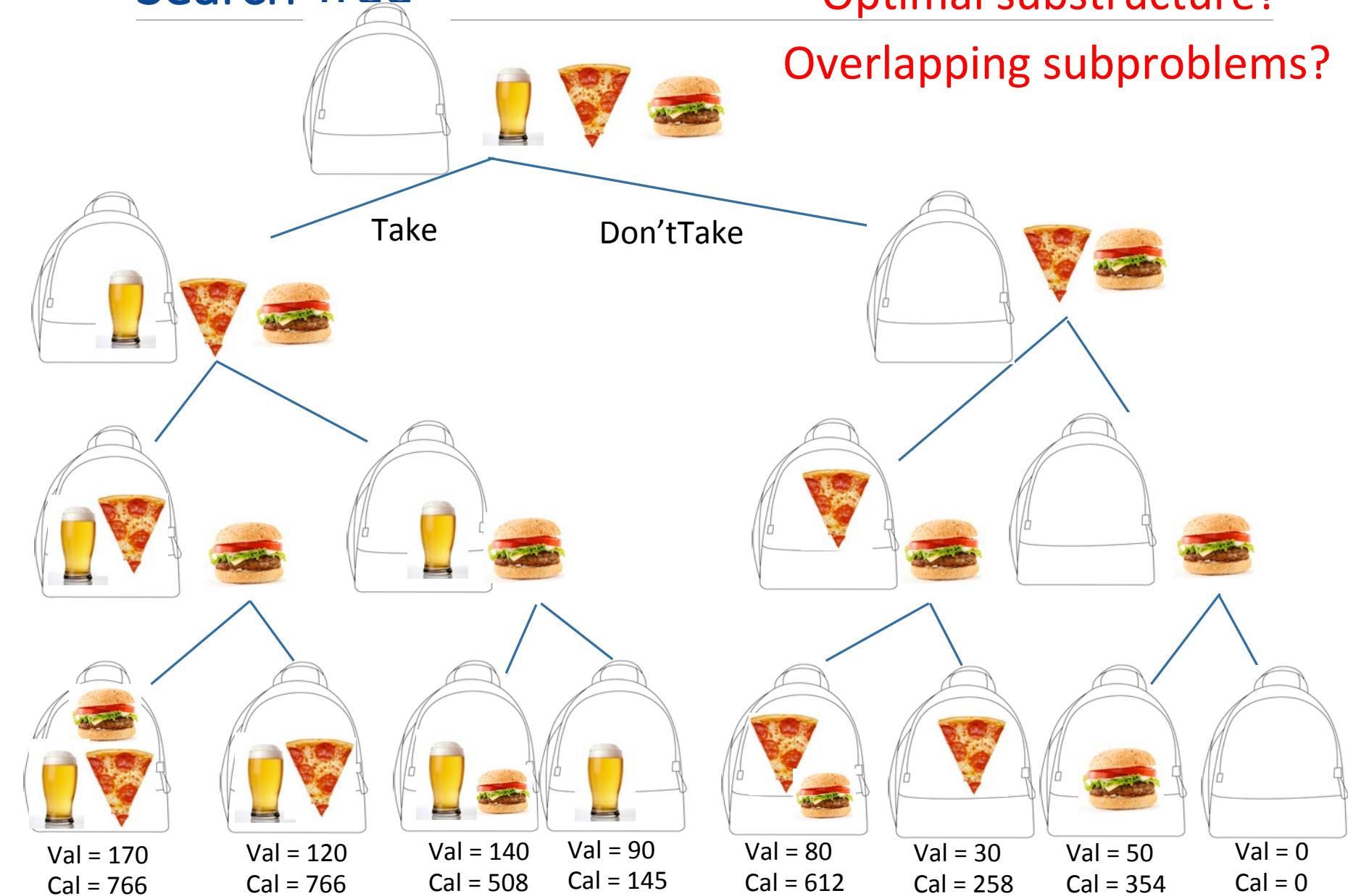
- Do these conditions hold?



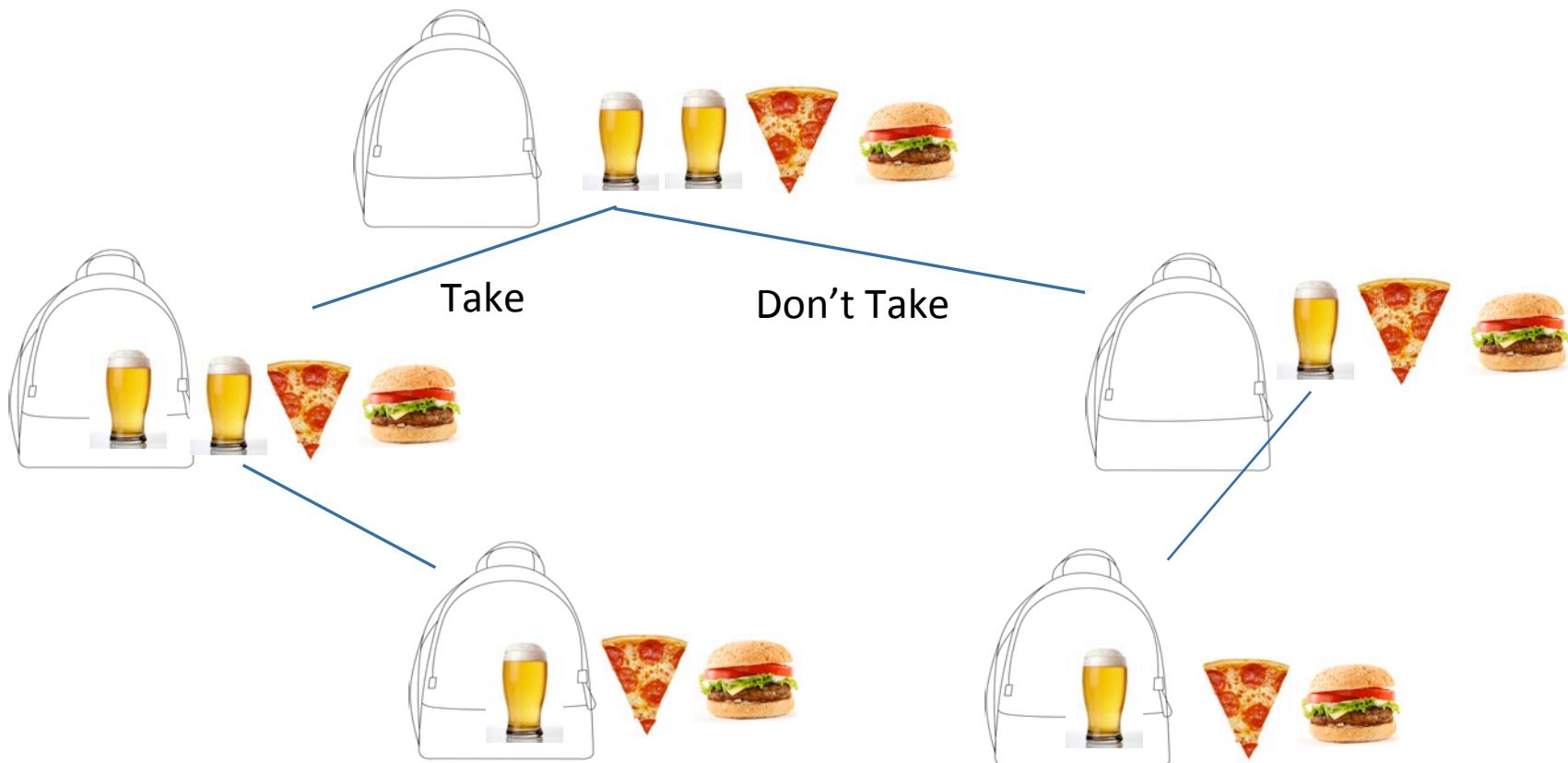
Questions 2 and 3

Search Tree

Optimal substructure?
Overlapping subproblems?



A Different Menu

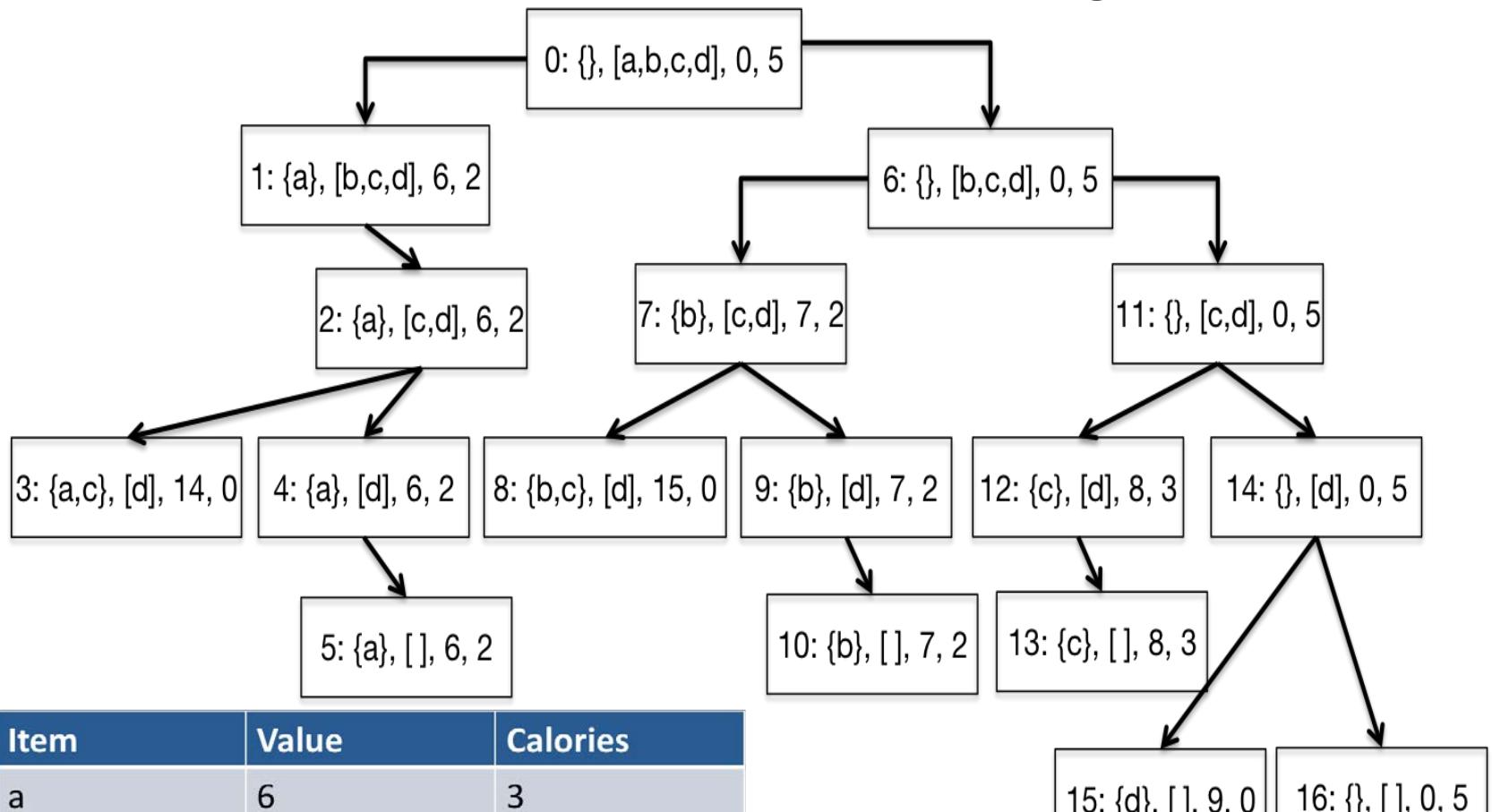


Need Not Have Copies of Items

Item	Value	Calories
a	6	3
b	7	3
c	8	2
d	9	5

Search Tree

- Each node = <taken, left, value, remaining calories>

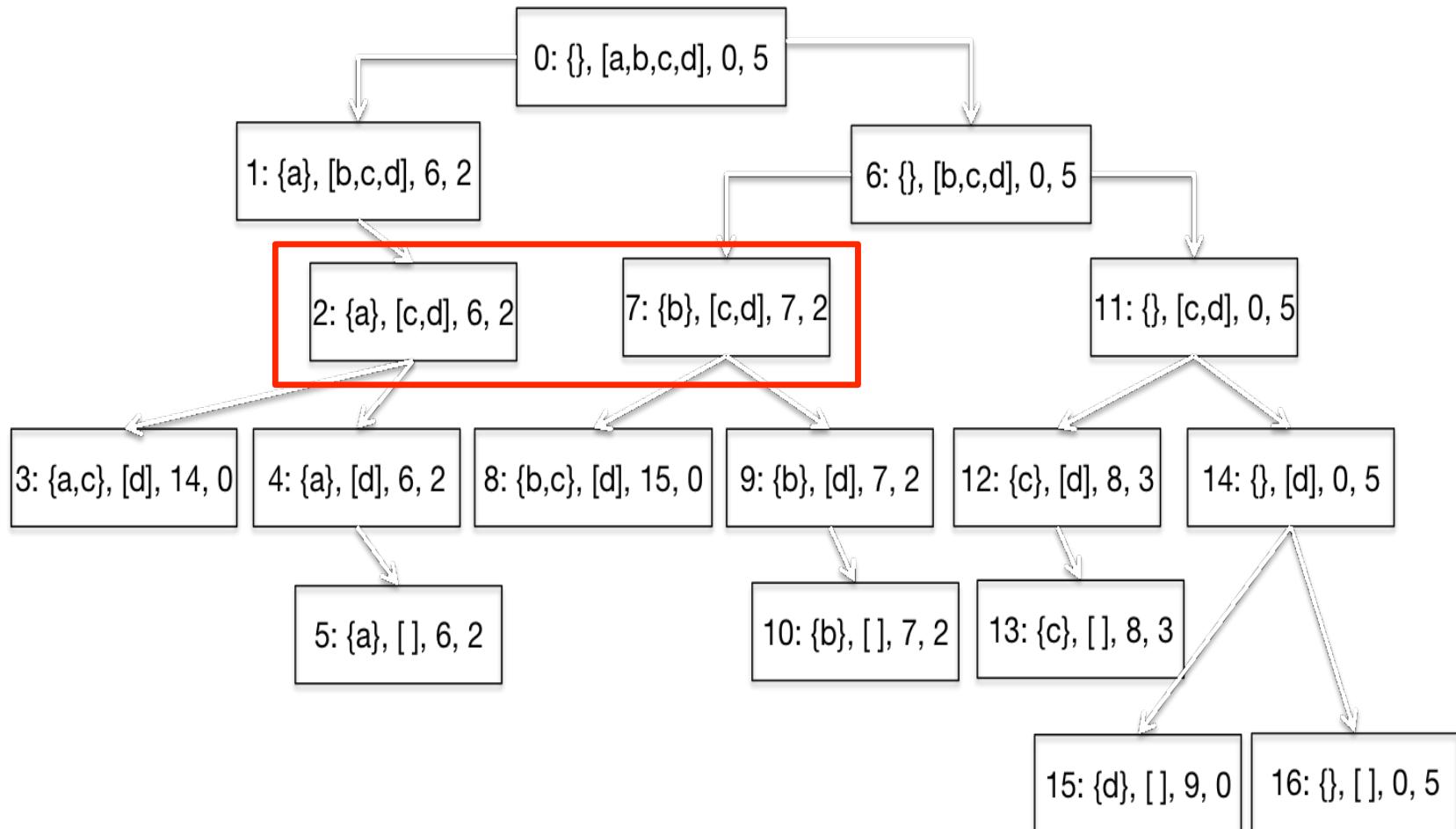


Item	Value	Calories
a	6	3
b	7	3
c	8	2
d	9	5

What Problem is Solved at Each Node?

- Given remaining weight, maximize value by choosing among remaining items
- Set of previously chosen items, or even value of that set, doesn't matter!

Overlapping Subproblems



Modify maxVal to Use a Memo

- Add memo as a third argument
 - `def fastMaxVal(toConsider, avail, memo = {}):`
- Key of memo is a tuple
 - (items left to be considered, available weight)
 - Items left to be considered represented by
`len(toConsider)`
- First thing body of function does is check whether the optimal choice of items given the available weight is already in the memo
- Last thing body of function does is update the memo

Performance

len(items)	$2^{**\text{len(items)}}$	Number of calls
2	4	7
4	16	25
8	256	427
16	65,536	5,191
32	4,294,967,296	22,701
64	18,446,744,073,709 ,551,616	42,569
128	Big	83,319
256	Really Big	176,614
512	Ridiculously big	351,230
1024	Absolutely huge	703,802

How Can This Be?

- Problem is exponential
- Have we overturned the laws of the universe?
- Is dynamic programming a miracle?
- No, but computational complexity can be subtle
- Running time of `fastMaxVal` is governed by number of distinct pairs, `<toConsider, avail>`
 - Number of possible values of `toConsider` bounded by `len(items)`
 - Possible values of `avail` a bit harder to characterize
 - Bounded by number of distinct sums of weights
 - Covered in more detail in assigned reading

Summary of Lectures 1-2

- Many problems of practical importance can be formulated as **optimization problems**
- **Greedy algorithms** often provide adequate (though not necessarily optimal) solutions
- Finding an optimal solution is usually **exponentially hard**
- But **dynamic programming** often yields good performance for a subclass of optimization problems—those with optimal substructure and overlapping subproblems
 - Solution always correct
 - Fast under the right circumstances

The “Roll-over” Optimization Problem

$$\text{Score} = ((60 - (a+b+c+d+e)) * F + a * ps1 + b * ps2 + c * ps3 + d * ps4 + e * ps5$$

Objective:

Given values for F, ps1, ps2, ps3, ps4, ps5

Find values for a, b, c, d, e that maximize score

Constraints:

a, b, c, d, e are each 10 or 0

$a + b + c + d + e \geq 20$

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Graph-theoretic Models

Eric Grimson

MIT Department Of Electrical Engineering and
Computer Science

Relevant Reading for Today's Lecture

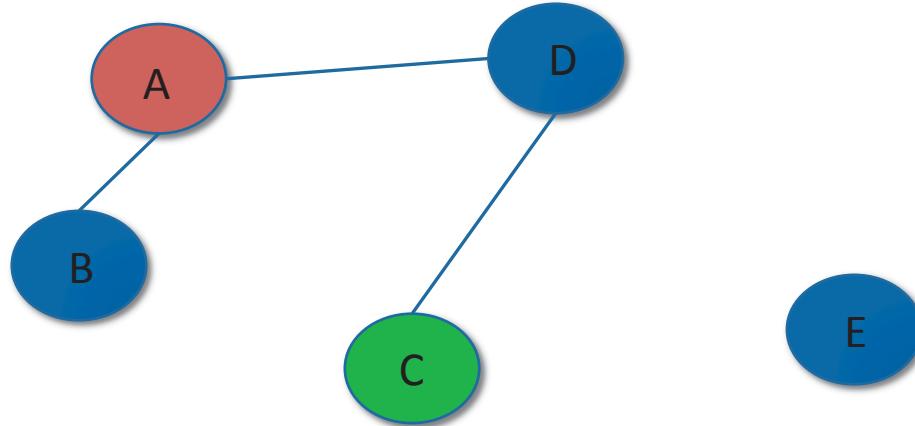
- Section 12.2

Computational Models

- Programs that help us understand the world and solve practical problems
- Saw how we could map the informal problem of choosing what to eat into an optimization problem, and how we could design a program to solve it
- Now want to look at class of models called graphs

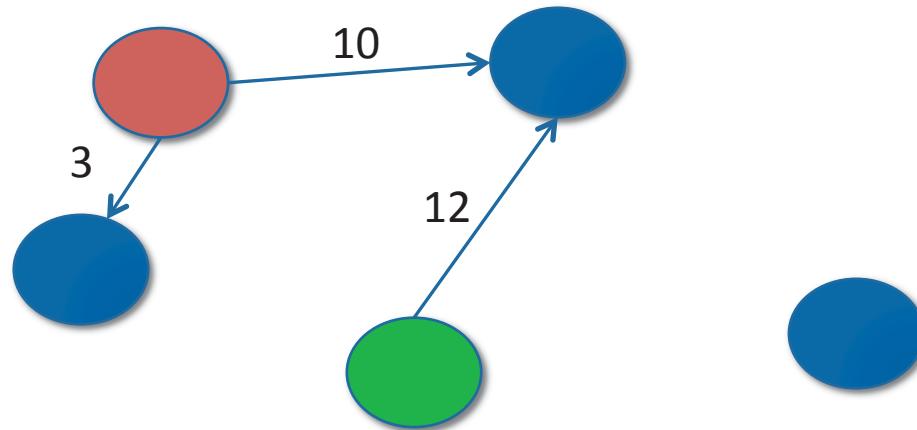
What's a Graph?

- Set of nodes (vertices)
 - Might have properties associated with them
- Set of edges (arcs) each consisting of a pair of nodes
 - Undirected (graph)
 - Directed (digraph)
 - Source (parent) and destination (child) nodes
 - Unweighted or weighted



What's a Graph?

- Set of nodes (vertices)
 - Might have properties associated with them
- Set of edges (arcs) each consisting of a pair of nodes
 - Undirected (graph)
 - Directed (digraph)
 - Source (parent) and destination (child) nodes
 - Unweighted or weighted

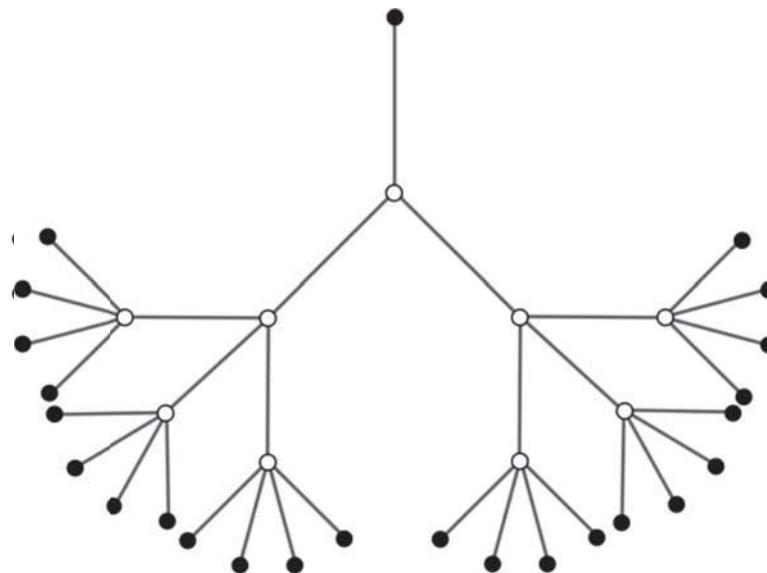


Why Graphs?

- To capture useful relationships among entities
 - Rail links between Paris and London
 - How the atoms in a molecule are related to one another
 - Ancestral relationships

Trees: An Important Special Case

- A special kind of directed graph in which any pair of nodes is connected by a single path
 - Recall the search trees we used to solve knapsack problem

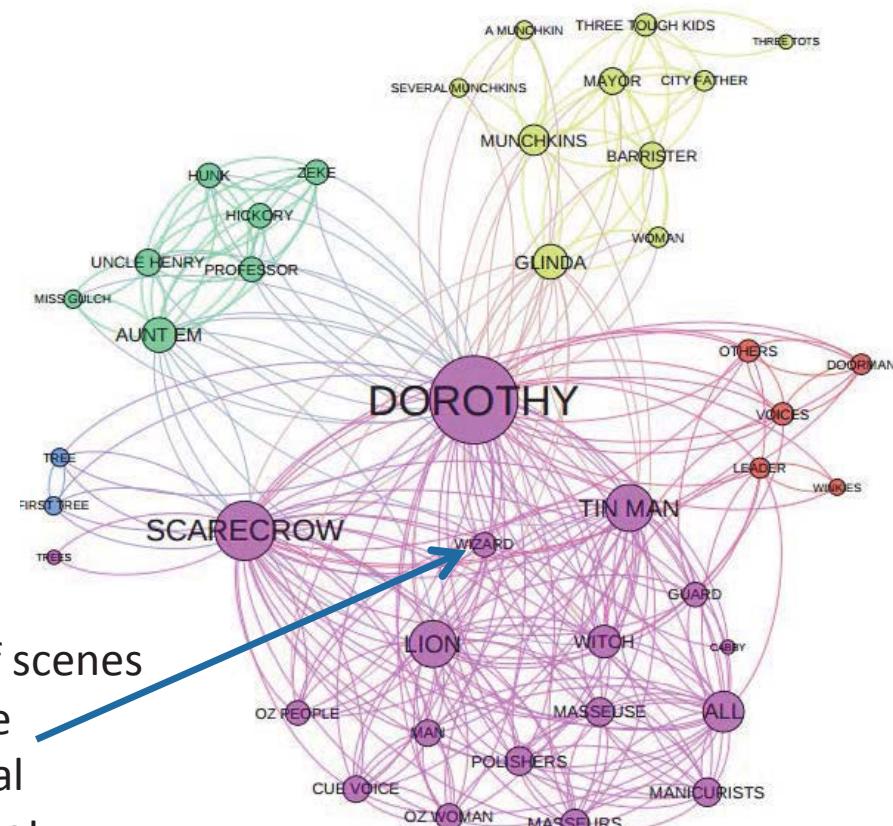


Why Graphs Are So Useful

- World is full of networks based on relationships
 - Computer networks
 - Transportation networks
 - Financial networks
 - Sewer or water networks
 - Political networks
 - Criminal networks
 - Social networks
 - Etc.

Analysis of “Wizard of Oz”:

- size of node reflects number of scenes in which character shares dialogue
- color of clusters reflects natural interactions with each other but not others

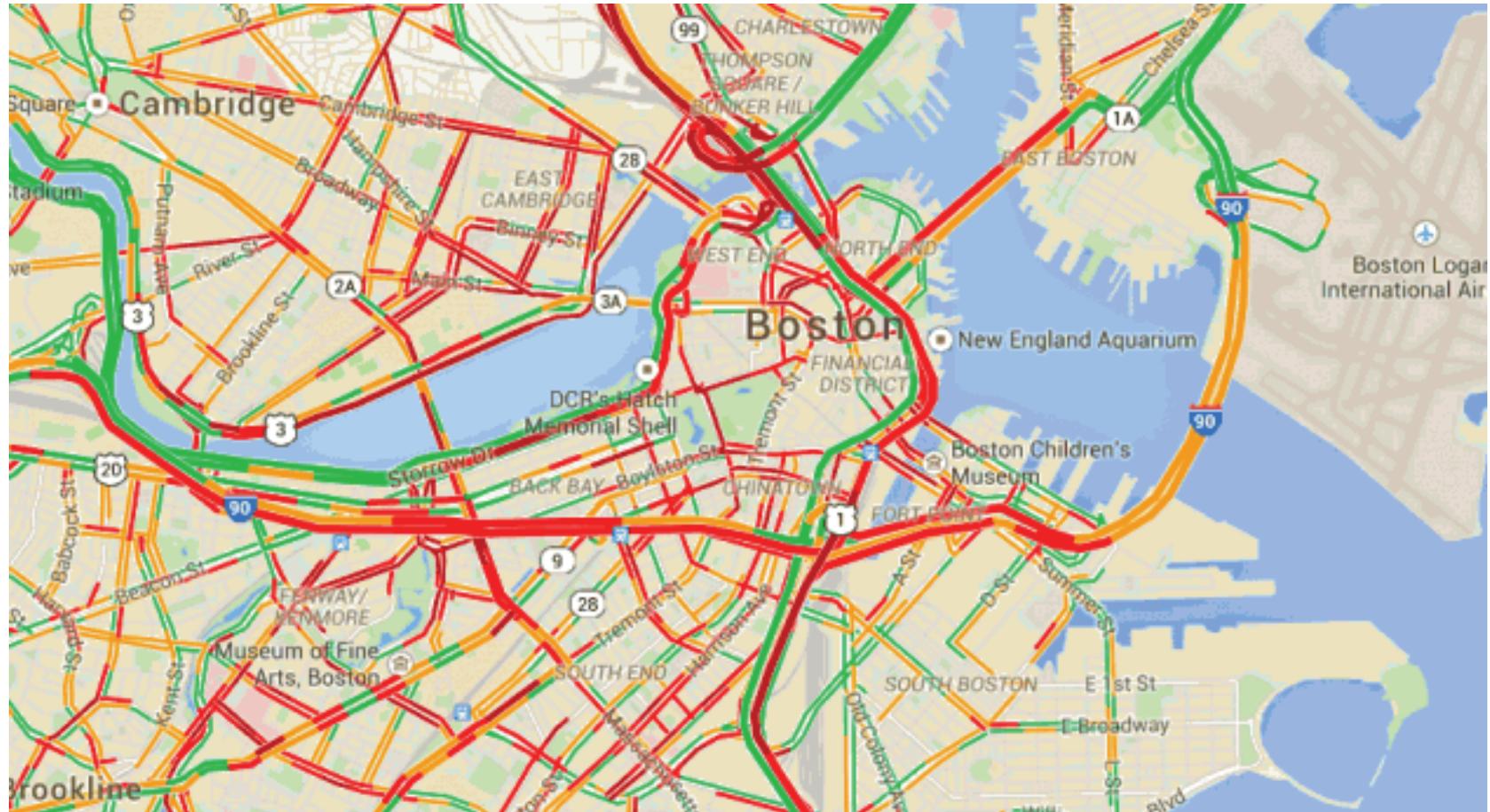


Wizard of Oz dialogue map © Mapr.com. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

Why Graphs Are So Useful

- We will see that not only do graphs capture relationships in connected networks of elements, they also support inference on those structures
 - Finding sequences of links between elements – is there a path from A to B
 - Finding the least expensive path between elements (aka shortest path problem)
 - Partitioning the graph into sets of connected elements (aka graph partition problem)
 - Finding the most efficient way to separate sets of connected elements (aka the min-cut/max-flow problem)

Graph Theory Saves Me Time Every Day



Map image © source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

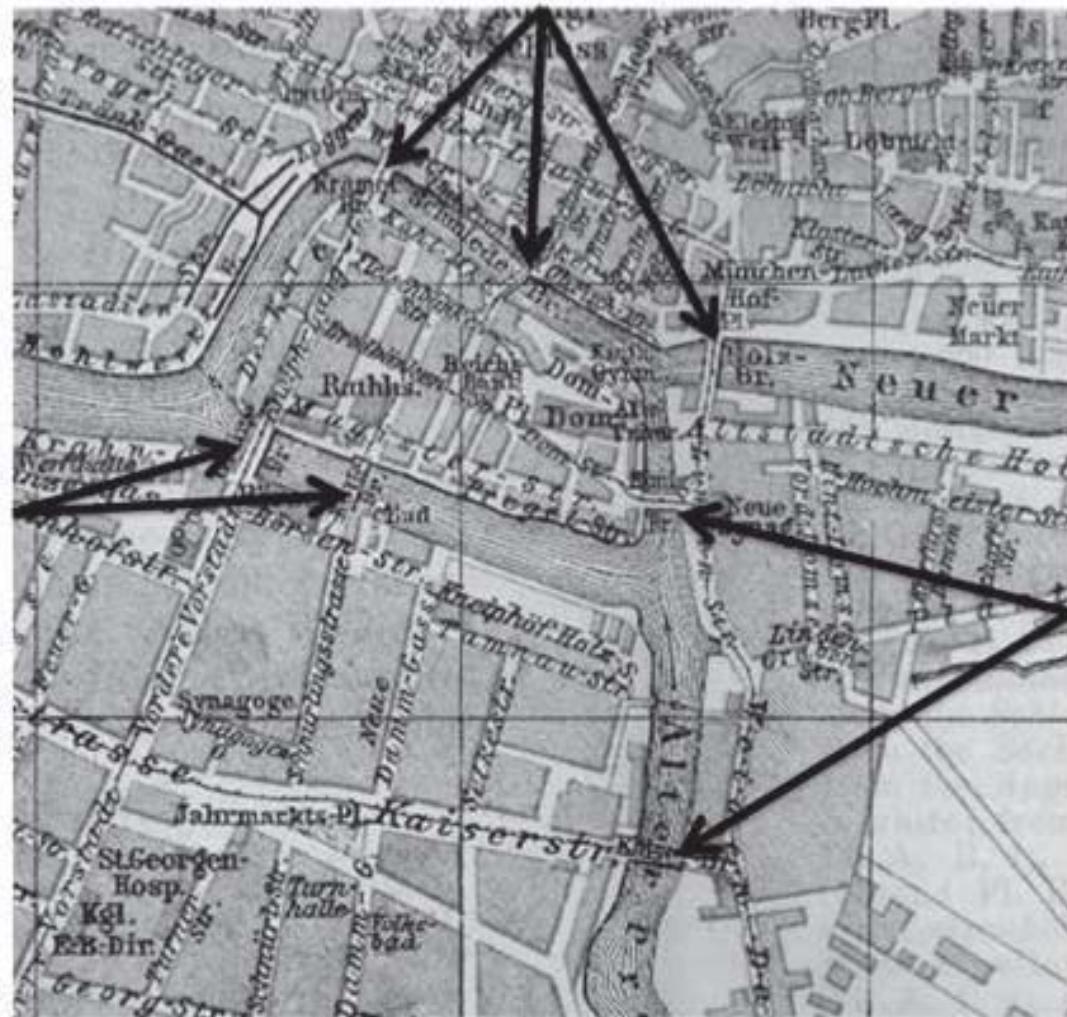
Getting Eric to his Office

- Model road system using a digraph
 - Nodes: points where roads end or meet
 - Edges: connections between points
 - Each edge has a weight
 - Expected time to get from source node to destination node for that edge
 - Distance between source and destination nodes
 - Average speed of travel between source and destination nodes
- Solve a graph optimization problem
 - Shortest weighted path between my house and my office



First Reported Use of Graph Theory

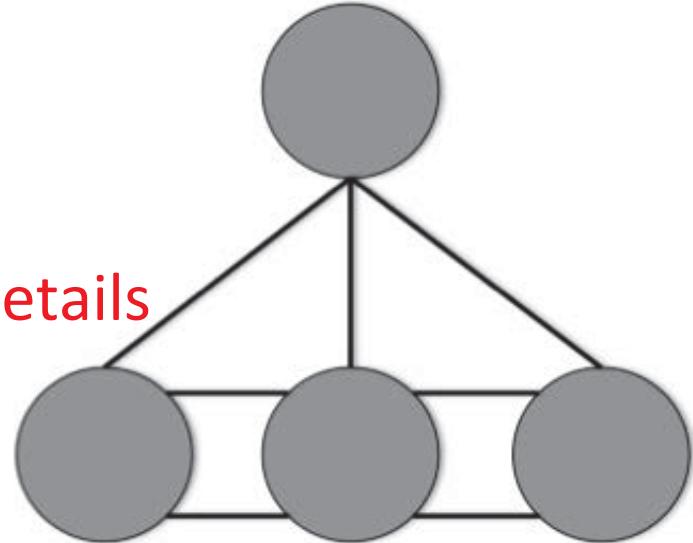
- Bridges of Königsberg (1735)
- Possible to take a walk that traverses each of the 7 bridges exactly once?



Map image © source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

Leonhard Euler's Model

- Each island a node
- Each bridge an undirected edge
- Model abstracts away irrelevant details
 - Size of islands
 - Length of bridges
- Is there a path that contains each edge exactly once?
 - No!



Implementing and using graphs

- Building graphs

- Nodes
- Edges
- Stitching together to make graphs

- Using graphs

- Searching for paths between nodes
- Searching for optimal paths between nodes

Class Node

```
class Node(object):
    def __init__(self, name):
        """Assumes name is a string"""
        self.name = name
    def getName(self):
        return self.name
    def __str__(self):
        return self.name
```

Class Edge

```
class Edge(object):
    def __init__(self, src, dest):
        """Assumes src and dest are nodes"""
        self.src = src
        self.dest = dest
    def getSource(self):
        return self.src
    def getDestination(self):
        return self.dest
    def __str__(self):
        return self.src.getName() + '->' \
            + self.dest.getName()
```

Common Representations of Digraphs

- Digraph is a directed graph
 - Edges pass in one direction only
- Adjacency matrix
 - Rows: source nodes
 - Columns: destination nodes
 - $\text{Cell}[s, d] = 1$ if there is an edge from s to d
= 0 otherwise
 - Note that in digraph, matrix is **not** symmetric
- Adjacency list
 - Associate with each node a list of destination nodes

Class Digraph, part 1

```
class Digraph(object):
    """edges is a dict mapping each node to a list of
    its children"""

    def __init__(self):
        self.edges = {}

    def addNode(self, node):
        if node in self.edges:
            raise ValueError('Duplicate node')
        else:
            self.edges[node] = []

    def addEdge(self, edge):
        src = edge.getSource()
        dest = edge.getDestination()
        if not (src in self.edges and dest in self.edges):
            raise ValueError('Node not in graph')
        self.edges[src].append(dest)
```

Nodes are represented as keys in dictionary

Edges are represented by destinations as values in list associated with a source key

Class Digraph, part 2

```
def childrenOf(self, node):
    return self.edges[node]

def hasNode(self, node):
    return node in self.edges

def getNode(self, name):
    for n in self.edges:
        if n.getName() == name:
            return n
    raise NameError(name)

def __str__(self):
    result = ''
    for src in self.edges:
        for dest in self.edges[src]:
            result = result + src.getName() + '->' \
                    + dest.getName() + '\n'
    return result[:-1] #omit final newline
```

Class Graph

```
class Graph(Digraph):
    def addEdge(self, edge):
        Digraph.addEdge(self, edge)
        rev = Edge(edge.getDestination(), edge.getSource())
        Digraph.addEdge(self, rev)
```

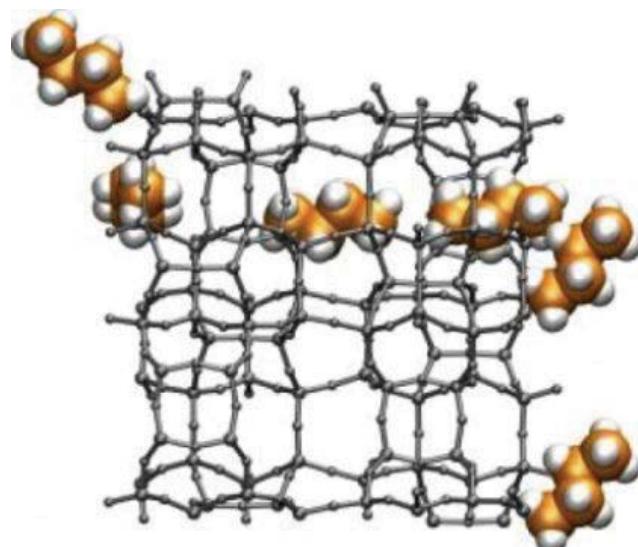
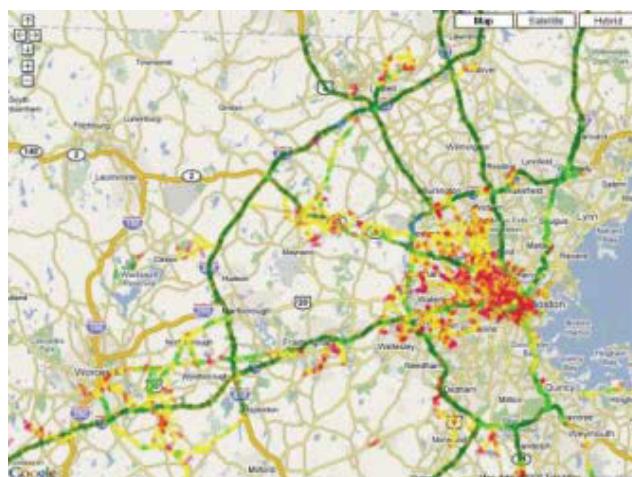
- Graph does not have directionality associated with an edge
 - Edges allow passage in either direction
- Why is Graph a subclass of Digraph?
- Remember the substitution rule?
 - If client code works correctly using an instance of the supertype, it should also work correctly when an instance of the subtype is substituted for the instance of the supertype
- Any program that works with a Digraph will also work with a Graph (but not *vice versa*)

A Classic Graph Optimization Problem

- Shortest path from n_1 to n_2
 - Shortest sequence of edges such that
 - Source node of first edge is n_1
 - Destination of last edge is n_2
 - For edges, e_1 and e_2 , in the sequence, if e_2 follows e_1 in the sequence, the source of e_2 is the destination of e_1
- Shortest weighted path
 - Minimize the sum of the weights of the edges in the path

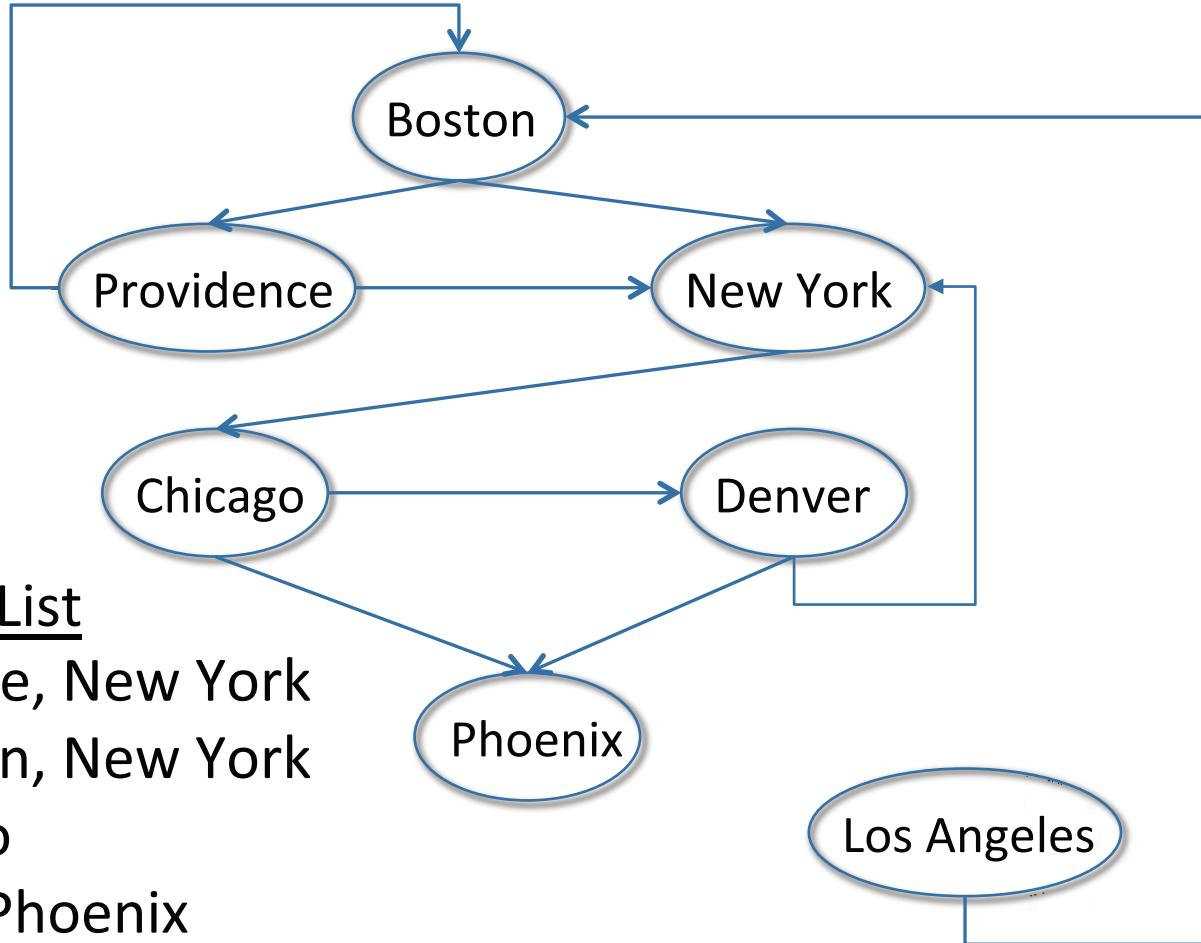
Some Shortest Path Problems

- Finding a route from one city to another
- Designing communication networks
- Finding a path for a molecule through a chemical labyrinth
- ...



Images © sources unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use>.

An Example



Adjacency List

Boston: Providence, New York

Providence: Boston, New York

New York: Chicago

Chicago: Denver, Phoenix

Denver: Phoenix, New York

Los Angeles: Boston

Phoenix:

Build the Graph

```
def buildCityGraph(graphType):
    g = graphType()
    for name in ('Boston', 'Providence', 'New York', 'Chicago',
                 'Denver', 'Phoenix', 'Los Angeles'): #Create 7 nodes
        g.addNode(Node(name))
    g.addEdge(Edge(g.getNode('Boston'), g.getNode('Providence')))
    g.addEdge(Edge(g.getNode('Boston'), g.getNode('New York')))
    g.addEdge(Edge(g.getNode('Providence'), g.getNode('Boston')))
    g.addEdge(Edge(g.getNode('Providence'), g.getNode('New York')))
    g.addEdge(Edge(g.getNode('New York'), g.getNode('Chicago')))
    g.addEdge(Edge(g.getNode('Chicago'), g.getNode('Denver')))
    g.addEdge(Edge(g.getNode('Chicago'), g.getNode('Phoenix')))
    g.addEdge(Edge(g.getNode('Denver'), g.getNode('Phoenix')))
    g.addEdge(Edge(g.getNode('Denver'), g.getNode('New York')))
    g.addEdge(Edge(g.getNode('Los Angeles'), g.getNode('Boston')))
```

Finding the Shortest Path

- Algorithm 1, depth-first search (DFS)
- Similar to left-first depth-first method of enumerating a search tree (Lecture 2)
- Main difference is that graph might have cycles, so we must keep track of what nodes we have visited to avoid going in infinite loops

Note that we are using divide-and-conquer: if we can find a path from a source to an intermediate node, and a path from the intermediate node to the destination, the combination is a path from source to destination

Depth First Search



- Start at an initial node
- Consider all the edges that leave that node, in some order
- Follow the first edge, and check to see if at goal node
- If not, repeat the process from new node
- Continue until either find goal node, or run out of options
 - When run out of options, backtrack to the previous node and try the next edge, repeating this process

Depth First Search (DFS)

```
def DFS(graph, start, end, path, shortest, toPrint = False):
    path = path + [start]
    if toPrint:
        print('Current DFS path:', printPath(path))
    if start == end:
        return path
    for node in graph.childrenOf(start):
        if node not in path: #avoid cycles
            if shortest == None or len(path) < len(shortest):
                newPath = DFS(graph, node, end, path, shortest, toPrint)
                if newPath != None:
                    shortest = newPath
    elif toPrint:
        print('Already visited', node)
    return shortest
```

```
def shortestPath(graph, start, end, toPrint = False):
    return DFS(graph, start, end, [], None, toPrint)
```

DFS called from a
wrapper function:
shortestPath

... returning to this point in the recursion to try next node
Note how will explore all paths through first node, before ...

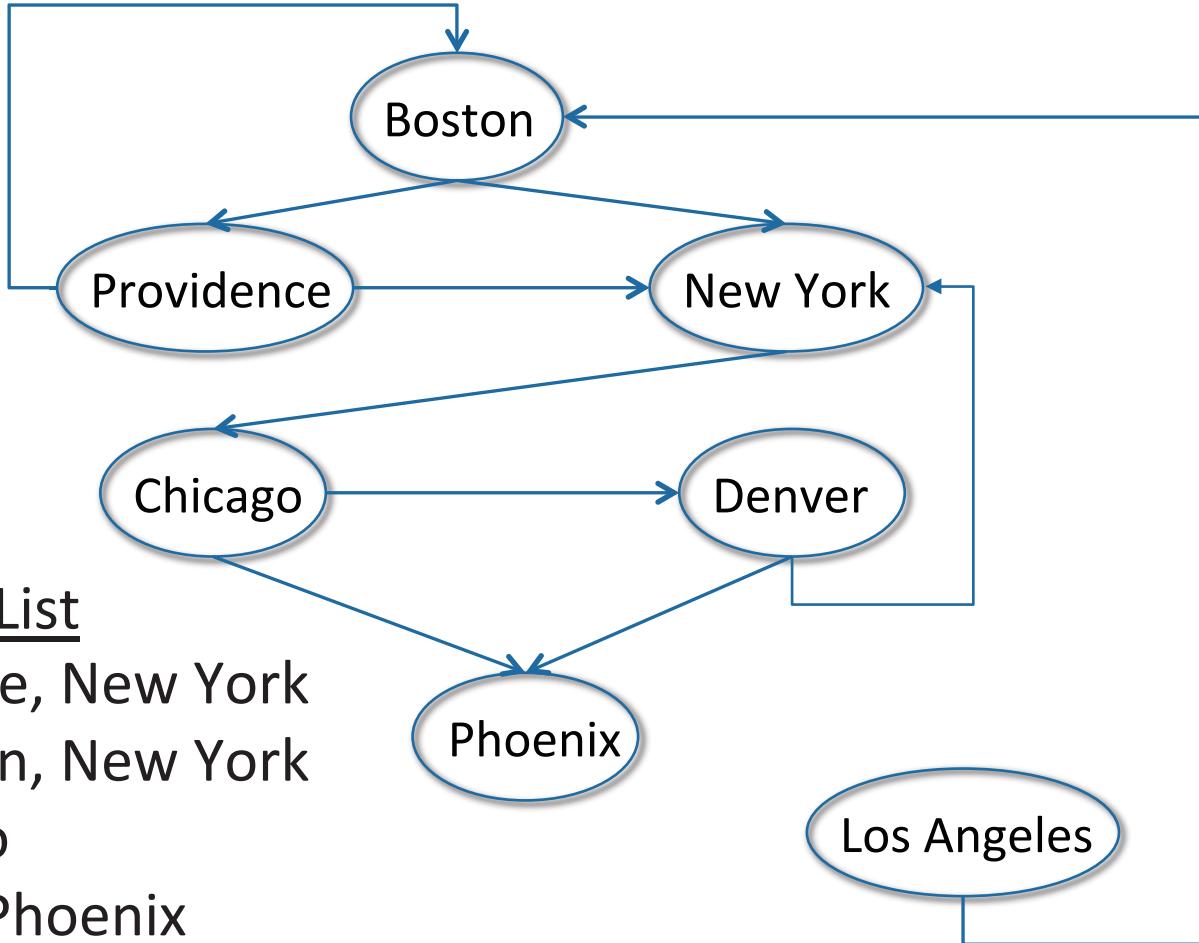
Gets recursion started properly
Provides appropriate abstraction

Test DFS

```
def testSP(source, destination):
    g = buildCityGraph(DiGraph)
    sp = shortestPath(g, g.getNode(source), g.getNode(destination),
                      toPrint = True)
    if sp != None:
        print('Shortest path from', source, 'to',
              destination, 'is', printPath(sp))
    else:
        print('There is no path from', source, 'to', destination)

testSP('Boston', 'Chicago')
```

An Example



Adjacency List

Boston: Providence, New York

Providence: Boston, New York

New York: Chicago

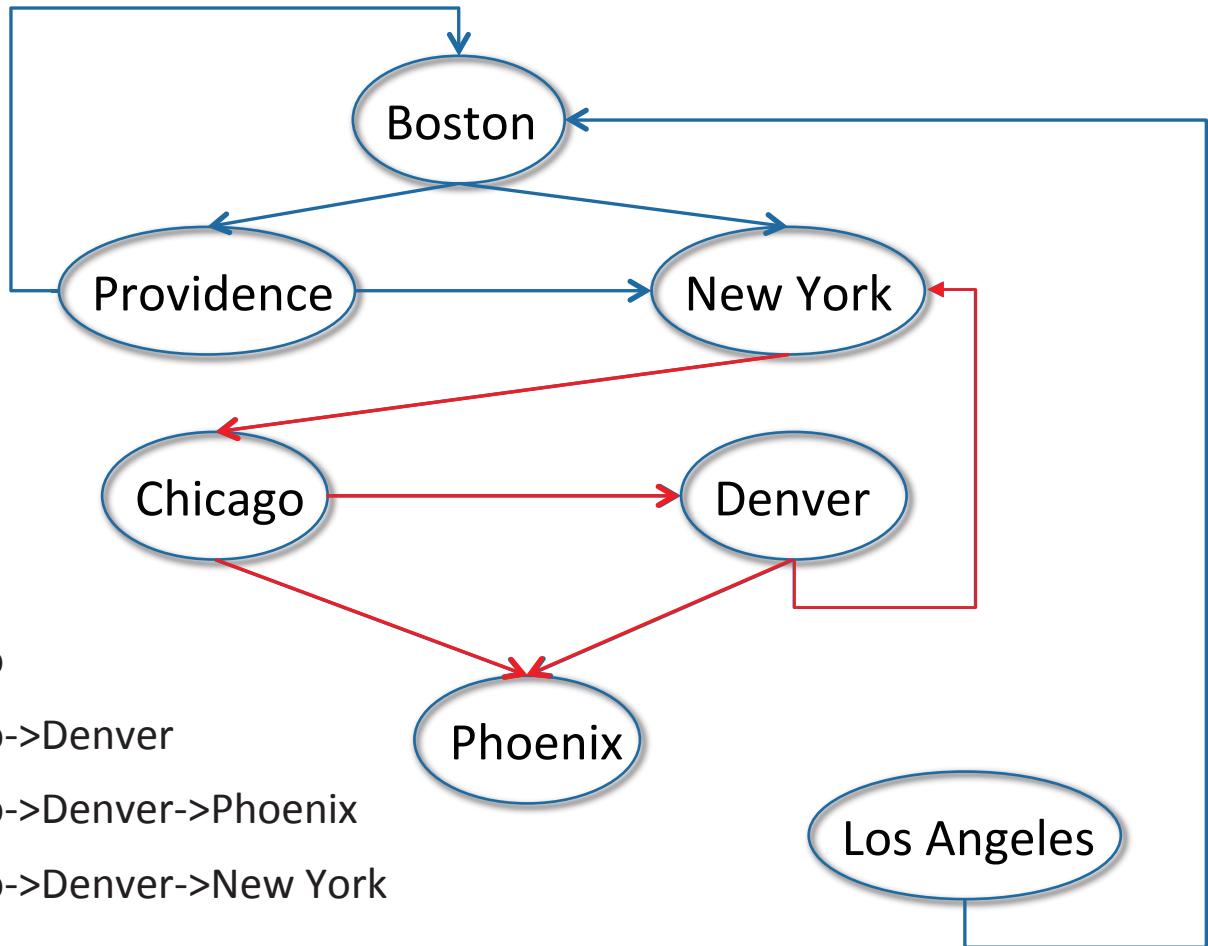
Chicago: Denver, Phoenix

Denver: Phoenix, New York

Los Angeles: Boston

Phoenix:

Output (Chicago to Boston)



Current DFS path: Chicago

Current DFS path: Chicago->Denver

Current DFS path: Chicago->Denver->Phoenix

Current DFS path: Chicago->Denver->New York

Already visited Chicago

Current DFS path: Chicago->Phoenix

There is no path from Chicago to Boston

Output (Boston to Phoenix)

Current DFS path: Boston

Current DFS path: Boston->Providence

Already visited Boston

Current DFS path: Boston->Providence->New York

Current DFS path: Boston->Providence->New York->Chicago

Current DFS path: Boston->Providence->New York->Chicago->Denver

Current DFS path: Boston->Providence->New York->Chicago->Denver->Phoenix **Found path**

Already visited New York

Current DFS path: Boston->Providence->New York->Chicago->Phoenix **Found a shorter path**

Current DFS path: Boston->New York

Current DFS path: Boston->New York->Chicago

Current DFS path: Boston->New York->Chicago->Denver

Current DFS path: Boston->New York->Chicago->Denver->Phoenix **Found a “shorter” path**

Already visited New York

Current DFS path: Boston->New York->Chicago->Phoenix **Found a shorter path**

Shortest path from Boston to Phoenix is Boston->New York->Chicago->Denver->Phoenix

Breadth First Search



- Start at an initial node
- Consider all the edges that leave that node, in some order
- Follow the first edge, and check to see if at goal node
- If not, try the next edge from the current node
- Continue until either find goal node, or run out of options
 - When run out of edge options, move to next node at same distance from start, and repeat
 - When run out of node options, move to next level in the graph (all nodes one step further from start), and repeat

Algorithm 2: Breadth-first Search (BFS)

```
def BFS(graph, start, end, toPrint = False):
    initPath = [start]
    pathQueue = [initPath]
    while len(pathQueue) != 0:
        #Get and remove oldest element in pathQueue
        tmpPath = pathQueue.pop(0)
        if toPrint:
            print('Current BFS path:', printPath(tmpPath))
        lastNode = tmpPath[-1]
        if lastNode == end:
            return tmpPath
        for nextNode in graph.childrenOf(lastNode):
            if nextNode not in tmpPath:
                newPath = tmpPath + [nextNode]
                pathQueue.append(newPath)
    return None
```

Explore all paths with n hops before
exploring any path with more than n hops

Output (Boston to Phoenix)

Current BFS path: Boston

Current BFS path: Boston->Providence

Current BFS path: Boston->New York

Current BFS path: Boston->Providence->New York

Current BFS path: Boston->New York->Chicago

Current BFS path: Boston->Providence->New York->Chicago

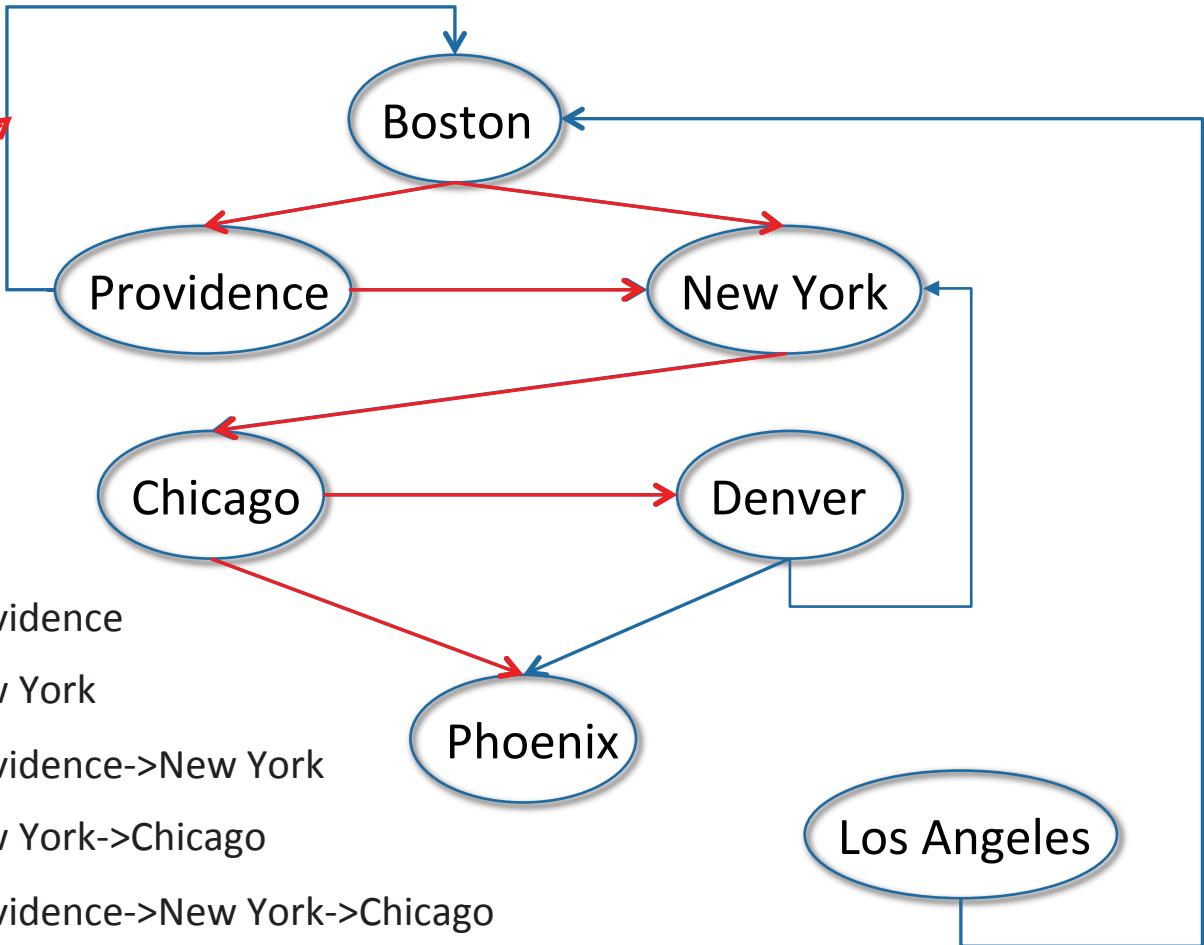
Current BFS path: Boston->New York->Chicago->Denver

Current BFS path: Boston->New York->Chicago->Phoenix

Shortest path from Boston to Phoenix is Boston->New York->Chicago->Phoenix

Output (Boston to Phoenix)

Note that we skip a path that revisits a node



Current BFS path: Boston

Current BFS path: Boston->Providence

Current BFS path: Boston->New York

Current BFS path: Boston->Providence->New York

Current BFS path: Boston->New York->Chicago

Current BFS path: Boston->Providence->New York->Chicago

Current BFS path: Boston->New York->Chicago->Denver

Current BFS path: Boston->New York->Chicago->Phoenix

Shortest path from Boston to Phoenix is Boston->New York->Chicago->Phoenix

What About a Weighted Shortest Path

- Want to minimize the sum of the weights of the edges, not the number of edges
- DFS can be easily modified to do this
- BFS cannot, since shortest weighted path may have more than the minimum number of hops

Recap

- Graphs are cool
 - Best way to create a model of many things
 - Capture relationships among objects
 - Many important problems can be posed as graph optimization problems we already know how to solve
- Depth-first and breadth-first search are important algorithms
 - Can be used to solve many problems

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Lecture 4: Stochastic Thinking and Random Walks

Relevant Reading

- Pages 235-238
- Chapter 14

The World is Hard to Understand

- Uncertainty is uncomfortable
- But certainty is usually unjustified

Newtonian Mechanics

- Every effect has a cause
- The world can be understood causally

Copenhagen Doctrine

- Copenhagen Doctrine (Bohr and Heisenberg) of **causal nondeterminism**
 - At its most fundamental level, the behavior of the physical world cannot be predicted.
 - Fine to make statements of the form “x is highly likely to occur,” but not of the form “x is certain to occur.”
- Einstein and Schrödinger objected
 - “God does not play dice.” -- Albert Einstein

Does It Really Matter

Did the flips yield
2 heads
2 tails
1 head and 1 tail?

The Moral

- The world may or may not be inherently unpredictable
- But our lack of knowledge does not allow us to make accurate predictions
- Therefore we might as well treat the world as inherently unpredictable
- **Predictive nondeterminism**

Stochastic Processes

- An ongoing process where the next state might depend on both the previous states **and some random element**

```
def rollDie():
    """ returns an int between 1 and 6"""


```

```
def rollDie():
    """ returns a randomly chosen int
        between 1 and 6"""


```

Implementing a Random Process

```
import random

def rollDie():
    """returns a random int between 1 and 6"""
    return random.choice([1,2,3,4,5,6])

def testRoll(n = 10):
    result = ''
    for i in range(n):
        result = result + str(rollDie())
    print(result)
```

Probability of Various Results

- Consider `testRoll(5)`
- How probable is the output 11111?

Probability Is About Counting

- Count the number of possible events
- Count the number of events that have the property of interest
- Divide one by the other
- Probability of 11111?
 - 11111, 11112, 11113, ..., 11121, 11122, ..., 66666
 - $1/(6^{**}5)$
 - ~ 0.0001286

Three Basic Facts About Probability

- Probabilities are always in the range **0 to 1**. 0 if impossible, and 1 if guaranteed.
- If the probability of an event occurring is p , the probability of it not occurring must be
- When events are **independent** of each other, the probability of all of the events occurring is equal to a **product** of the probabilities of each of the events occurring.

Independence

- Two events are **independent** if the outcome of one event has no influence on the outcome of the other
- Independence should not be taken for granted

Will One of the Patriots and Broncos Lose?

- Patriots have winning percentage of $7/8$, Broncos of $6/8$
- Probability of both winning next Sunday is $7/8 * 6/8 = 42/64$
- Probability of at least one losing is $1 - 42/64 = 22/64$
- What about Sunday, December 18
 - Outcomes are not independent
 - Probability of one of them losing is much closer to 1 than to $22/64$!

A Simulation of Die Rolling

```
def runSim(goal, numTrials, txt):
    total = 0
    for i in range(numTrials):
        result = ''
        for j in range(len(goal)):
            result += str(rollDie())
        if result == goal:
            total += 1
    print('Actual probability of', txt, '=', round(1/(6**len(goal)), 8))
    estProbability = round(total/numTrials, 8)
    print('Estimated Probability of', txt, '=', round(estProbability, 8))

runSim('11111', 1000, '11111')
```

Output of Simulation

- Actual probability = 0.0001286
 - Estimated Probability = 0.0
 - Actual probability = 0.0001286
 - Estimated Probability = 0.0
-
- How did I **know** that this is what would get printed?
 - Why did simulation give me the **wrong** answer?

Let's try 1,000,000 trials

Morals

- Moral 1: It takes a lot of trials to get a good estimate of the frequency of occurrence of a rare event. We'll talk lots more in later lectures about how to **know** when we have enough trials.
- Moral 2: One should not confuse the **sample probability** with the actual probability
- Moral 3: There was really no need to do this by simulation, since there is a perfectly good closed form answer. We will see many examples where this is not true.
- But simulations are often useful.

The Birthday Problem

- What's the probability of at least two people in a group having the same birthday
- If there are 367 people in the group?
- What about smaller numbers?
- If we assume that each birthdate is equally likely
 - $1 - \frac{366!}{366^N * (366-N)!}$
- Without this assumption, VERY complicated

shoutkey.com/niece

Approximating Using a Simulation

```
def sameDate(numPeople, numSame):
    possibleDates = range(366)
    birthdays = [0]*366
    for p in range(numPeople):
        birthDate = random.choice(possibleDates)
        birthdays[birthDate] += 1
    return max(birthdays) >= numSame
```

Approximating Using a Simulation

```
def birthdayProb(numPeople, numSame, numTrials):  
    numHits = 0  
    for t in range(numTrials):  
        if sameDate(numPeople, numSame):  
            numHits += 1  
    return numHits/numTrials  
  
for numPeople in [10, 20, 40, 100]:  
    print('For', numPeople,  
          'est. prob. of a shared birthday is',  
          birthdayProb(numPeople, 2, 10000))  
    numerator = math.factorial(366)  
    denom = (366**numPeople)*math.factorial(366-numPeople)  
    print('Actual prob. for N = 100 =',  
          1 - numerator/denom)
```

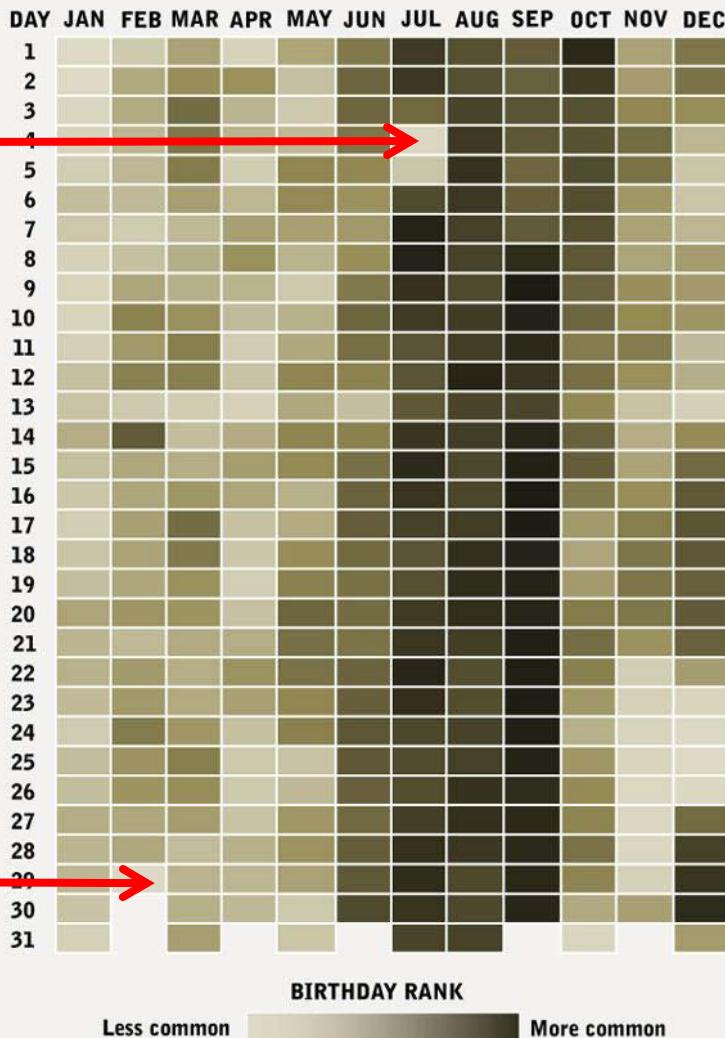
Suppose we want the probability of 3 people sharing

Why 3 Is Much Harder Mathematically

- For 2 the complementary problem is “all birthdays distinct”
- For 3 people, the complementary problem is a complicated disjunct
 - All birthdays distinct or
 - One pair and rest distinct or
 - Two pairs and rest distinct or
 - ...
- But changing the simulation is dead easy

But All Dates Are Not Equally Likely

Which Birth Dates Are Most Common?



Are you exceptional?

Another Win for Simulation

- Adjusting analytic model a pain
- Adjusting simulation model easy

```
def sameDate(numPeople, numSame):
    possibleDates = 4*list(range(0, 57)) + [58]\
                    + 4*list(range(59, 366))\
                    + 4*list(range(180, 270))
    birthdays = [0]*366
    for p in range(numPeople):
        birthDate = random.choice(possibleDates)
        birthdays[birthDate] += 1
    return max(birthdays) >= numSame
```

Simulation Models

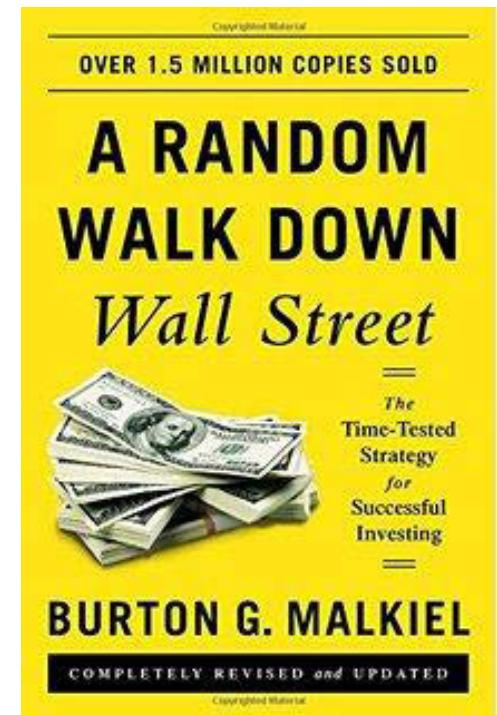
- A description of computations that provide useful information about the possible behaviors of the system being modeled
- Descriptive, not prescriptive
- Only an approximation to reality
- “All models are wrong, but some are useful.” – George Box

Simulations Are Used a Lot

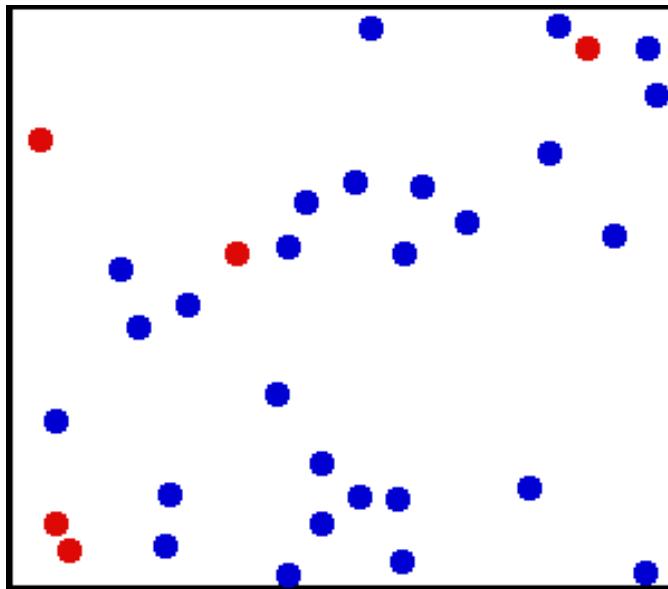
- To model systems that are mathematically intractable
- To extract useful intermediate results
- Lend themselves to development by successive refinement and “what if” questions
- Start by simulating random walks

Why Random Walks?

- Random walks are important in many domains
 - Understanding the stock market (maybe)
 - Modeling diffusion processes
 - Etc.
- Good illustration of how to use simulations to understand things
- Excuse to cover some important programming topics
 - Practice with classes
 - More about plotting



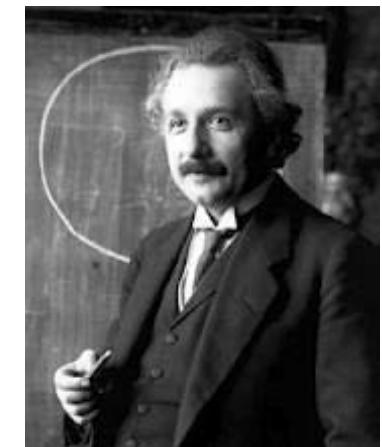
Brownian Motion Is a Random Walk



Robert
Brown
1827



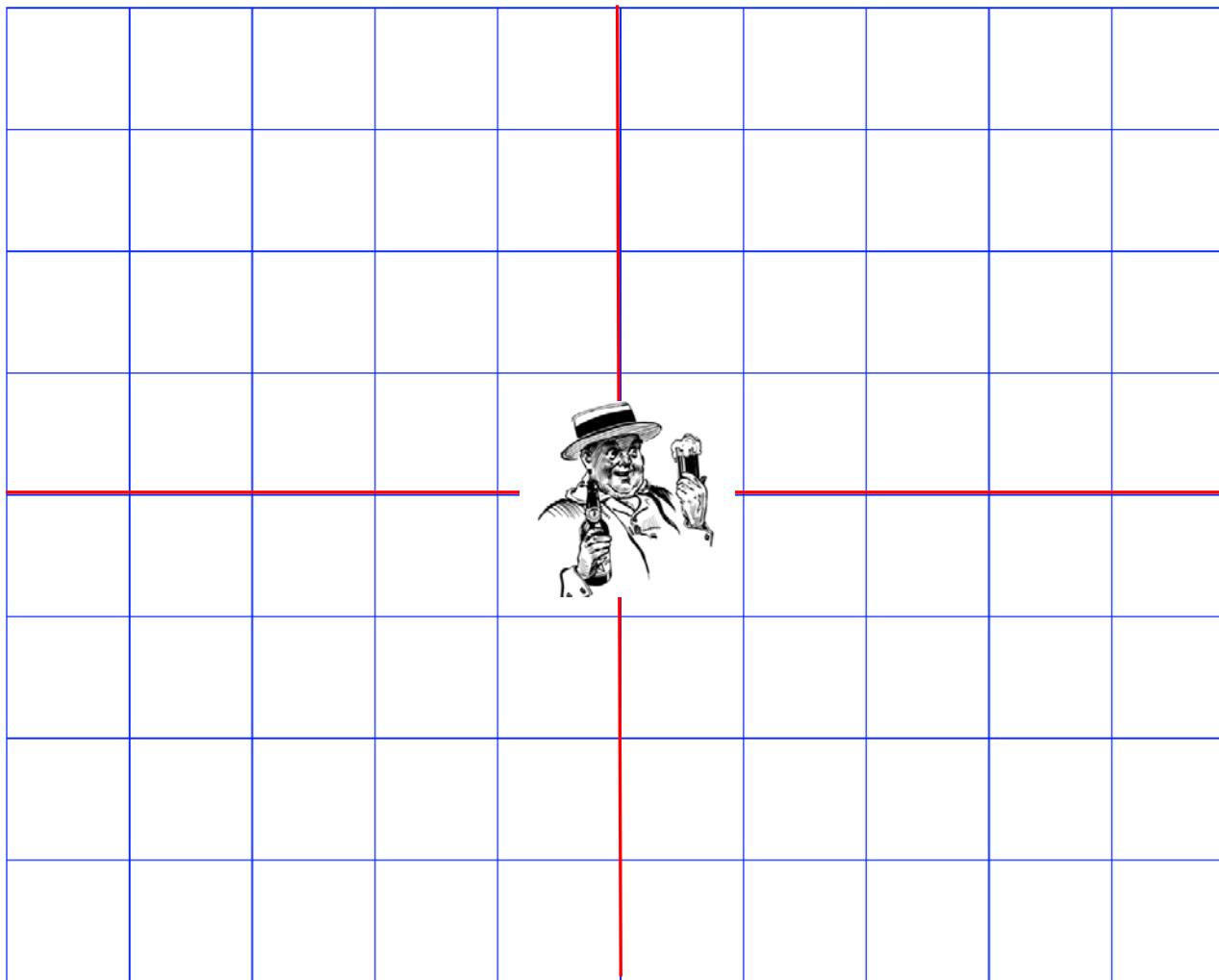
Louis
Bachelier
1900



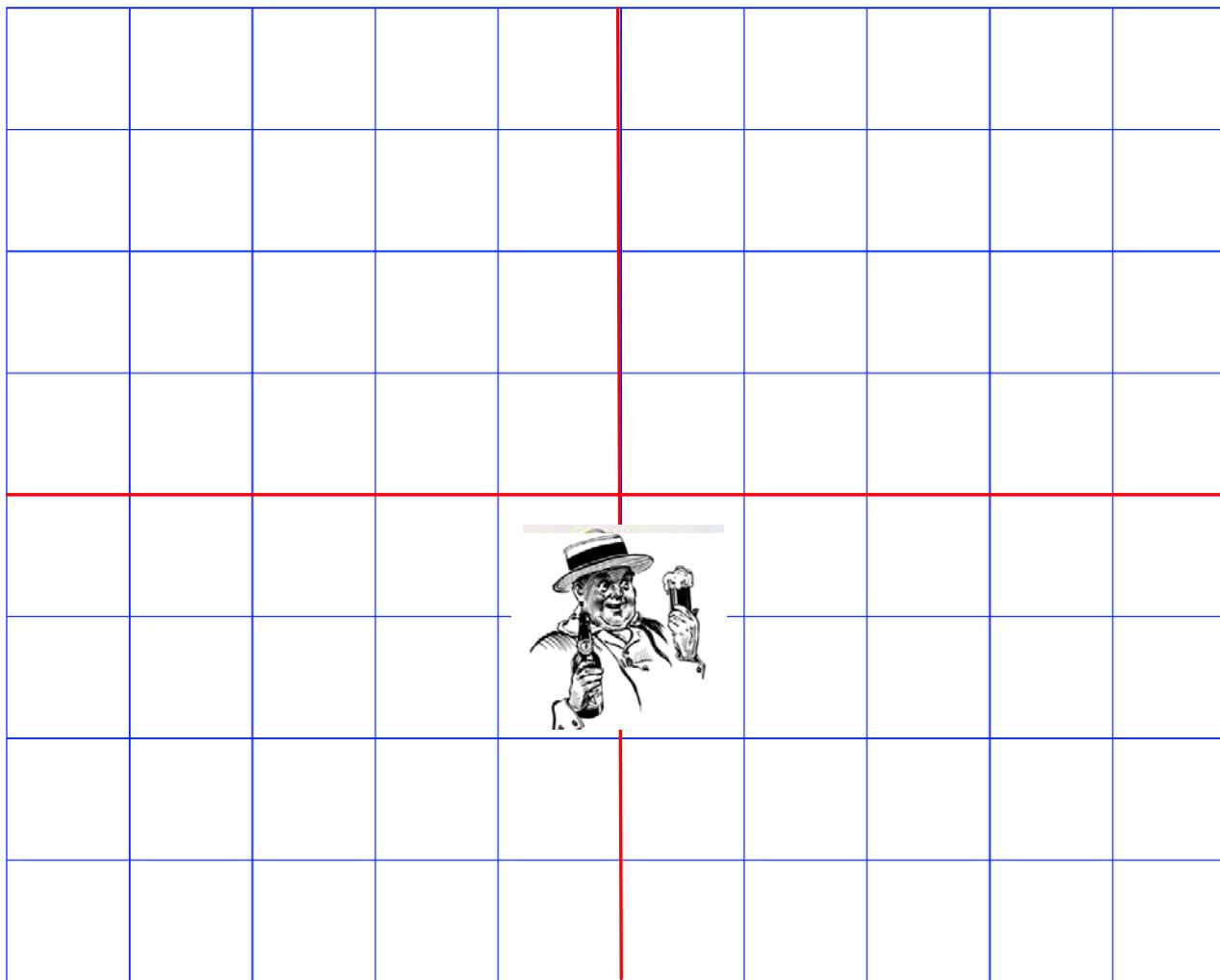
Albert
Einstein
1905

Images of Robert Brown and Albert Einstein are in the public domain. Image of Louis Bachelier © unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.

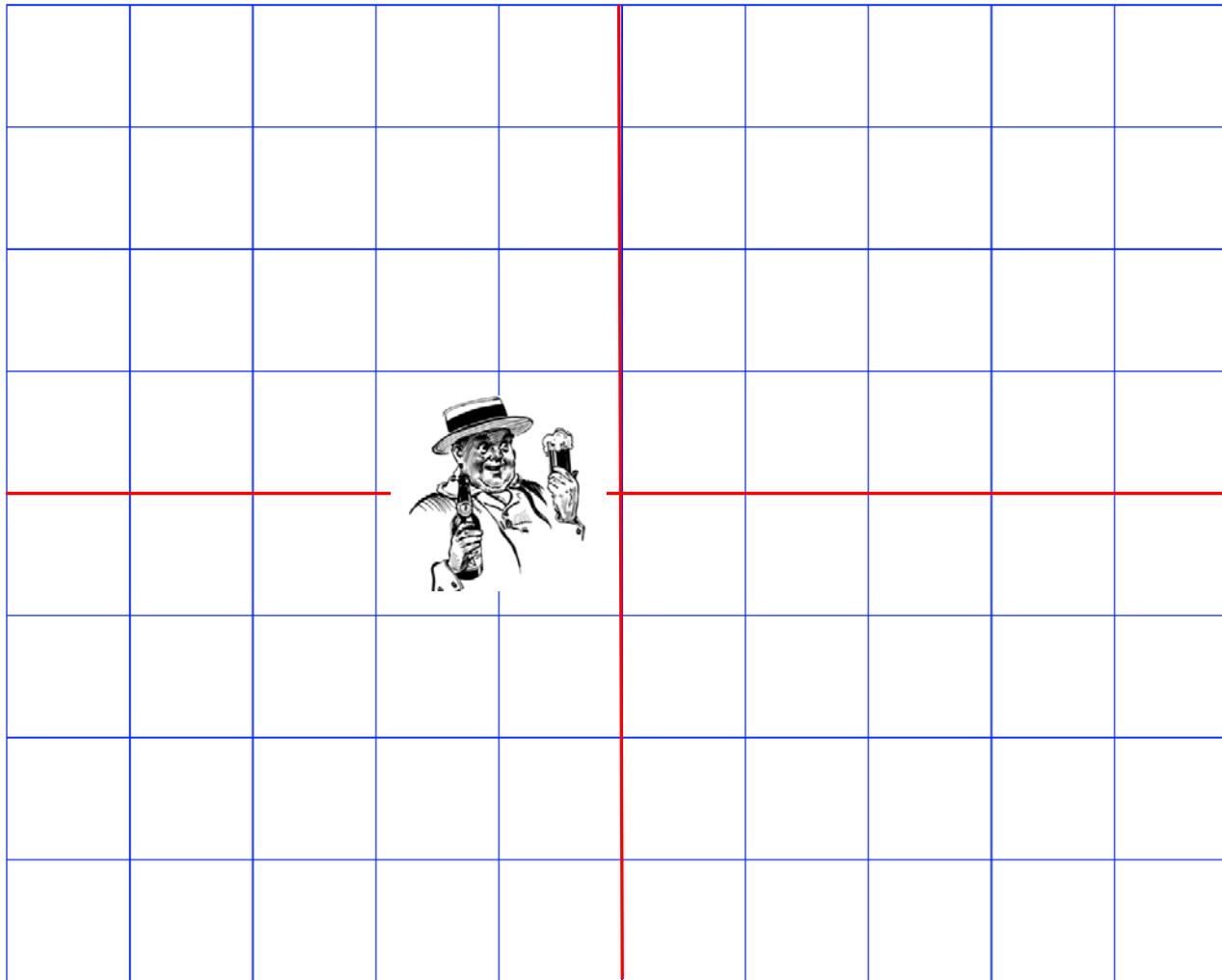
Drunkard's Walk



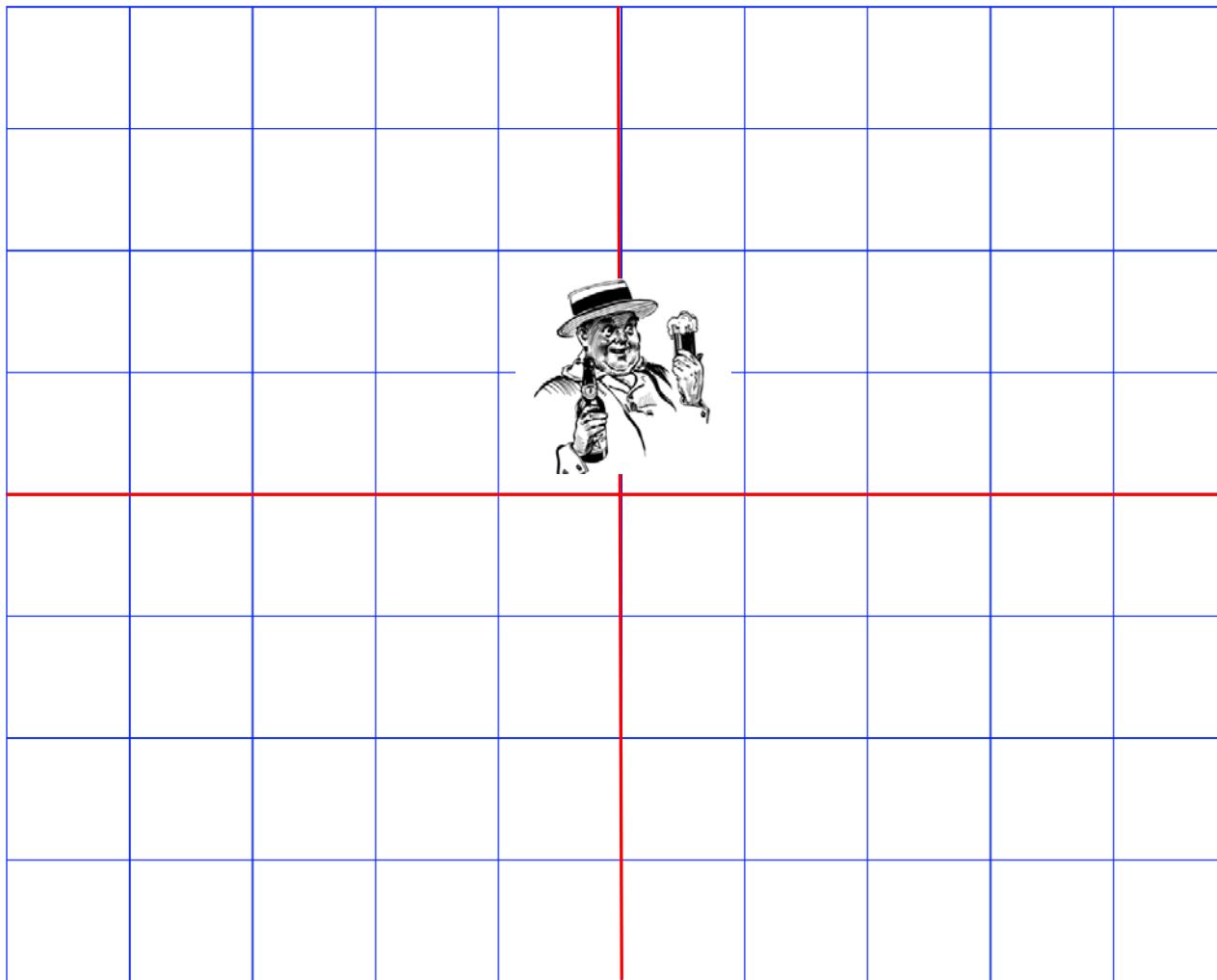
One Possible First Step



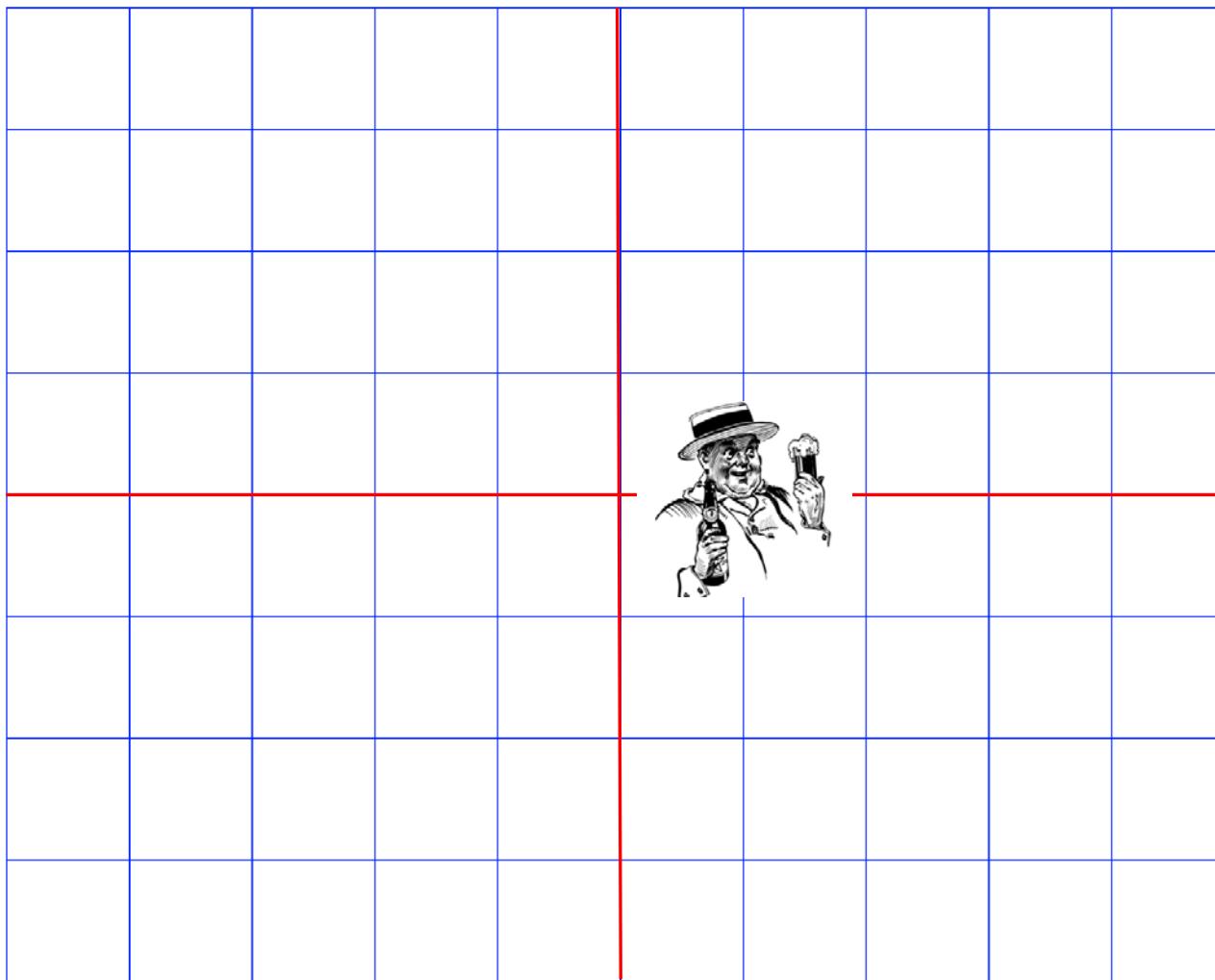
Another Possible First Step



Yet Another Possible First Step



Last Possible First Step



Possible Distances After Two Steps

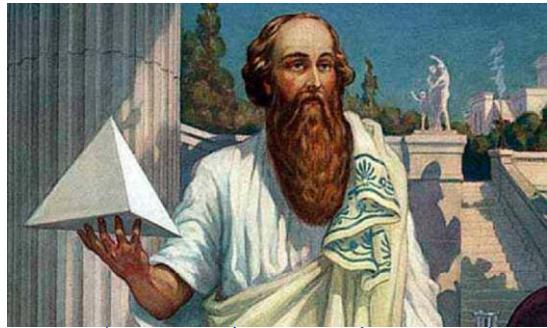
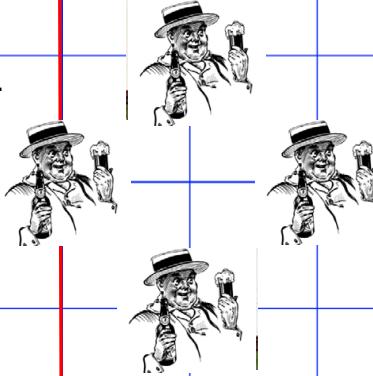


Image of Pythagoras © unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.



Expected Distance After 100,000 Steps?

- Need a different approach to problem
- Will use simulation
- But not until the next lecture

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

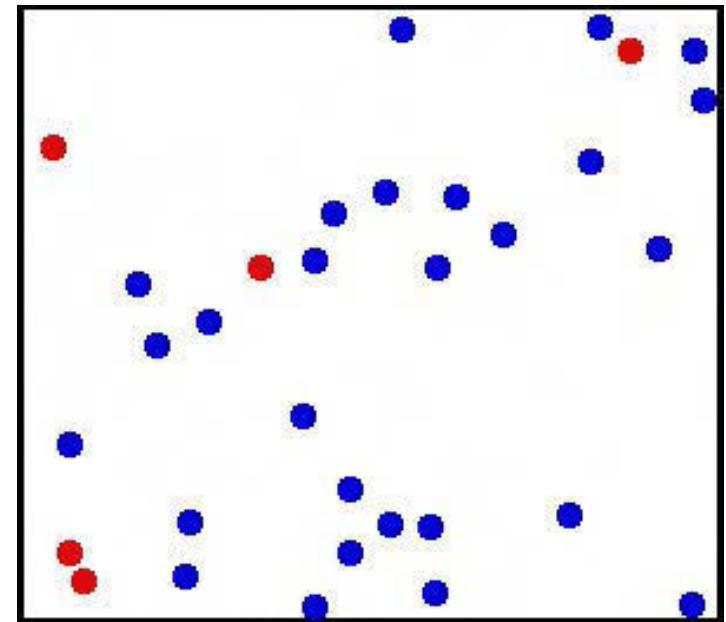
Lecture 5: Random Walks

Relevant Reading

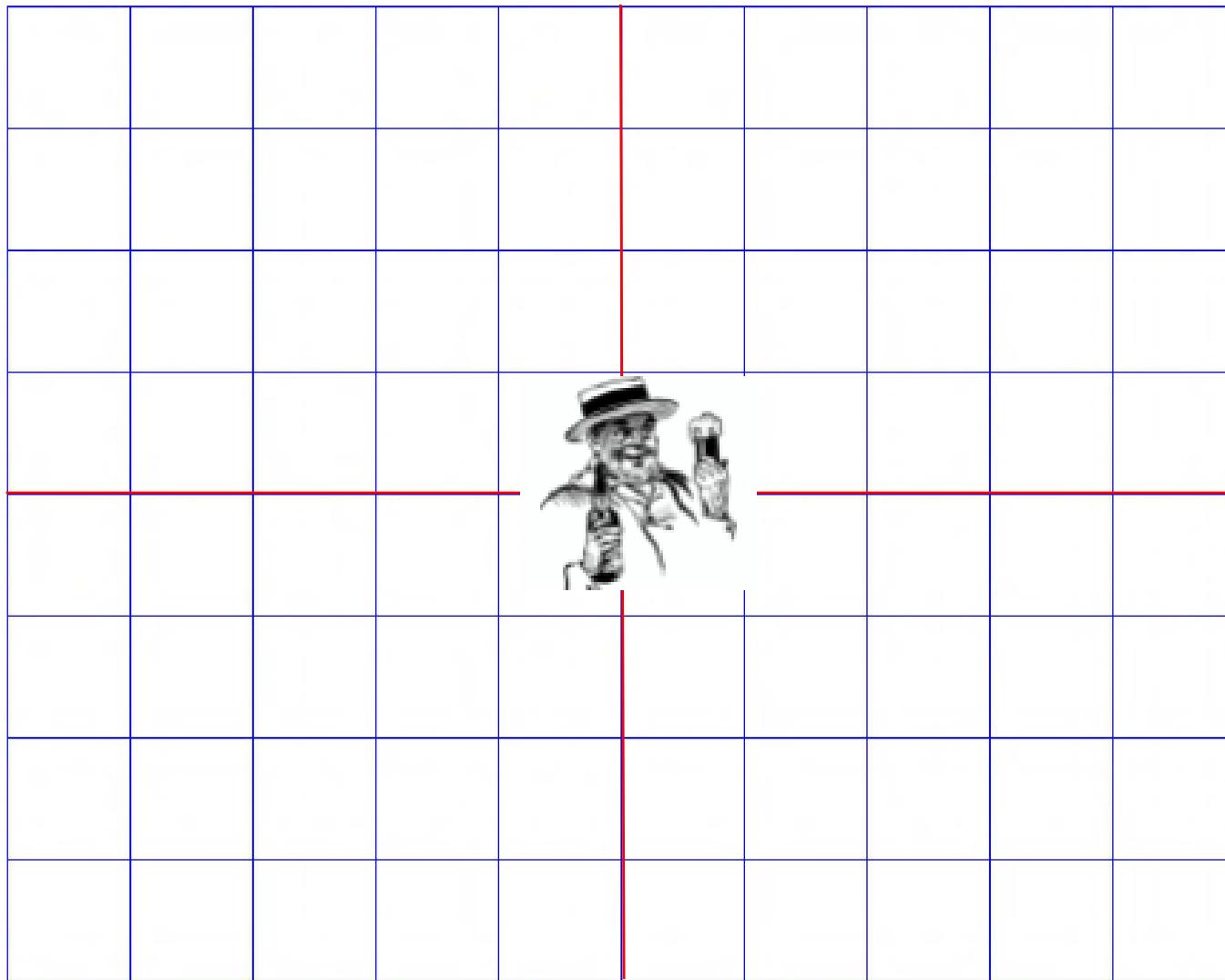
- Chapter 11
- Chapter 14

Why Random Walks?

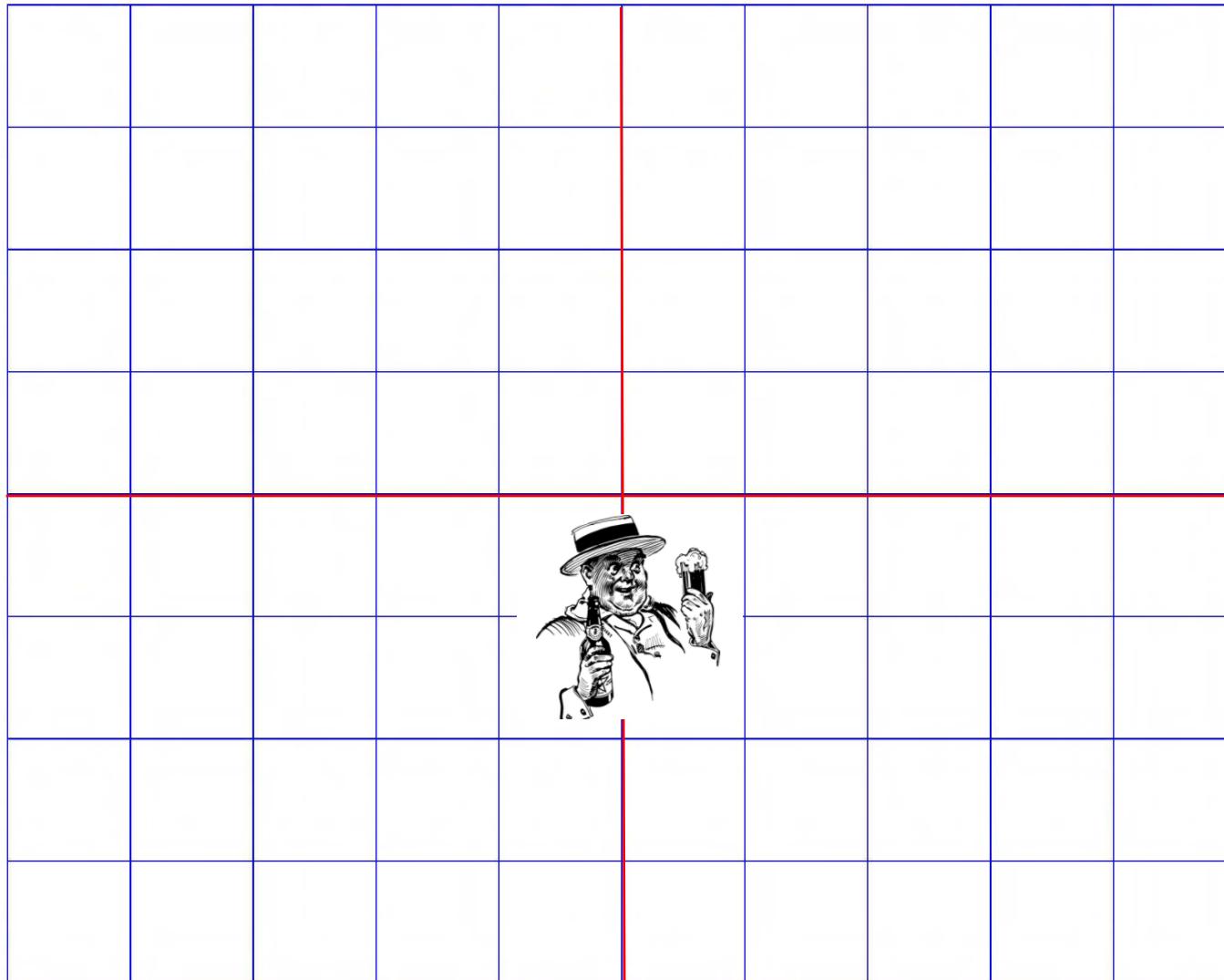
- Random walks are important in many domains
 - Understanding the stock market (maybe)
 - Modeling diffusion processes
 - Etc.
- Good illustration of how to use simulations to understand things
- Excuse to cover some important programming topics
 - Practice with classes
 - Practice with plotting



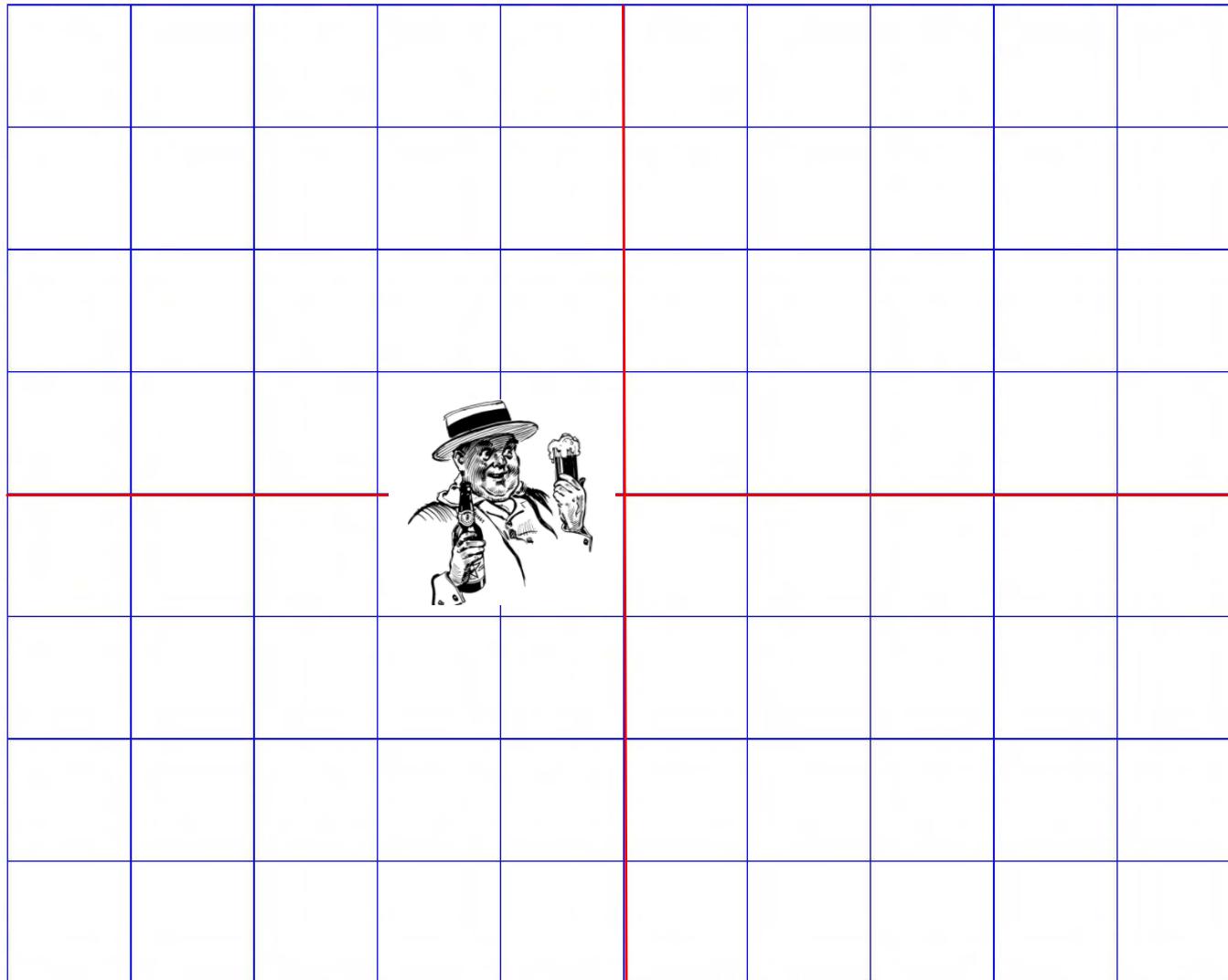
Drunkard's Walk



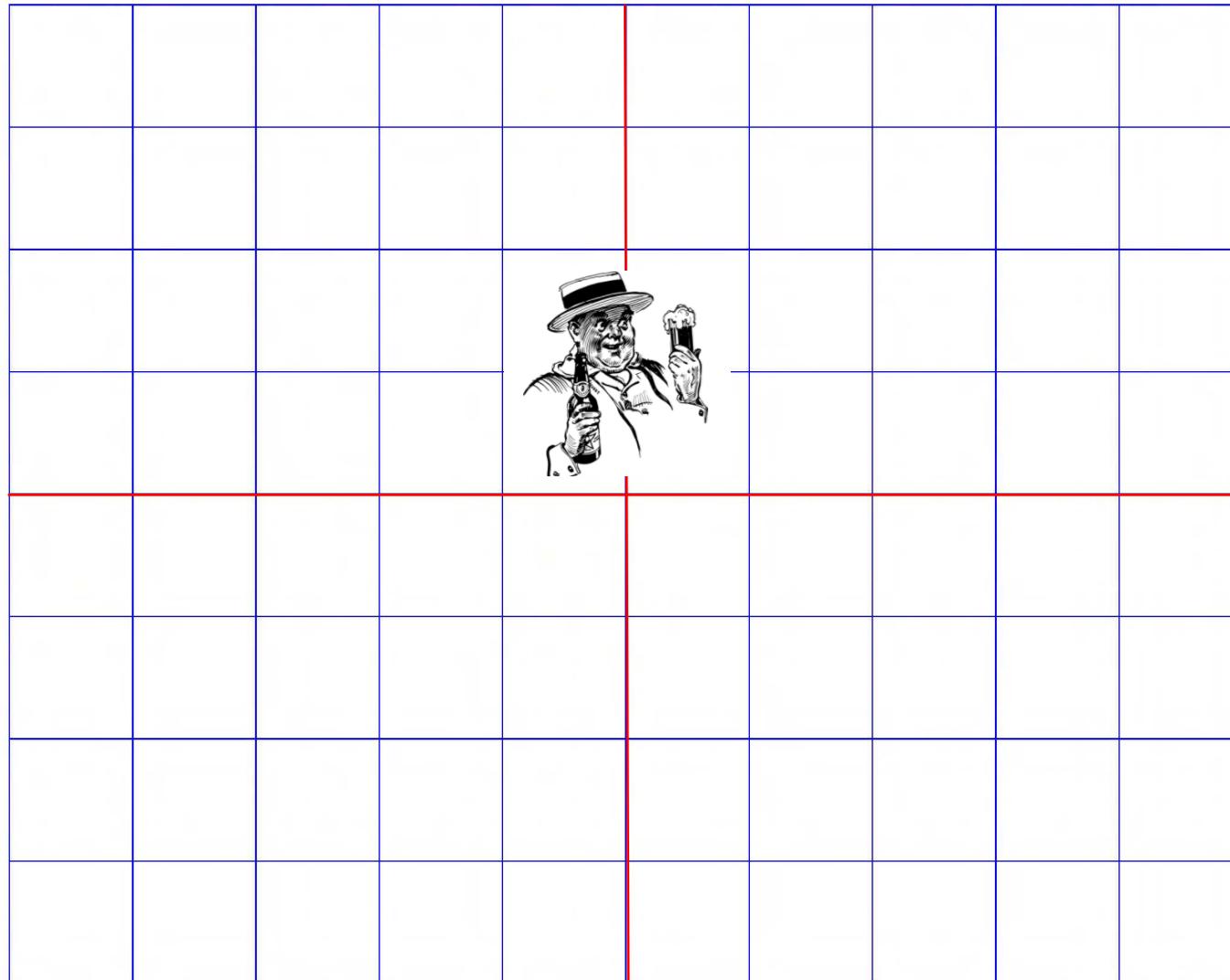
One Possible First Step



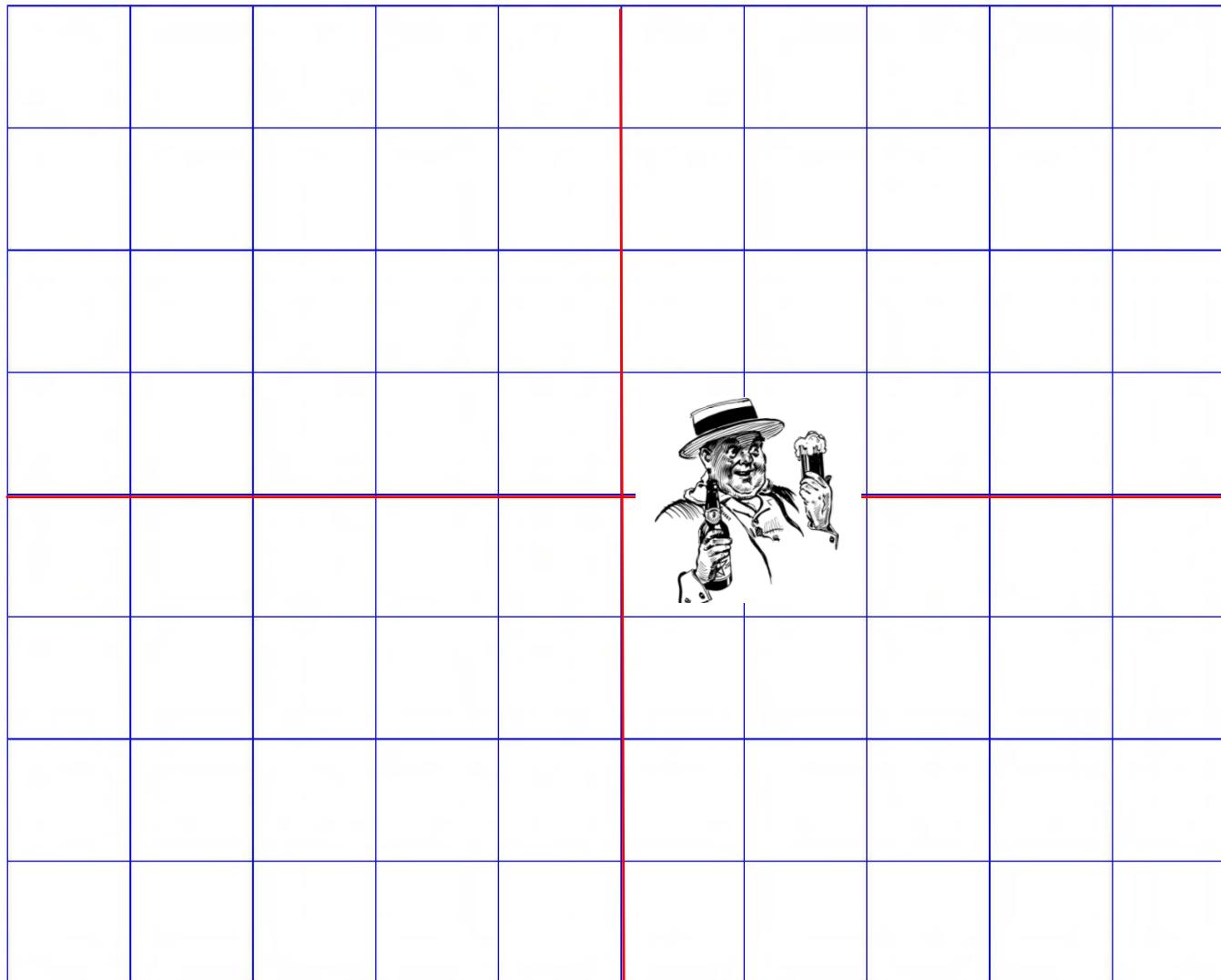
Another Possible First Step



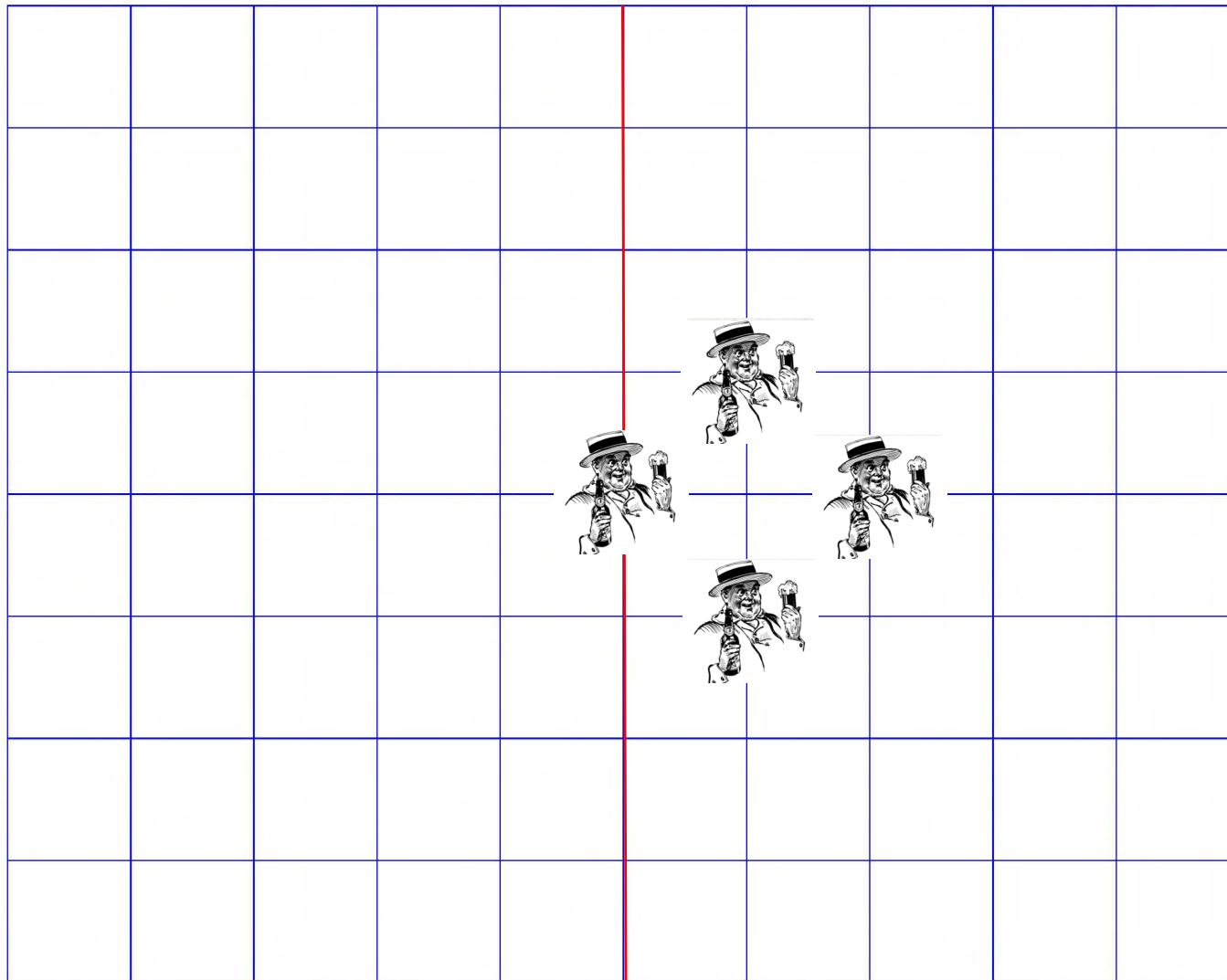
Yet Another Possible First Step



Last Possible First Step



Possible Distances After Two Steps



Expected Distance After 100,000 Steps?

- Need a different approach to problem
- Will use simulation

Structure of Simulation

- Simulate one walks of k steps
- Simulate n such walks
- Report average distance from origin

First, Some Useful Abstractions

- Location—a place
- Field—a collection of places and drunks
- Drunk—somebody who wanders from place to place in a field

Class Location, part 1

```
class Location(object):           Immutable type
    def __init__(self, x, y):
        """x and y are floats"""
        self.x = x
        self.y = y

    def move(self, deltaX, deltaY):
        """deltaX and deltaY are floats"""
        return Location(self.x + deltaX,
                        self.y + deltaY)

    def getX(self):
        return self.x

    def getY(self):
        return self.y
```

Class Location, continued

```
def distFrom(self, other):
    xDist = self.x - other.getX()
    yDist = self.y - other.getY()
    return (xDist**2 + yDist**2)**0.5

def __str__(self):
    return '<' + str(self.x) + ', ' +
           + str(self.y) + '>'
```

Class Drunk

```
class Drunk(object):
    def __init__(self, name = None):
        """Assumes name is a str"""
        self.name = name

    def __str__(self):
        if self != None:
            return self.name
        return 'Anonymous'
```

Not intended to be useful on its own

A base class to be inherited

Two Subclasses of Drunk

- The “usual” drunk, who wanders around at random
- The “masochistic” drunk, who tries to move northward

Two Kinds of Drunks

```
import random

class UsualDrunk(Drunk):
    def takeStep(self):
        stepChoices = [(0,1), (0,-1), (1, 0), (-1, 0)]
        return random.choice(stepChoices)

class MasochistDrunk(Drunk):
    def takeStep(self):
        stepChoices = [(0.0,1.1), (0.0,-0.9),
                      (1.0, 0.0), (-1.0, 0.0)]
        return random.choice(stepChoices)
```

Immutable or not?

Class Field, part 1

```
class Field(object):
    def __init__(self):
        self.drunks = {}

    def addDrunk(self, drunk, loc):
        if drunk in self.drunks:
            raise ValueError('Duplicate drunk')
        else:
            self.drunks[drunk] = loc

    def getLoc(self, drunk):
        if drunk not in self.drunks:
            raise ValueError('Drunk not in field')
        return self.drunks[drunk]
```

Class Field, continued

```
def moveDrunk(self, drunk):
    if drunk not in self.drunks:
        raise ValueError('Drunk not in field')
    xDist, yDist = drunk.takeStep()
    #use move method of Location to get new location
    self.drunks[drunk] =\
        self.drunks[drunk].move(xDist, yDist)
```

Immutable or not?

Simulating a Single Walk

```
def walk(f, d, numSteps):
    """Assumes: f a Field, d a Drunk in f, and
    numSteps an int >= 0.
    Moves d numSteps times; returns the distance
    between the final location and the location
    at the start of the walk."""
    start = f.getLoc(d)
    for s in range(numSteps):
        f.moveDrunk(d)
    return start.distFrom(f.getLoc(d))
```

Simulating Multiple Walks

```
def simWalks(numSteps, numTrials, dClass):
    """Assumes numSteps an int >= 0, numTrials an
       int > 0, dClass a subclass of Drunk
       Simulates numTrials walks of numSteps steps
       each. Returns a list of the final distances
       for each trial"""
    Homer = dClass()
    origin = Location(0, 0)
    distances = []
    for t in range(numTrials):
        f = Field()
        f.addDrunk(Homer, origin)
        distances.append(round(walk(f, Homer,
                                     numTrials), 1))
    return distances
```

Putting It All Together

```
def drunkTest(walkLengths, numTrials, dClass):
    """Assumes walkLengths a sequence of ints >= 0
       numTrials an int > 0,
       dClass a subclass of Drunk
       For each number of steps in walkLengths,
       runs simWalks with numTrials walks and
       prints results"""

    for numSteps in walkLengths:
        distances = simWalks(numSteps, numTrials,
                             dClass)
        print(dClass.__name__, 'random walk of',
              numSteps, 'steps')
        print(' Mean =',
              round(sum(distances)/len(distances), 4))
        print(' Max =', max(distances),
              'Min =', min(distances))
```

Let's Try It

```
drunkTest((10, 100, 1000, 10000), 100,  
          UsualDrunk)
```

UsualDrunk random walk of 10 steps

Mean = 8.634

Max = 21.6 Min = 1.4

UsualDrunk random walk of 100 steps

Mean = 8.57

Max = 22.0 Min = 0.0

UsualDrunk random walk of 1000 steps

Mean = 9.206

Max = 21.6 Min = 1.4

UsualDrunk random walk of 10000 steps

Mean = 8.727

Max = 23.5 Min = 1.4

Plausible?

Let's Try a Sanity Check

- Try on cases where we think we know the answer
 - A very important precaution!

Sanity Check

```
drunkTest((0, 1, 2) 100, UsualDrunk)
```

UsualDrunk random walk of 0 steps

Mean = 8.634

Max = 21.6 Min = 1.4

UsualDrunk random walk of 1 steps

Mean = 8.57

Max = 22.0 Min = 0.0

UsualDrunk random walk of 2 steps

Mean = 9.206

Max = 21.6 Min = 1.4

```
distances.append(round(walk(f, Homer,  
numTrials), 1))
```

Let's Try It

```
drunkTest((10, 100, 1000, 10000), 100,  
          UsualDrunk)
```

UsualDrunk random walk of 10 steps

Mean = 2.863

Max = 7.2 Min = 0.0

UsualDrunk random walk of 100 steps

Mean = 8.296

Max = 21.6 Min = 1.4

UsualDrunk random walk of 1000 steps

Mean = 27.297

Max = 66.3 Min = 4.2

UsualDrunk random walk of 10000 steps

Mean = 89.241

Max = 226.5 Min = 10.0

And the Masochistic Drunk?

```
random.seed(0)
simAll((UsualDrunk, MasochistDrunk),
       (1000, 10000), 100)
```

UsualDrunk random walk of 1000 steps

Mean = 26.828

Max = 66.3 Min = 4.2

UsualDrunk random walk of 10000 steps

Mean = 90.073

Max = 210.6 Min = 7.2

MasochistDrunk random walk of 1000 steps

Mean = 58.425

Max = 133.3 Min = 6.7

MasochistDrunk random walk of 10000 steps

Mean = 515.575

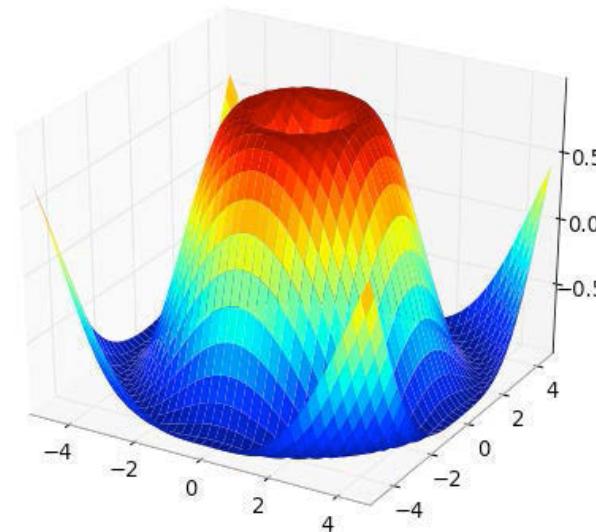
Max = 694.6 Min = 377.7

Visualizing the Trend

- Simulate walks of multiple lengths for each kind of drunk
- Plot distance at end of each length walk for each kind of drunk

Pylab

- **NumPy** adds vectors, matrices, and many high-level mathematical functions
- **SciPy** adds mathematical classes and functions useful to scientists
- **Matplotlib** adds an object-oriented API for plotting
- **PyLab** combines the other libraries to provide a MATLAB[®] like interface



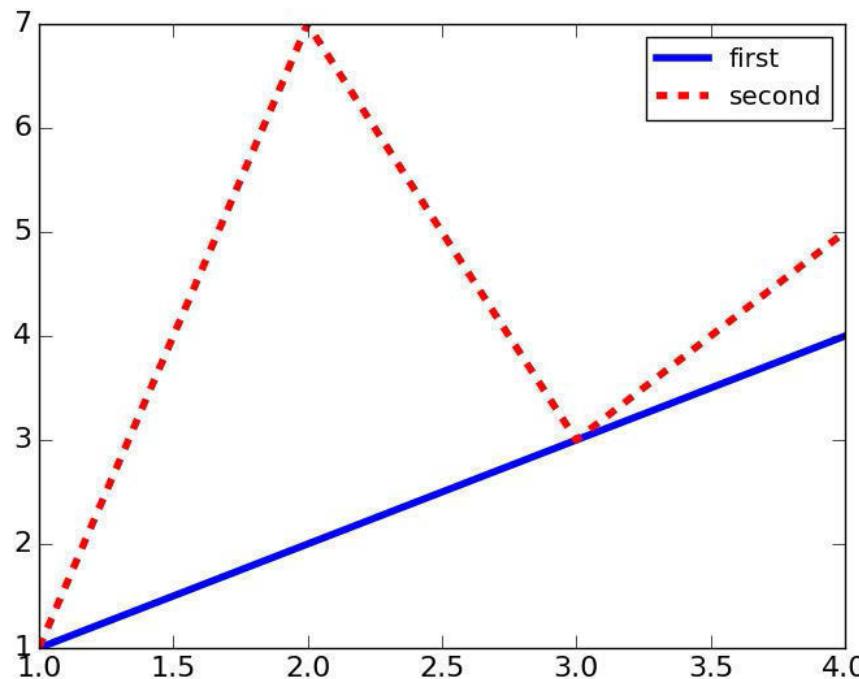
plot

- The first two arguments to `pylab.plot` must be sequences of the same length.
- First argument gives x-coordinates.
- Second argument gives y-coordinates.
- Many optional arguments
- Points plotted in order. In default style, as each point is plotted, a line is drawn connecting it to the previous point.

Example

```
import pylab

xVals = [1, 2, 3, 4]
yVals1 = [1, 2, 3, 4]
pylab.plot(xVals, yVals1, 'b-', label = 'first')
yVals2 = [1, 7, 3, 5]
pylab.plot(xVals, yVals2, 'r--', label = 'second')
pylab.legend()
```

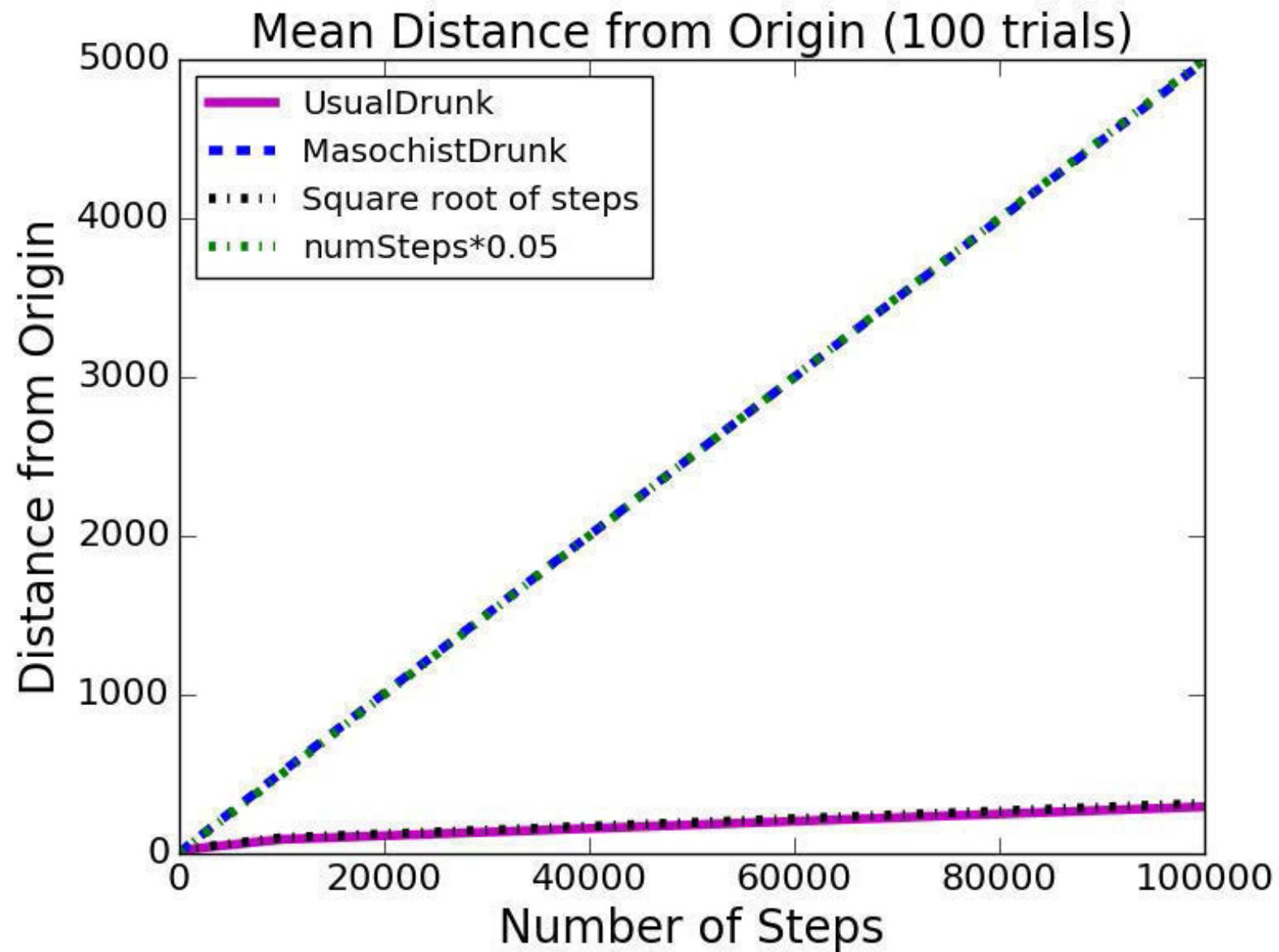


Details and Many More Examples

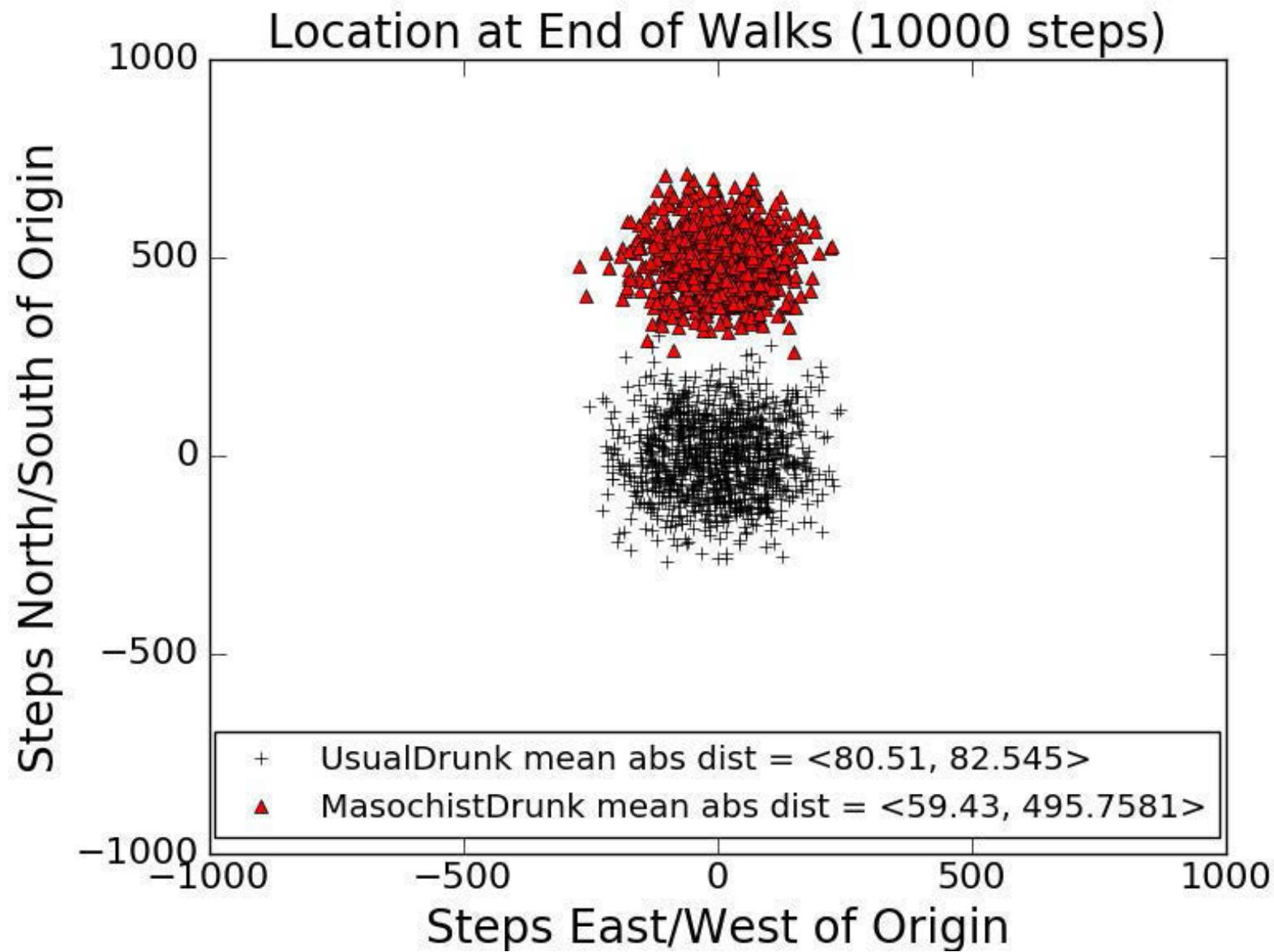
- Assigned reading
- Video of Prof. Grimson's lecture from 6.00x.1
- Code for this lecture
- matplotlib.org/api/pyplot_summary.html
- www.scipy.org/Plotting_Tutorial

You should learn how to produce
the plots that I will show you

Distance Trends



Ending Locations



Fields with Wormholes

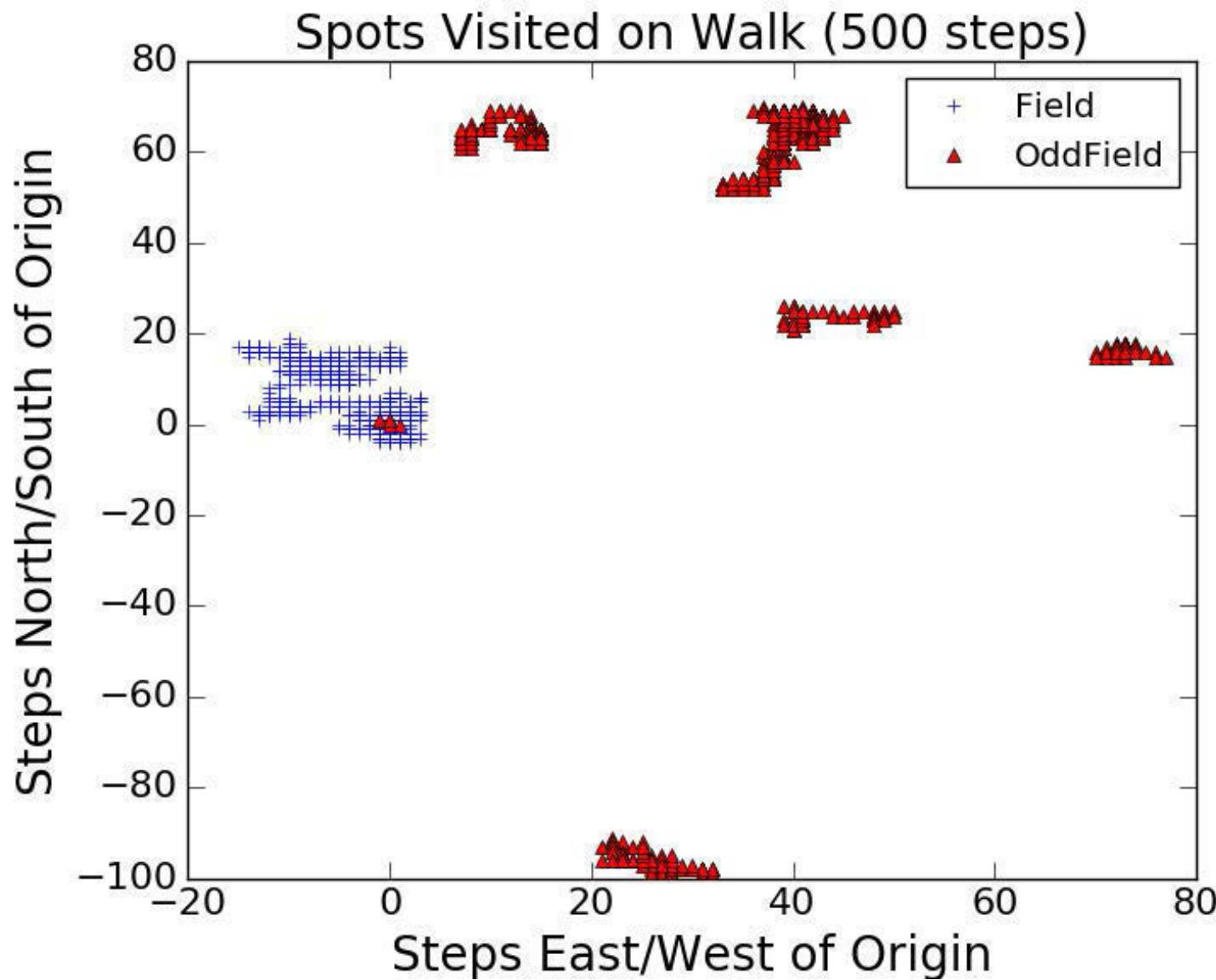
A Subclass of Field, part 1

```
class OddField(Field):
    def __init__(self, numHoles = 1000,
                 xRange = 100, yRange = 100):
        Field.__init__(self)
        self.wormholes = {}
        for w in range(numHoles):
            x = random.randint(-xRange, xRange)
            y = random.randint(-yRange, yRange)
            newX = random.randint(-xRange, xRange)
            newY = random.randint(-yRange, yRange)
            newLoc = Location(newX, newY)
            self.wormholes[(x, y)] = newLoc
```

A Subclass of Field, part 2

```
def moveDrunk(self, drunk):
    Field.moveDrunk(self, drunk)
    x = self.drunks[drunk].getX()
    y = self.drunks[drunk].getY()
    if (x, y) in self.wormholes:
        self.drunks[drunk] = self.wormholes[(x, y)]
```

Spots Reached During One Walk



Summary

- Point is not the simulations themselves, but how we built them
- Started by defining classes
- Built functions corresponding to
 - One trial, multiple trials, result reporting
- Made series of incremental changes to simulation so that we could investigate different questions
 - Get simple version working first
 - Did a sanity check!
 - Elaborate a step at a time
- Showed how to use plots to get insights

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Lecture 6: Monte Carlo Simulation

Relevant Reading

- Sections 15-1 – 15.4
- Chapter 16

A Little History

- Ulam, recovering from an illness, was playing a lot of solitaire
- Tried to figure out probability of winning, and failed
- Thought about playing lots of hands and counting number of wins, but decided it would take years
- Asked Von Neumann if he could build a program to simulate many hands on ENIAC

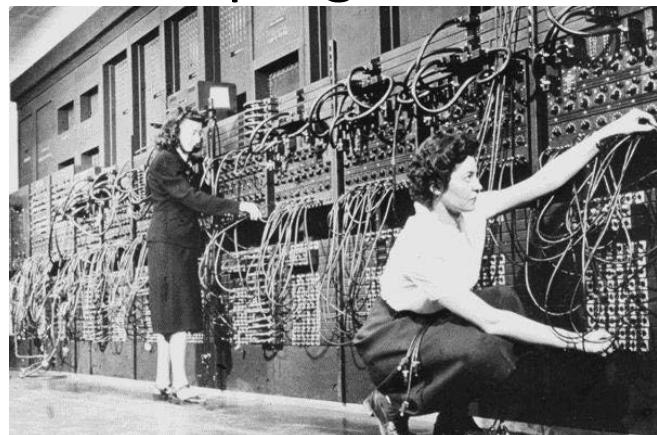


Image of ENIAC programmers © unknown. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.

Monte Carlo Simulation

- A method of estimating the value of an unknown quantity using the principles of inferential statistics
- Inferential statistics
 - *Population*: a set of examples
 - *Sample*: a proper subset of a population
 - Key fact: a *random sample* tends to exhibit the same properties as the population from which it is drawn
- Exactly what we did with random walks

An Example

- Given a single coin, estimate fraction of heads you would get if you flipped the coin an infinite number of times
- Consider one flip



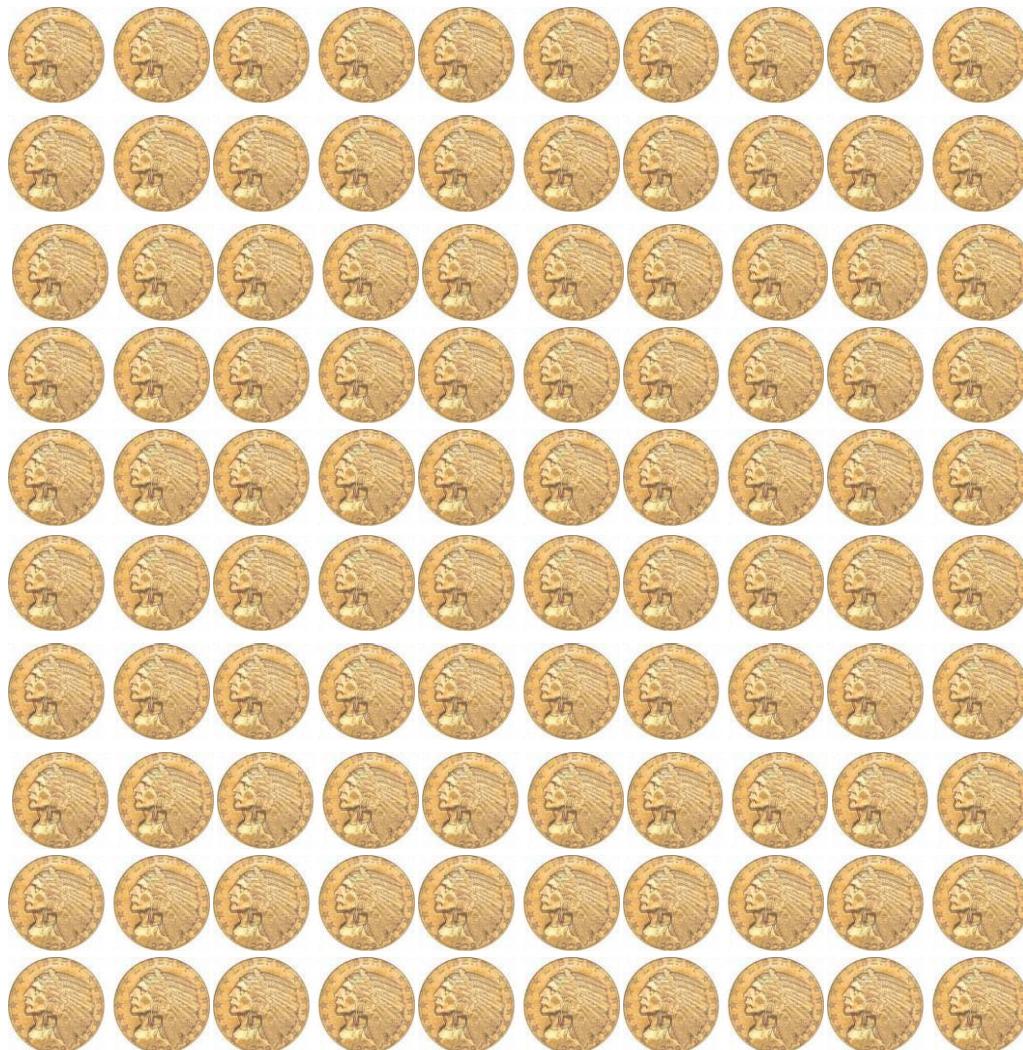
How confident would you be about answering 1.0?

Flipping a Coin Twice



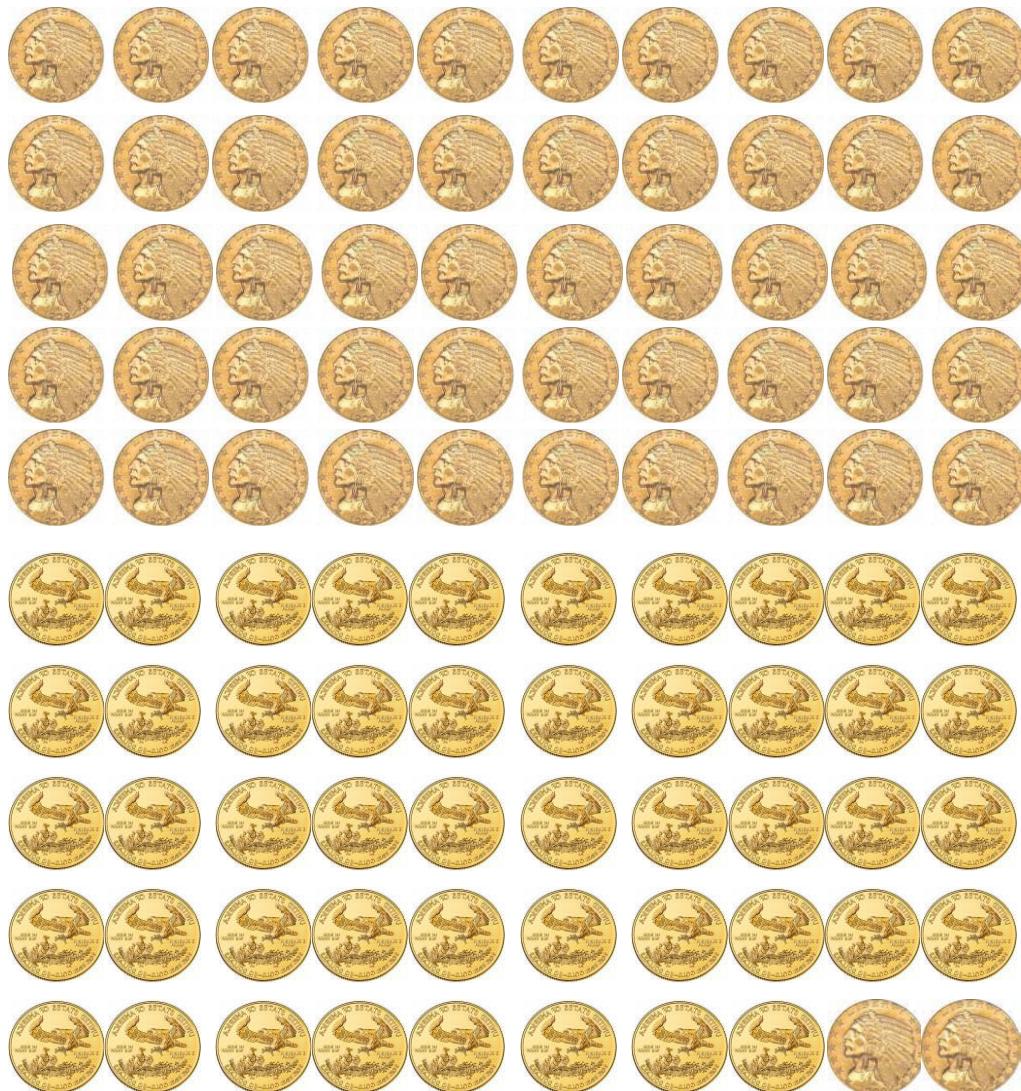
Do you think that the next flip will come up heads?

Flipping a Coin 100 Times



Now do you
think that the
next flip will
come up heads?

Flipping a Coin 100 Times



Do you think
that the
probability of
the next flip
coming up
heads is $52/100$?

Given the data,
it's your best
estimate

But confidence
should be low

Why the Difference in Confidence?

- Confidence in our estimate depends upon two things
- Size of sample (e.g., 100 versus 2)
- Variance of sample (e.g., all heads versus 52 heads)
- As the variance grows, we need larger samples to have the same degree of confidence

Roulette



Image of roulette wheel © unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.

No need to simulate, since answers obvious

Allows us to compare simulation results to actual probabilities

Class Definition

```
class FairRoulette():
    def __init__(self):
        self.pockets = []
        for i in range(1,37):
            self.pockets.append(i)
        self.ball = None
        self.pocketOdds = len(self.pockets) - 1
    def spin(self):
        self.ball = random.choice(self.pockets)
    def betPocket(self, pocket, amt):
        if str(pocket) == str(self.ball):
            return amt*self.pocketOdds
        else: return -amt
    def __str__(self):
        return 'Fair Roulette'
```

Monte Carlo Simulation

```
def playRoulette(game, numSpins, pocket, bet):
    totPocket = 0
    for i in range(numSpins):
        game.spin()
        totPocket += game.betPocket(pocket, bet)
    if toPrint:
        print(numSpins, 'spins of', game)
        print('Expected return betting', pocket, '=', \
              str(100*totPocket/numSpins) + '%\n')
    return (totPocket/numSpins)

game = FairRoulette()
for numSpins in (100, 1000000):
    for i in range(3):
        playRoulette(game, numSpins, 2, 1, True)
```

100 and 1M Spins of the Wheel

100 spins of Fair Roulette

Expected return betting 2 = -100.0%

100 spins of Fair Roulette

Expected return betting 2 = 44.0%

100 spins of Fair Roulette

Expected return betting 2 = -28.0%

1000000 spins of Fair Roulette

Expected return betting 2 = -0.046%

1000000 spins of Fair Roulette

Expected return betting 2 = 0.602%

1000000 spins of Fair Roulette

Expected return betting 2 = 0.7964%

Law of Large Numbers

- In repeated independent tests with the same actual probability p of a particular outcome in each test, the chance that the fraction of times that outcome occurs differs from p converges to zero as the number of trials goes to infinity

Does this imply that if deviations from expected behavior occur, these deviations are likely to be ***evened out*** by opposite deviations in the future?

Gambler's Fallacy

- “On August 18, 1913, at the casino in Monte Carlo, black came up a record twenty-six times in succession [in roulette]. ... [There] was a near-panicky rush to bet on red, beginning about the time black had come up a phenomenal fifteen times.” -- Huff and Geis, *How to Take a Chance*
- Probability of 26 consecutive reds
 - $1/67,108,865$
- Probability of 26 consecutive reds when previous 25 rolls were red
 - $1/2$

Regression to the Mean

- Following an extreme random event, the next random event is likely to be less extreme
- If you spin a fair roulette wheel 10 times and get 100% reds, that is an extreme event (probability = 1/1024)
- It is likely that in the next 10 spins, you will get fewer than 10 reds
 - But the expected number is only 5
- So, if you look at the average of the 20 spins, it will be closer to the expected mean of 50% reds than to the 100% of the first 10 spins

Casinos Not in the Business of Being Fair



Two Subclasses of Roulette

```
class EuRoulette(FairRoulette):
    def __init__(self):
        FairRoulette.__init__(self)
        self.pockets.append('0')
    def __str__(self):
        return 'European Roulette'

class AmRoulette(EuRoulette):
    def __init__(self):
        EuRoulette.__init__(self)
        self.pockets.append('00')
    def __str__(self):
        return 'American Roulette'
```

Comparing the Games

Simulate 20 trials of 1000 spins each

Exp. return for Fair Roulette = 6.56%

Exp. return for European Roulette = -2.26%

Exp. return for American Roulette = -8.92%

Simulate 20 trials of 10000 spins each

Exp. return for Fair Roulette = -1.234%

Exp. return for European Roulette = -4.168%

Exp. return for American Roulette = -5.752%

Simulate 20 trials of 100000 spins each

Exp. return for Fair Roulette = 0.8144%

Exp. return for European Roulette = -2.6506%

Exp. return for American Roulette = -5.113%

Simulate 20 trials of 1000000 spins each

Exp. return for Fair Roulette = -0.0723%

Exp. return for European Roulette = -2.7329%

Exp. return for American Roulette = -5.212%

Sampling Space of Possible Outcomes

- Never possible to guarantee perfect accuracy through sampling
- Not to say that an estimate is not precisely correct
- Key question:
 - How many samples do we need to look at before we can have justified confidence on our answer?
- Depends upon variability in underlying distribution

Quantifying Variation in Data

$$\text{variance}(X) = \frac{\sum_{x \in X} (x - \mu)^2}{|X|}$$

$$\sigma(X) = \sqrt{\frac{1}{|X|} \sum_{x \in X} (x - \mu)^2}$$

- Standard deviation simply the square root of the variance
- Outliers can have a big effect
- Standard deviation should always be considered relative to mean

For Those Who Prefer Code

```
def getMeanAndStd(X):
    mean = sum(X)/float(len(X))
    tot = 0.0
    for x in X:
        tot += (x - mean)**2
    std = (tot/len(X))**0.5
    return mean, std
```

Confidence Levels and Intervals

- Instead of estimating an unknown parameter by a single value (e.g., the mean of a set of trials), a confidence interval provides a range that is likely to contain the unknown value and a confidence that the unknown value lays within that range
- “The return on betting a pocket 10k times in European roulette is -3.3%. The margin of error is +/- 3.5% with a 95% level of confidence.”
- What does this mean?
- If I were to conduct an infinite number of trials of 10k bets each,
 - My expected average return would be -3.3%
 - My return would be between roughly -6.8% and +0.2% 95% of the time

Empirical Rule

- Under some assumptions discussed later
 - ~68% of data within one standard deviation of mean
 - ~95% of data within 1.96 standard deviations of mean
 - ~99.7% of data within 3 standard deviations of mean

Applying Empirical Rule

```
resultDict = {}
games = (FairRoulette, EuRoulette, AmRoulette)
for G in games:
    resultDict[G().__str__()] = []
for numSpins in (100, 1000, 10000):
    print('\nSimulate betting a pocket for', numTrials,
          'trials of', numSpins, 'spins each')
    for G in games:
        pocketReturns = findPocketReturn(G(), 20,
                                         numSpins, False)
        mean, std = getMeanAndStd(pocketReturns)
        resultDict[G().__str__()].append((numSpins,
                                         100*mean,
                                         100*std))
    print('Exp. return for', G(), '=',
          str(round(100*mean, 3))
          + '%, ', '+/- ' + str(round(100*1.96*std, 3))
          + '% with 95% confidence')
```

Results

Simulate betting a pocket for 20 trials of 1000 spins each

Exp. return for Fair Roulette = 3.68%, +/- 27.189% with 95% confidence

Exp. return for European Roulette = -5.5%, +/- 35.042% with 95% confidence

Exp. return for American Roulette = -4.24%, +/- 26.494% with 95% confidence

Simulate betting a pocket for 20 trials of 100000 spins each

Exp. return for Fair Roulette = 0.125%, +/- 3.999% with 95% confidence

Exp. return for European Roulette = -3.313%, +/- 3.515% with 95% confidence

Exp. return for American Roulette = -5.594%, +/- 4.287% with 95% confidence

Simulate betting a pocket for 20 trials of 1000000 spins each

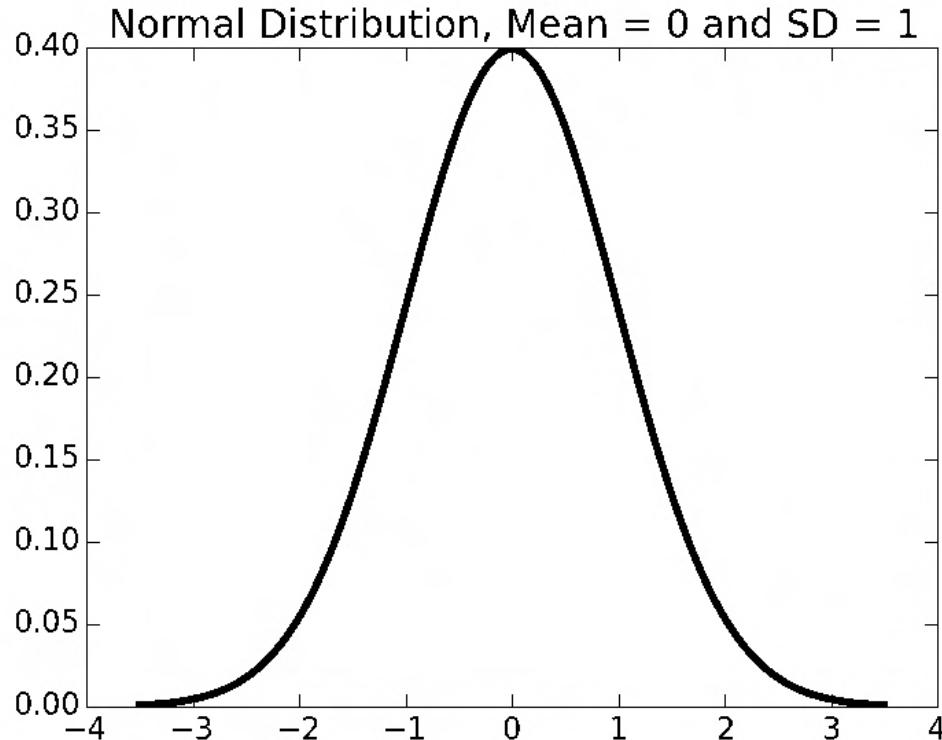
Exp. return for Fair Roulette = 0.012%, +/- 0.846% with 95% confidence

Exp. return for European Roulette = -2.679%, +/- 0.948% with 95% confidence

Exp. return for American Roulette = -5.176%, +/- 1.214% with 95% confidence

Assumptions Underlying Empirical Rule

- The mean estimation error is zero
- The distribution of the errors in the estimates is normal



Defining Distributions

- Use a probability distribution
- Captures notion of relative frequency with which a random variable takes on certain values
 - Discrete random variables drawn from finite set of values
 - Continuous random variables drawn from reals between two numbers (i.e., infinite set of values)
- For discrete variable, simply list the probability of each value, must add up to 1
- Continuous case trickier, can't enumerate probability for each of an infinite set of values

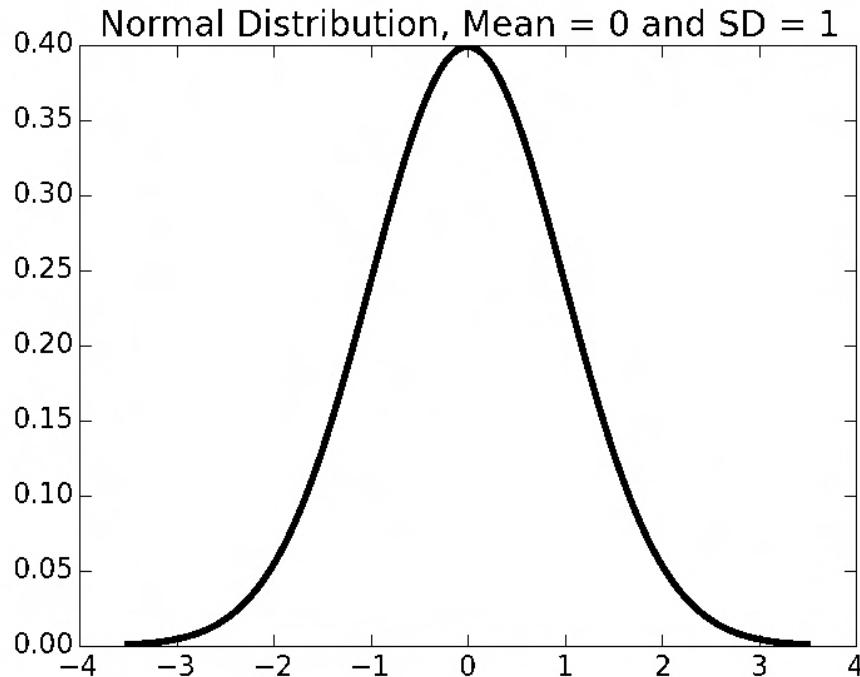
PDF's

- Distributions defined by *probability density functions* (PDFs)
- Probability of a random variable lying between two values
- Defines a curve where the values on the x-axis lie between minimum and maximum value of the variable
- Area under curve between two points, is probability of example falling within that range

Normal Distributions

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$



~68% of data within one standard deviation of mean

~95% of data within 1.96 standard deviations of mean

~99.7% of data within 3 standard deviations of mean

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

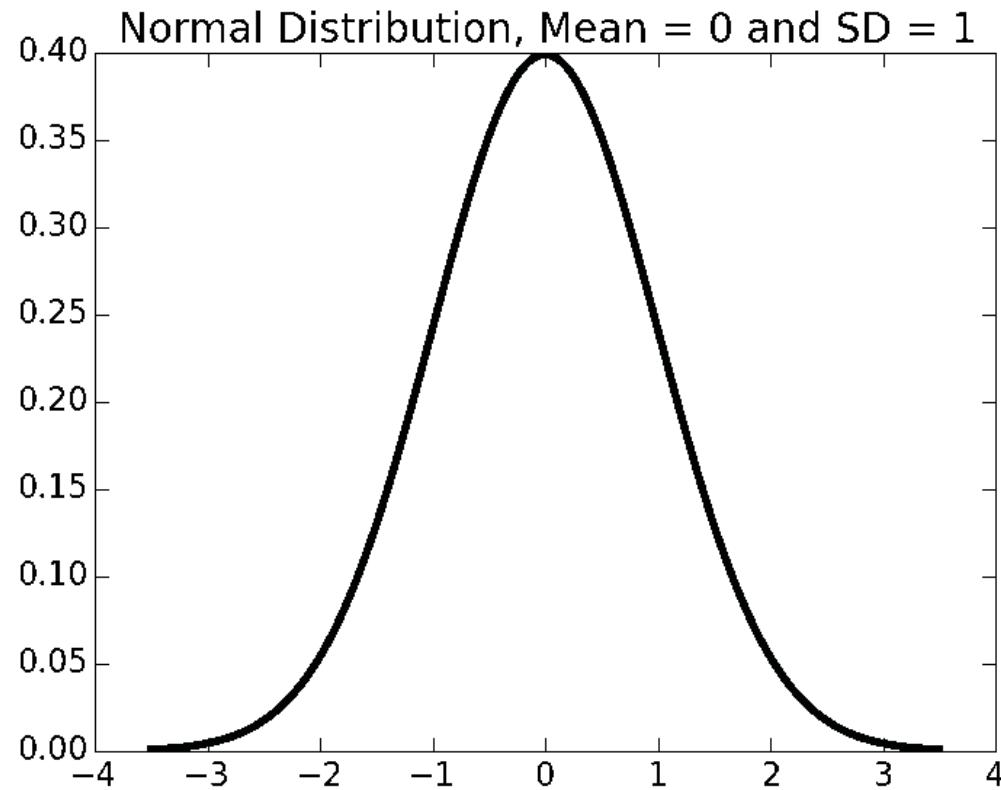
Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Lecture 7: Confidence Intervals

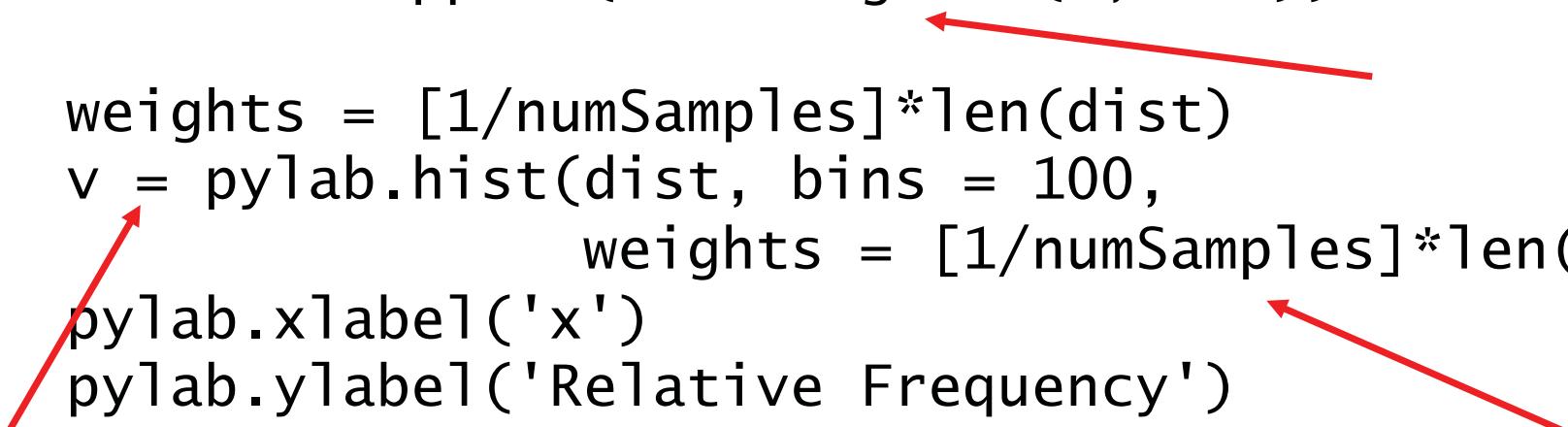
Assumptions Underlying Empirical Rule

- The mean estimation error is zero
- The distribution of the errors in the estimates is normal (Gaussian)



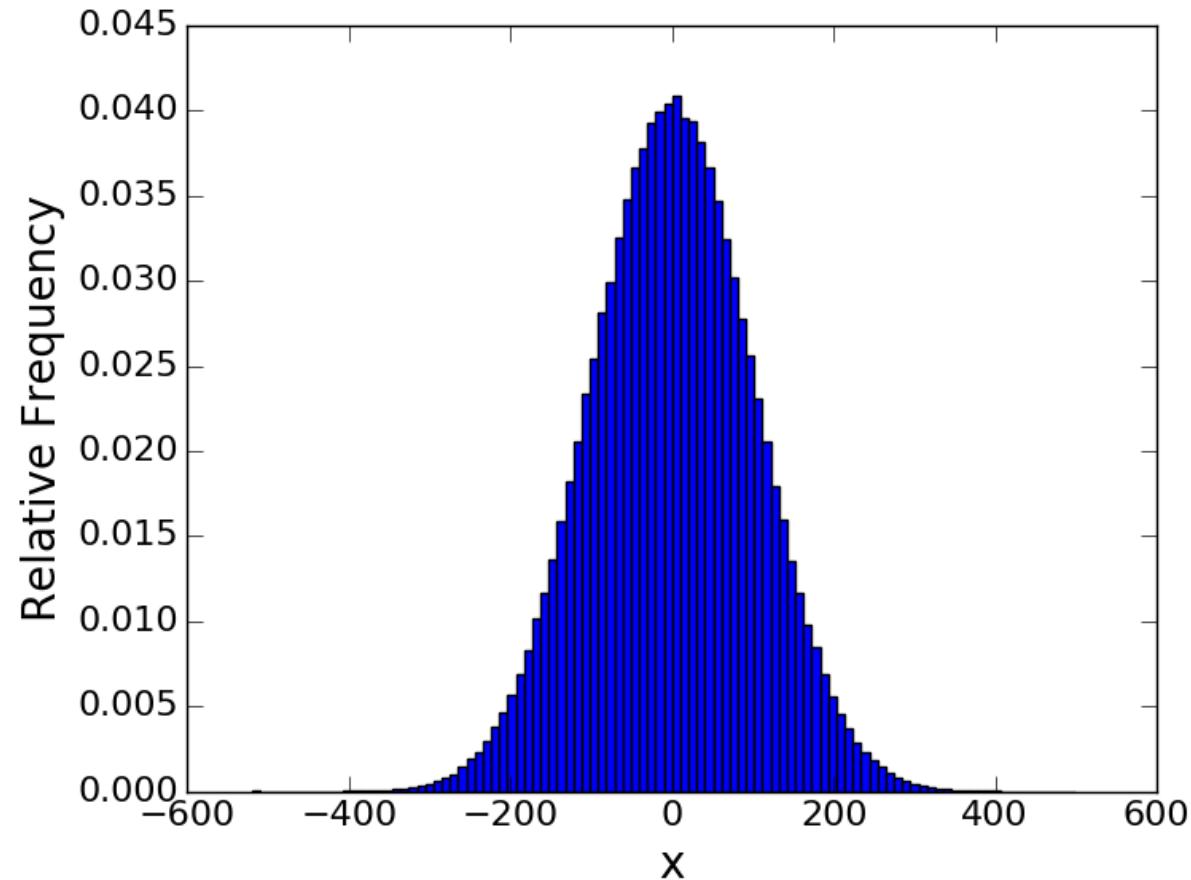
Generating Normally Distributed Data

```
dist, numSamples = [], 1000000  
  
for i in range(numSamples):  
    dist.append(random.gauss(0, 100))  
  
weights = [1/numSamples]*len(dist)  
v = pylab.hist(dist, bins = 100,  
               weights = [1/numSamples]*len(dist))  
pylab.xlabel('x')  
pylab.ylabel('Relative Frequency')  
  
print('Fraction within ~200 of mean =',  
      sum(v[0][30:70]))
```



Output

Discrete
Approximation
to PDF



Fraction within ~ 200 of mean = 0.957147

PDF's (recapping)

- Distributions defined by *probability density functions* (PDFs)
- Probability of a random variable lying between two values
- Defines a curve where the values on the x-axis lie between minimum and maximum value of the variable
- Area under curve between two points, is probability of example falling within that range

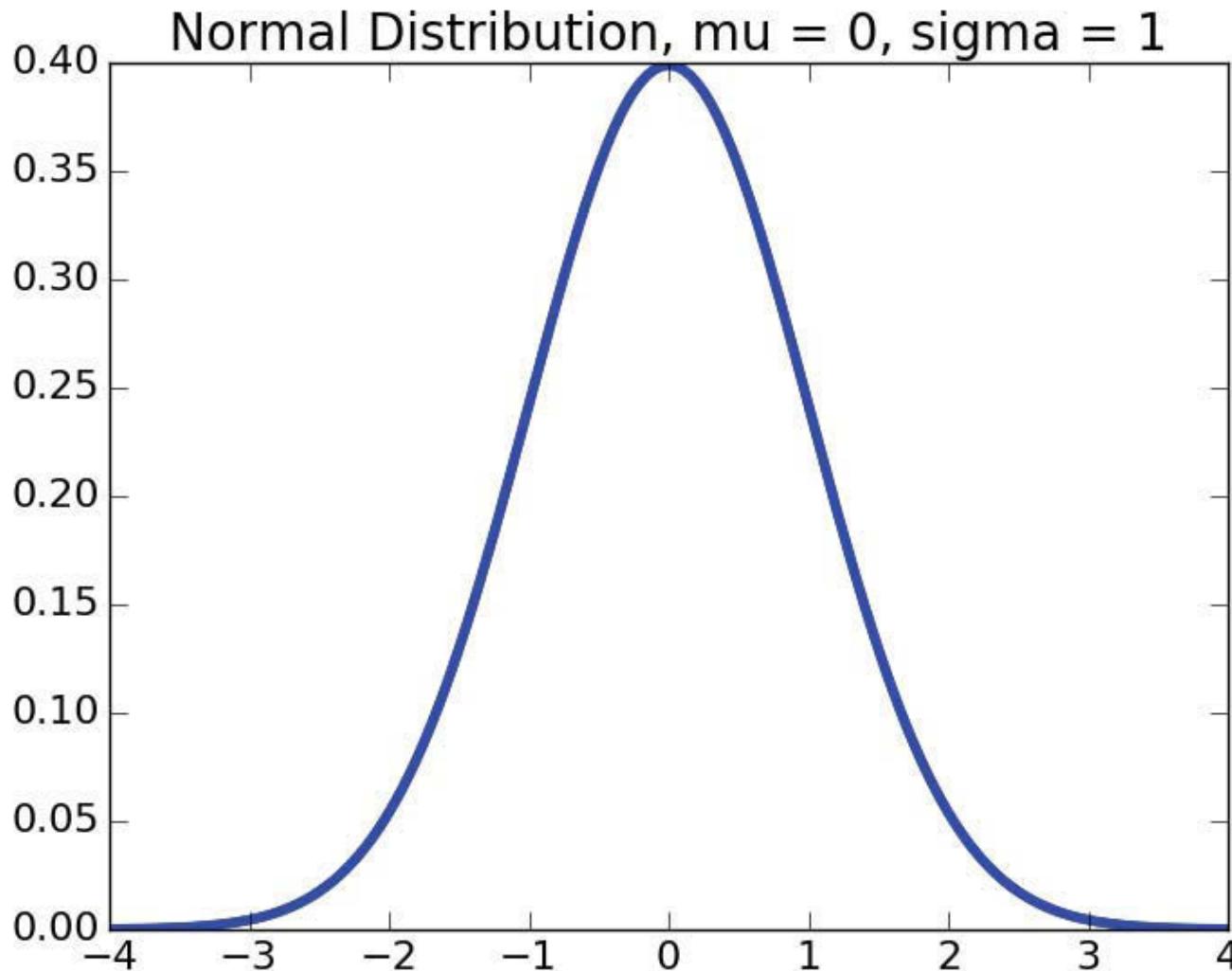
PDF for Normal Distribution

```
def gaussian(x, mu, sigma):
    factor1 = (1.0/(sigma*((2*pi)**0.5)))
    factor2 = pylab.e**-(((x-mu)**2)/(2*sigma**2))
    return factor1*factor2

xVals, yVals = [], []
mu, sigma = 0, 1
x = -4
while x <= 4:
    xVals.append(x)
    yVals.append(gaussian(x, mu, sigma))
    x += 0.05
pylab.plot(xVals, yVals)
pylab.title('Normal Distribution, mu = ' + str(mu) \
            + ', sigma = ' + str(sigma))
```

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Output



Are values on y-axis probabilities?

They are densities.
I.e., derivative of cumulative distribution function.

Hence we use integration to interpret a PDF

A Digression

- SciPy library contains many useful mathematical functions used by scientists and engineers
- `scipy.integrate.quad` has up to four arguments
 - a function or method to be integrated
 - a number representing the lower limit of the integration,
 - a number representing the upper limit of the integration, and
 - an optional tuple supplying values for all arguments, except the first, of the function to be integrated
- `scipy.integrate.quad` returns a tuple
 - Approximation to result
 - Estimate of absolute error

Checking the Empirical Rule

```
import scipy.integrate ←  
  
def gaussian(x, mu, sigma)  
    ...  
def checkEmpirical(numTrials):  
    for t in range(numTrials):  
        mu = random.randint(-10, 10)  
        sigma = random.randint(1, 10)  
        print('For mu =', mu, 'and sigma =', sigma)  
        for numStd in (1, 1.96, 3):  
            area = scipy.integrate.quad(gaussian,  
                                         mu-numStd*sigma,  
                                         mu+numStd*sigma,  
                                         (mu, sigma))[0]  
            print('Fraction within', numStd,  
                  'std =', round(area, 4))
```

Results

For $\mu = 9$ and $\sigma = 6$

Fraction within 1 std = 0.6827

Fraction within 1.96 std = 0.95

Fraction within 3 std = 0.9973

For $\mu = -6$ and $\sigma = 5$

Fraction within 1 std = 0.6827

Fraction within 1.96 std = 0.95

Fraction within 3 std = 0.9973

For $\mu = 2$ and $\sigma = 6$

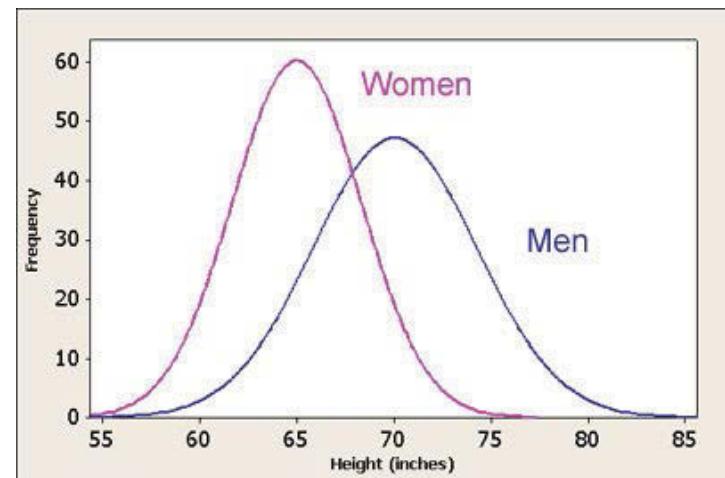
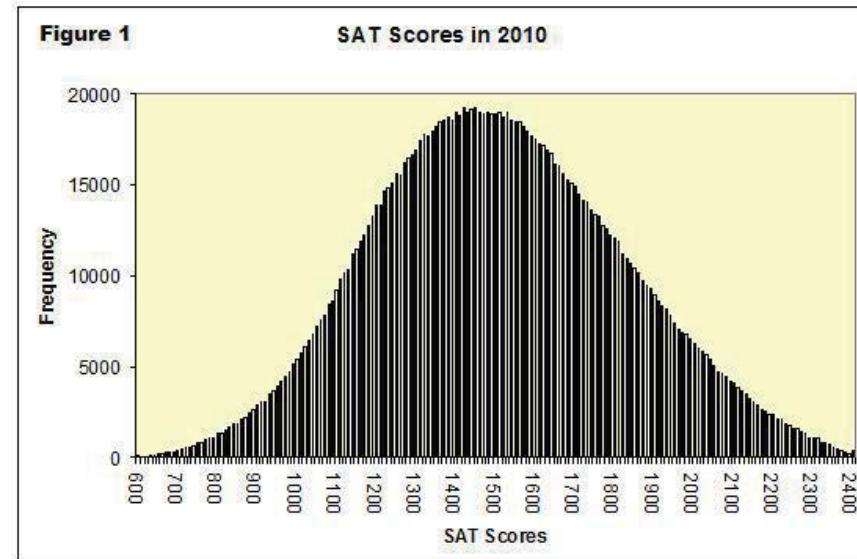
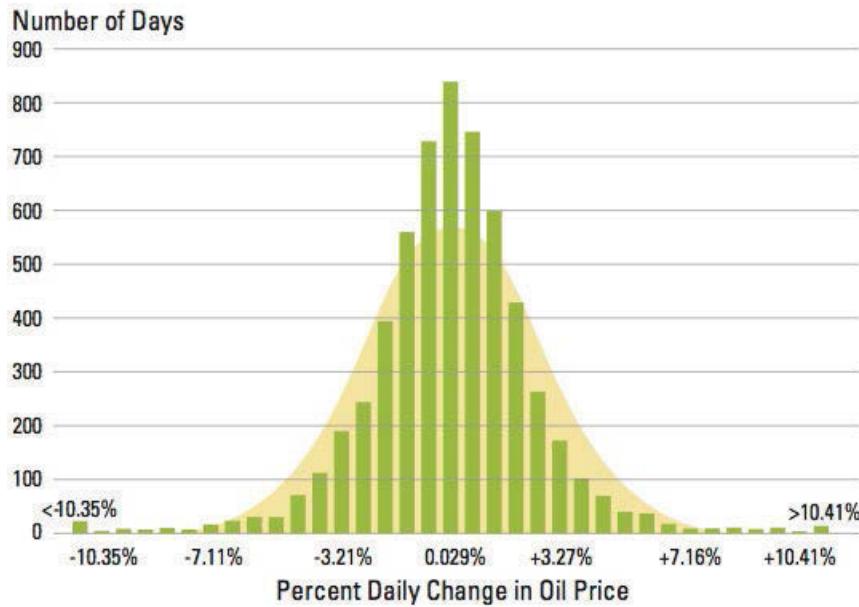
Fraction within 1 std = 0.6827

Fraction within 1.96 std = 0.95

Fraction within 3 std = 0.9973

Everybody Likes Normal Distributions

- Occur a lot!
- Nice mathematical properties

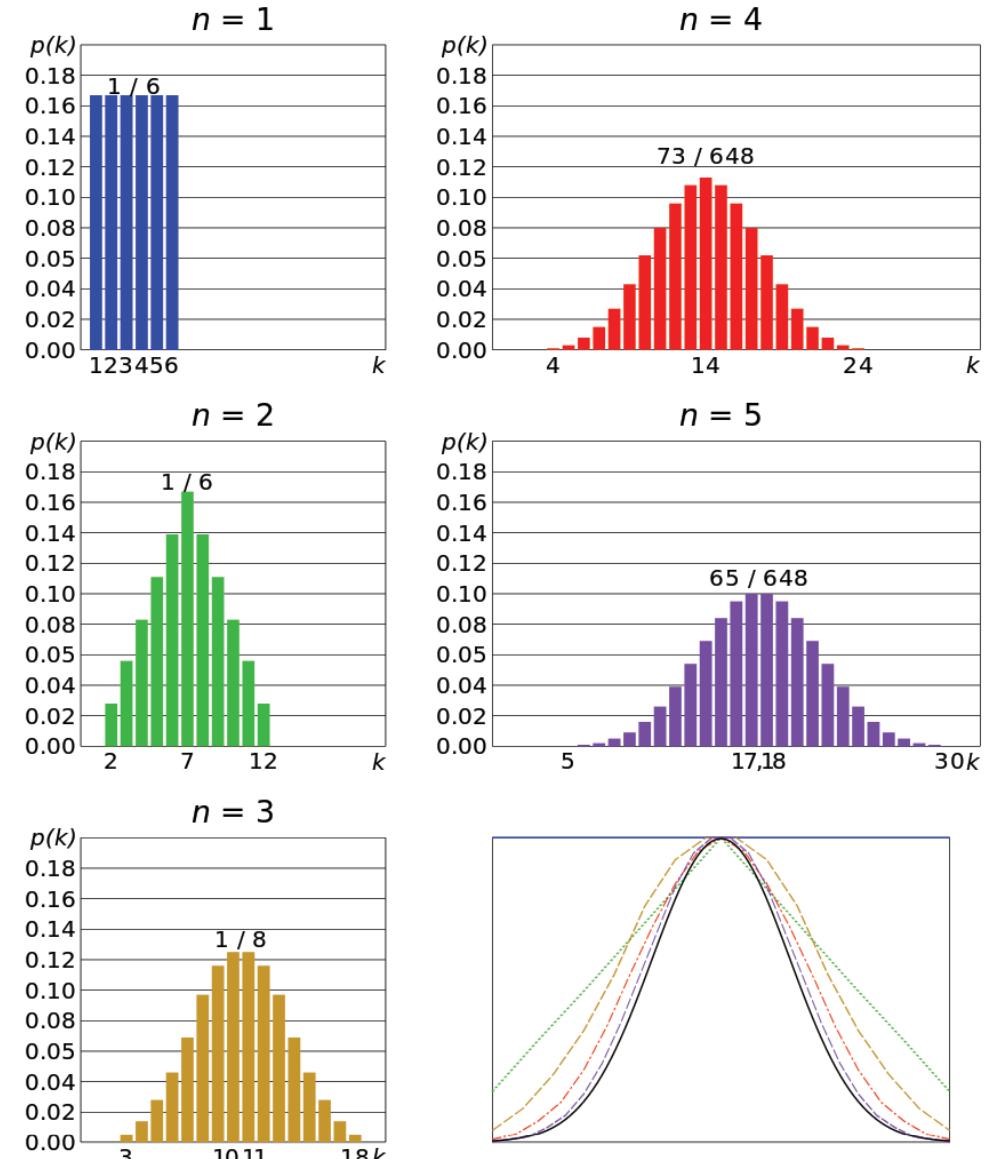


But Not All Distribution Are Normal

- Empirical works for normal distributions
- But are the outcomes of spins of a roulette wheel normally distributed?
- No, they are uniformly distributed
 - Each outcome is equally probable
- So, why does the empirical rule work here?

Why Did the Empirical Rule Work?

- Because we are reasoning not about a single spin, but about the mean of a set of spins
- And the **central limit theorem** applies



The Central Limit Theorem (CLT)

- Given a sufficiently large sample:
 - 1) The means of the samples in a set of samples (the sample means) will be approximately normally distributed,
 - 2) This normal distribution will have a mean close to the mean of the population, and
 - 3) The variance of the sample means will be close to the variance of the population divided by the sample size.

Checking CLT for a Continuous Die

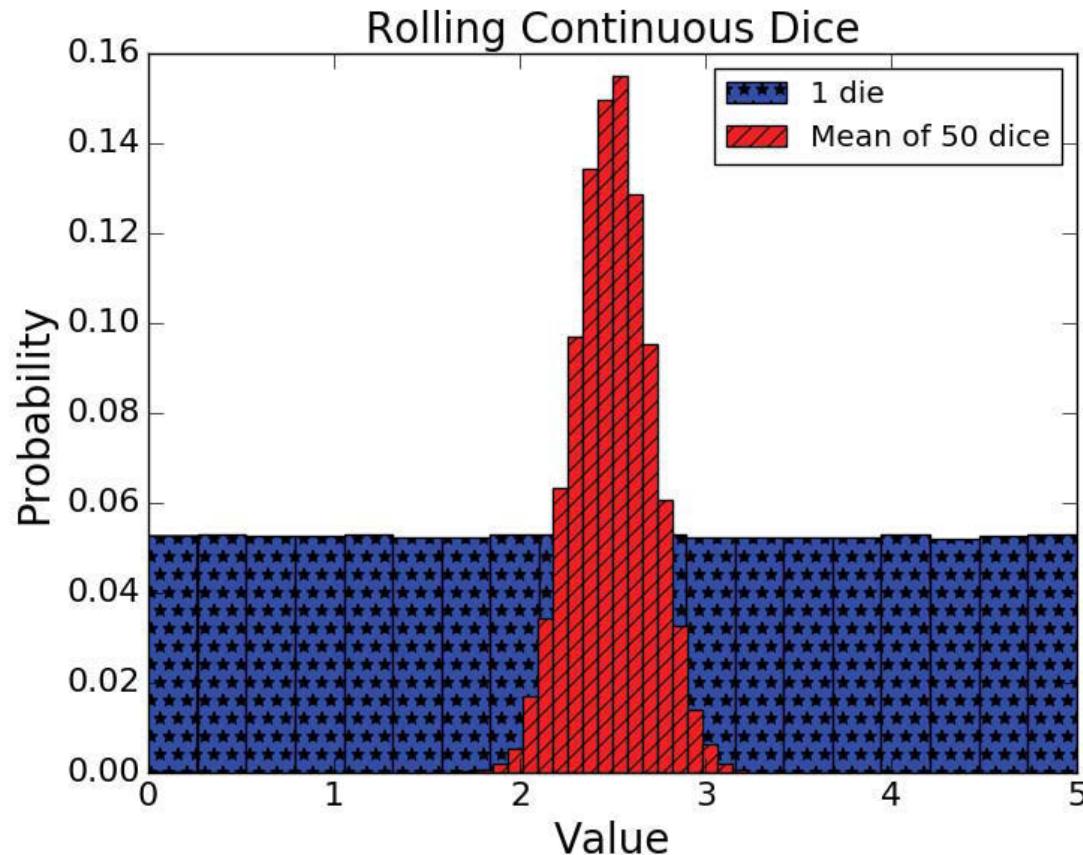
```
def plotMeans(numDice, numRolls, numBins, legend, color, style):
    means = []
    for i in range(numRolls//numDice):
        vals = 0
        for j in range(numDice):
            vals += 5*random.random()
        means.append(vals/float(numDice))
    pylab.hist(means, numBins, color = color, label = legend,
               → weights = pylab.array(len(means)*[1])/len(means),
               hatch = style)
    return getMeanAndStd(means)

mean, std = plotMeans(1, 1000000, 19, '1 die', 'b', '*')
print('Mean of rolling 1 die =', str(mean) + ',', 'Std =', std)
mean, std = plotMeans(50, 1000000, 19, 'Mean of 50 dice', 'r', '//')
print('Mean of rolling 50 dice =', str(mean) + ',', 'Std =', std)
pylab.title('Rolling Continuous Dice')
pylab.xlabel('Value')
pylab.ylabel('Probability')
pylab.legend()
```

Output

Mean of rolling 1 die = 2.49759575528, Std = 1.4439045633

Mean of rolling 50 dice = 2.49985051798, Std = 0.204887274645



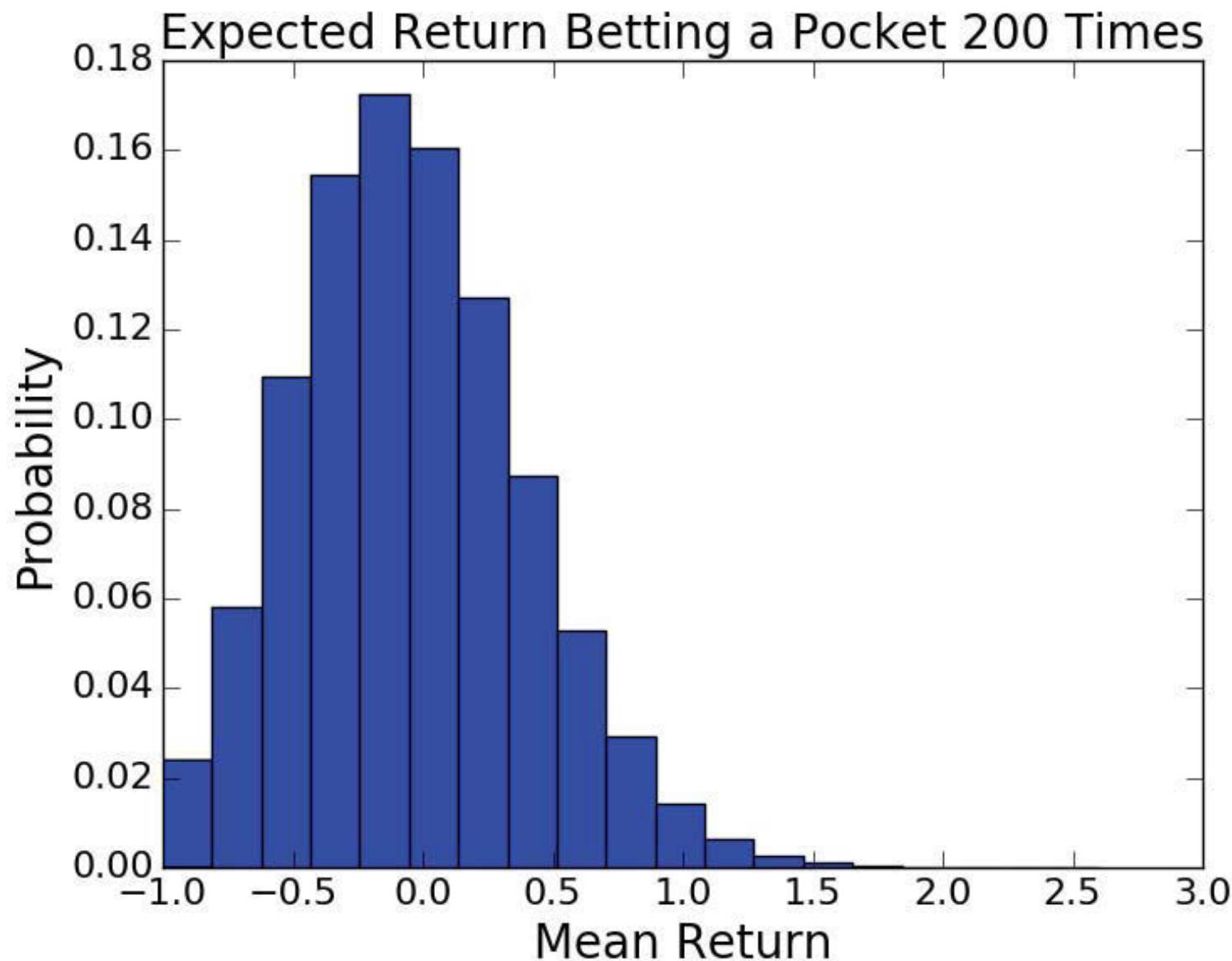
Try It for Roulette

```
numTrials = 1000000
numSpins = 200
game = FairRoulette()

means = []
for i in range(numTrials):
    means.append(findPocketReturn(game, 1, numSpins,
                                  False)[0])

pylab.hist(means, bins = 19,
           weights = [1/len(means)]*len(means))
pylab.xlabel('Mean Return')
pylab.ylabel('Probability')
pylab.title('Expected Return Betting a Pocket 200 Times')
```

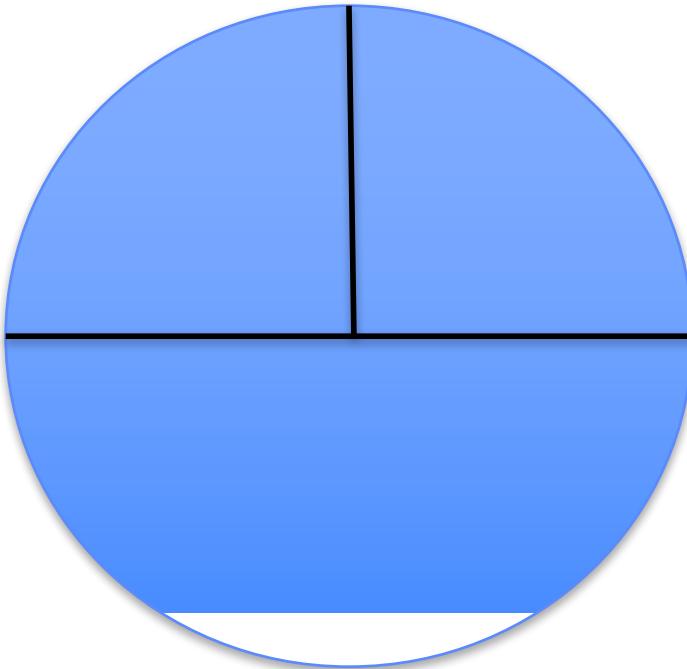
Betting a Pocket in Fair Roulette



Moral

- It doesn't matter what the shape of the distribution of values happens to be
- If we are trying to estimate the mean of a population using sufficiently large samples
- The CLT allows us to use the empirical rule when computing confidence intervals

Pi



$$\frac{\text{circumference}}{\text{diameter}} = \Pi \quad \text{area} = \Pi * \text{radius}^2$$

Rhind Papyrus



$$4 * (8/9)^2 = 3.16$$

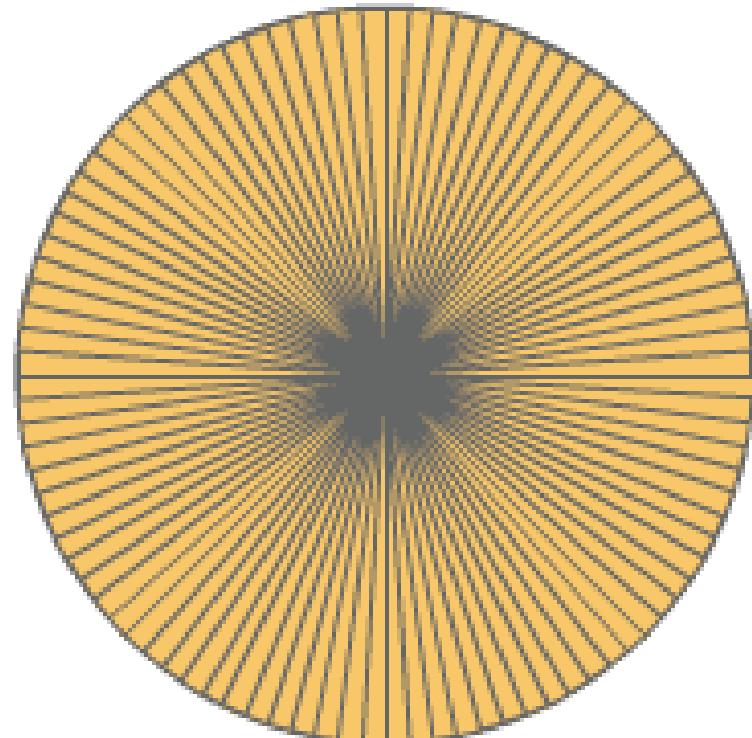
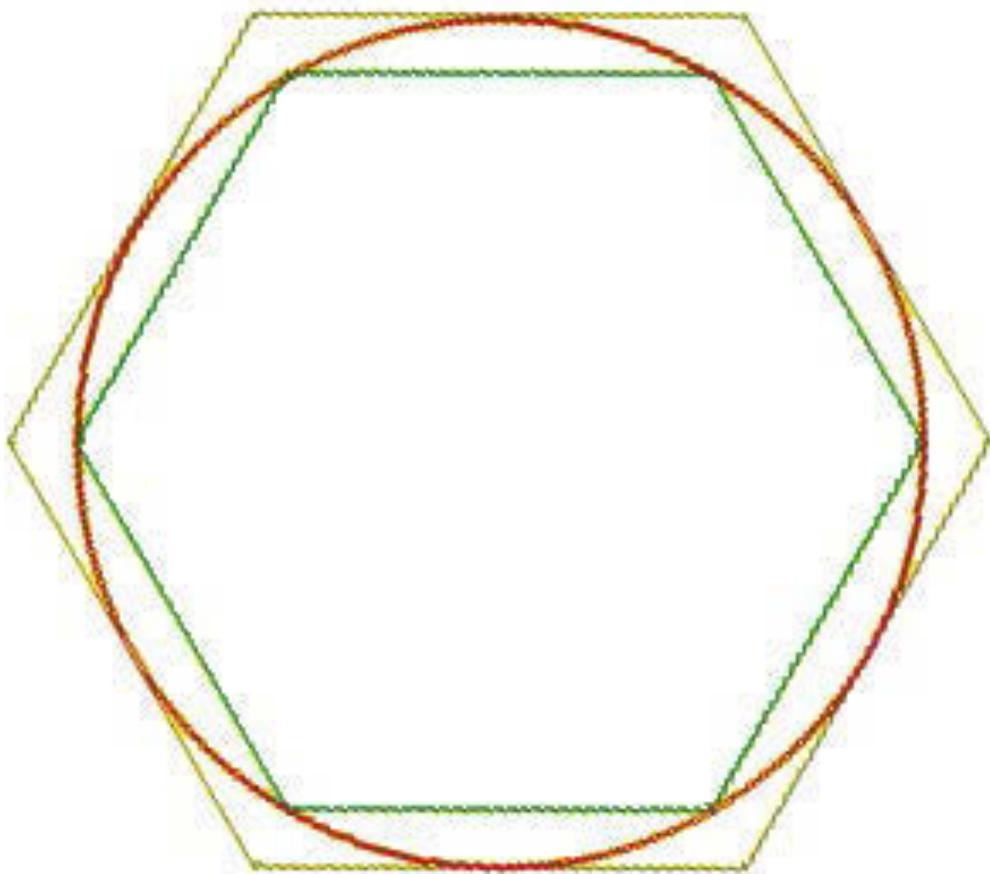
Image of the Rhind Papyrus is in the public domain. Source: [Wikimedia Commons](#).

~1100 Years Later

“And he made a molten sea, ten cubits from the one brim to the other: it was round all about, and his height was five cubits: and a line of thirty cubits did compass it round about.”

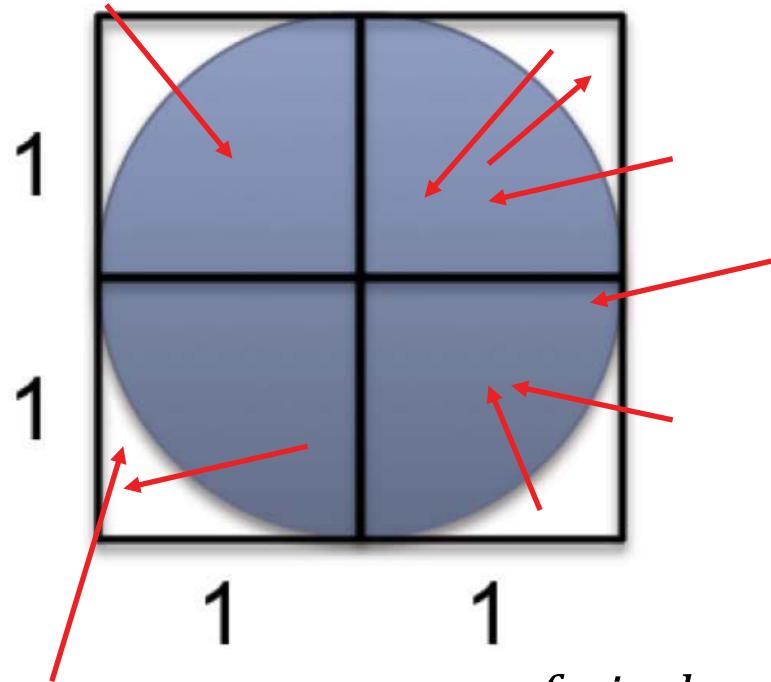
—1 Kings 7.23

~300 Years Later (Archimedes)



$$\frac{223}{71} < \pi < \frac{22}{7}$$

~2000 Years Later (Buffon-Laplace)



$$A_s = 2 * 2 = 4$$
$$A_c = \pi r^2 = \pi$$

$$\frac{\text{needles in circle}}{\text{needles in square}} = \frac{\text{area of circle}}{\text{area of square}}$$

$$\text{area of circle} = \frac{\text{area of square} * \text{needles in circle}}{\text{needles in square}}$$

$$\text{area of circle} = \frac{4 * \text{needles in circle}}{\text{needles in square}}$$

~200 Years Later



Crazy archer on closed course. Do not try ANYWHERE.

<https://www.youtube.com/watch?v=oYM6MljZ8IY>

Very End of Video



Simulating Buffon-Laplace Method

```
def throwNeedles(numNeedles):
    inCircle = 0
    for Needles in range(1, numNeedles + 1, 1):
        x = random.random()
        y = random.random()
        if (x*x + y*y)**0.5 <= 1.0:
            inCircle += 1
    return 4*(inCircle/float(numNeedles))
```

Simulating Buffon-Laplace Method, cont.

```
def getEst(numNeedles, numTrials):  
    estimates = []  
    for t in range(numTrials):  
        piGuess = throwNeedles(numNeedles)  
        estimates.append(piGuess)  
    sDev = stdDev(estimates)  
    curEst = sum(estimates)/len(estimates)  
    print('Est. = ' + str(curEst) + \  
          ', Std. dev. = ' + str(round(sDev, 6)) \  
          + ', Needles = ' + str(numNeedles))  
    return (curEst, sDev)
```

Simulating Buffon-Laplace Method, cont.

```
def estPi(precision, numTrials):  
    numNeedles = 1000  
    sDev = precision  
    while sDev >= precision/2:  
        curEst, sDev = getEst(numNeedles,  
                             numTrials)  
        numNeedles *= 2  
    return curEst  
  
estPi(0.005, 100)
```

Output

Est. = 3.1484400000000012, Std. dev. = 0.047886, Needles = 1000
Est. = 3.1391799999999987, Std. dev. = 0.035495, Needles = 2000
Est. = 3.1410799999999997, Std. dev. = 0.02713, Needles = 4000
Est. = 3.141435, Std. dev. = 0.016805, Needles = 8000
Est. = 3.141355, Std. dev. = 0.0137, Needles = 16000
Est. = 3.1413137500000006, Std. dev. = 0.008476, Needles = 32000
Est. = 3.14117187499999, Std. dev. = 0.007028, Needles = 64000
Est. = 3.141589687499993, Std. dev. = 0.004035, Needles = 128000
Est. = 3.141741406249995, Std. dev. = 0.003536, Needles = 256000
Est. = 3.14155671875, Std. dev. = 0.002101, Needles = 512000

Being Right is Not Good Enough

- Not sufficient to produce a good answer
- Need to have reason to believe that it is close to right
- In this case, small standard deviation implies that we are close to the true value of π

Right?

Is it Correct to State

- 95% of the time we run this simulation, we will estimate that the value of π is between 3.13743875875 and 3.14567467875?
- With a probability of 0.95 the actual value of π is between 3.13743875875 and 3.14567467875?
- Both are factually correct
- But only one of these statement can be inferred from our simulation
- *statistically valid* \neq *true*

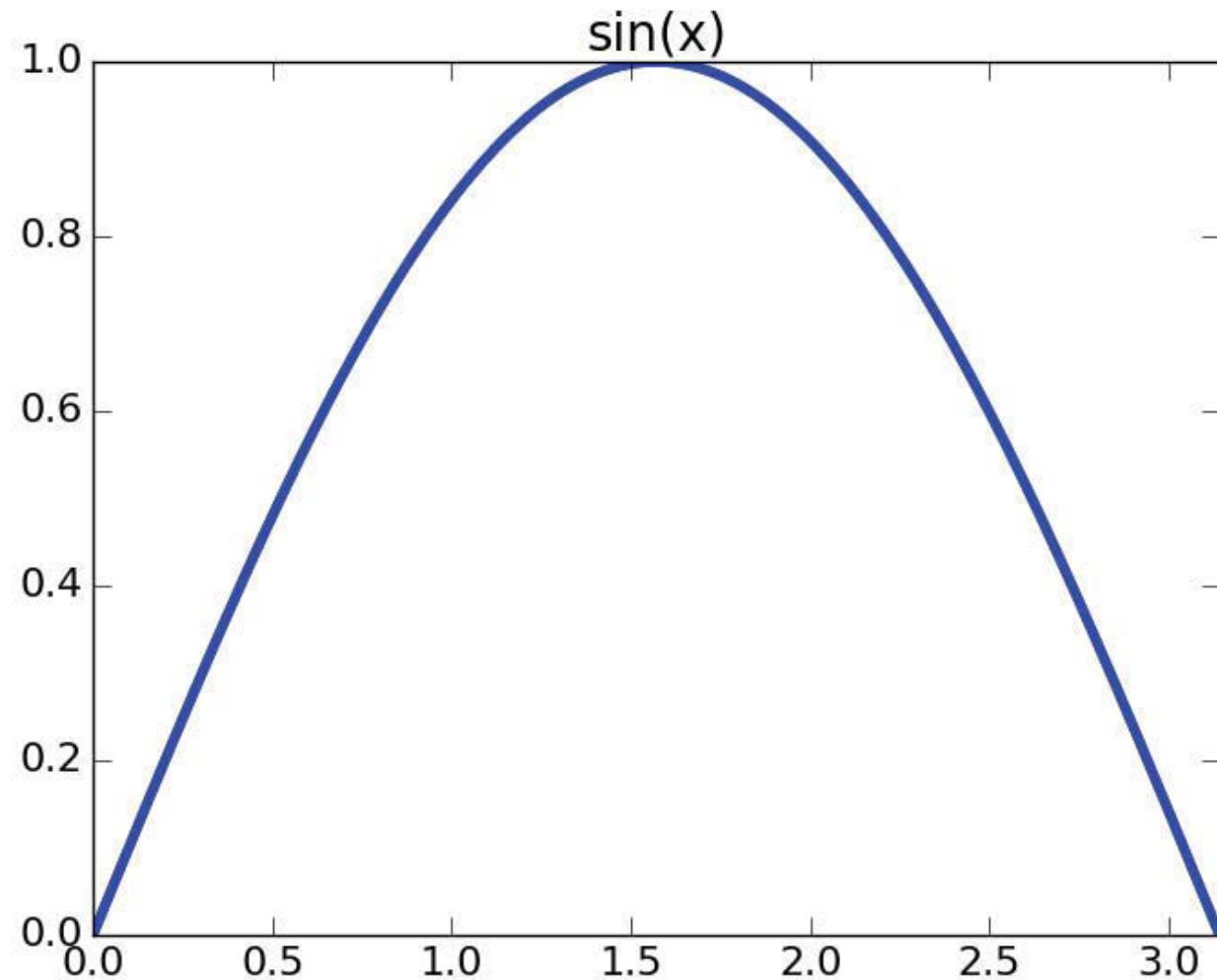
Introduce a Bug

```
def throwNeedles(numNeedles):
    inCircle = 0
    for Needles in range(1, numNeedles + 1, 1):
        x = random.random()
        y = random.random()
        if (x*x + y*y)**0.5 <= 1.0:
            inCircle += 1
    return 2*(inCircle/float(numNeedles))
```

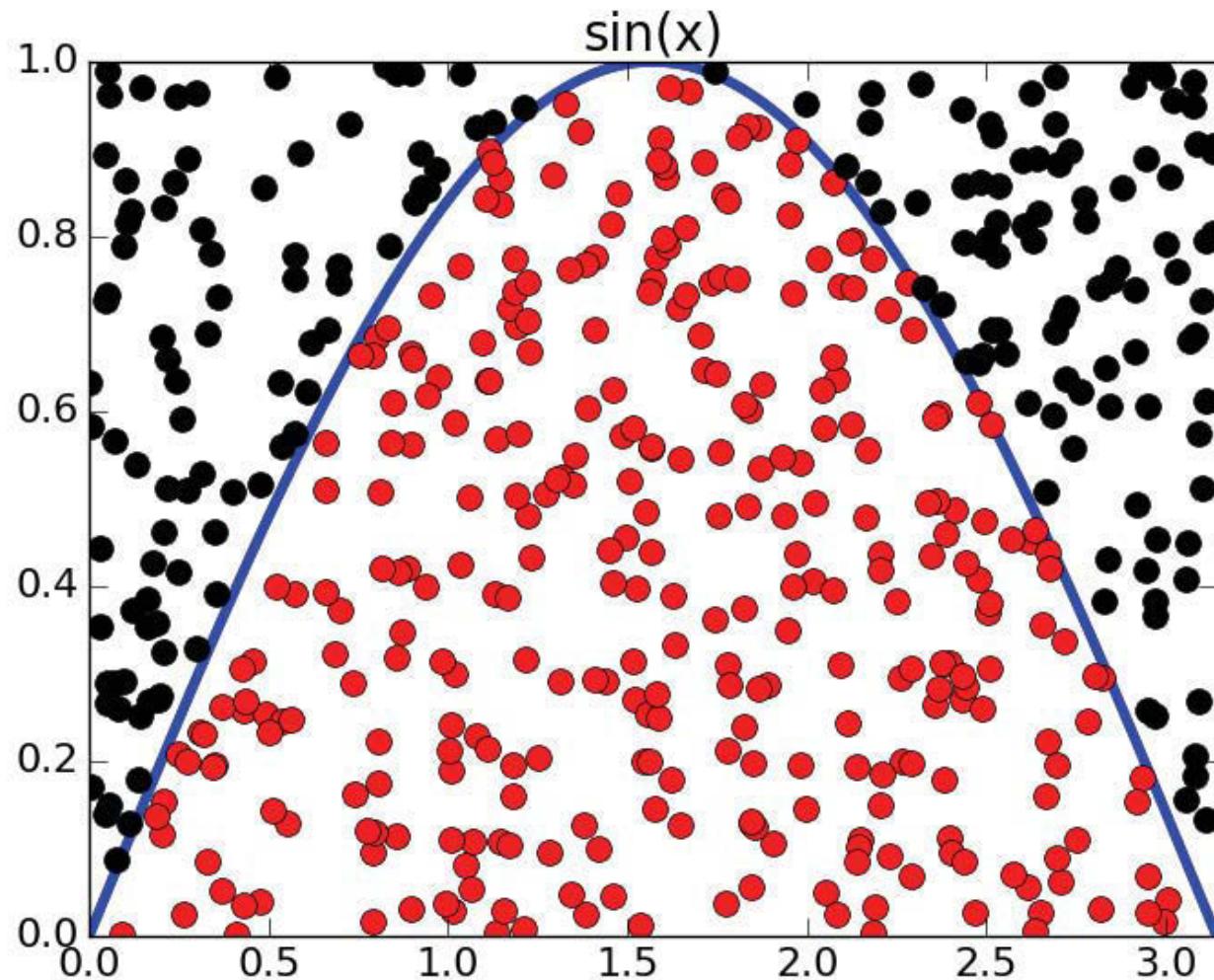
Generally Useful Technique

- To estimate the area of some region, R
 - Pick an enclosing region, E , such that the area of E is easy to calculate and R lies completely within E
 - Pick a set of random points that lie within E
 - Let F be the fraction of the points that fall within R
 - Multiply the area of E by F
- Way to estimate integrals

Sin(x)



Random Points



MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Lecture: Sampling and Standard Error

Announcements

- Relevant reading: Chapter 17
- No lecture Wednesday of next week!

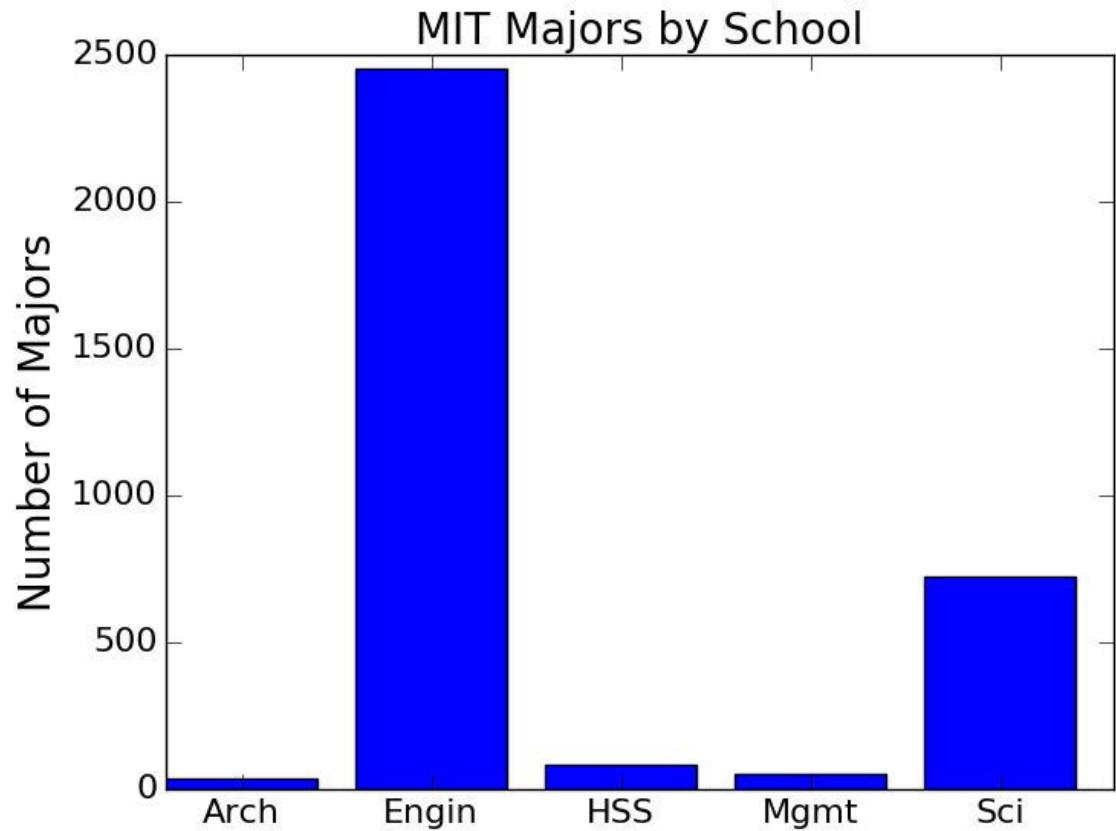
Recall Inferential Statistics

- Inferential statistics: making inferences about a populations by examining one or more random samples drawn from that population
- With Monte Carlo simulation we can generate lots of random samples, and use them to compute confidence intervals
- But suppose we can't create samples by simulation?
 - “According to the most recent poll Clinton leads Trump by 3.2 percentage points in swing states. The registered voter sample is 835 with with a margin of error of plus or minus 4 percentage points.” – October 2016

Probability Sampling

- Each member of the population has a nonzero probability of being included in a sample
- **Simple random sampling**: each member has an equal chance of being chosen
- Not always appropriate
 - Are MIT undergraduates nerds?
 - Consider a random sample of 100 students

Stratified Sampling



- Stratified sampling
 - Partition population into subgroups
 - Take a simple random sample from each subgroup

Stratified Sampling

- When there are small subgroups that should be represented
- When it is important that subgroups be represented proportionally to their size in the population
- Can be used to reduce the needed size of sample
 - Variability of subgroups less than of entire population
- Requires care to do properly
- Well stick to simple random samples

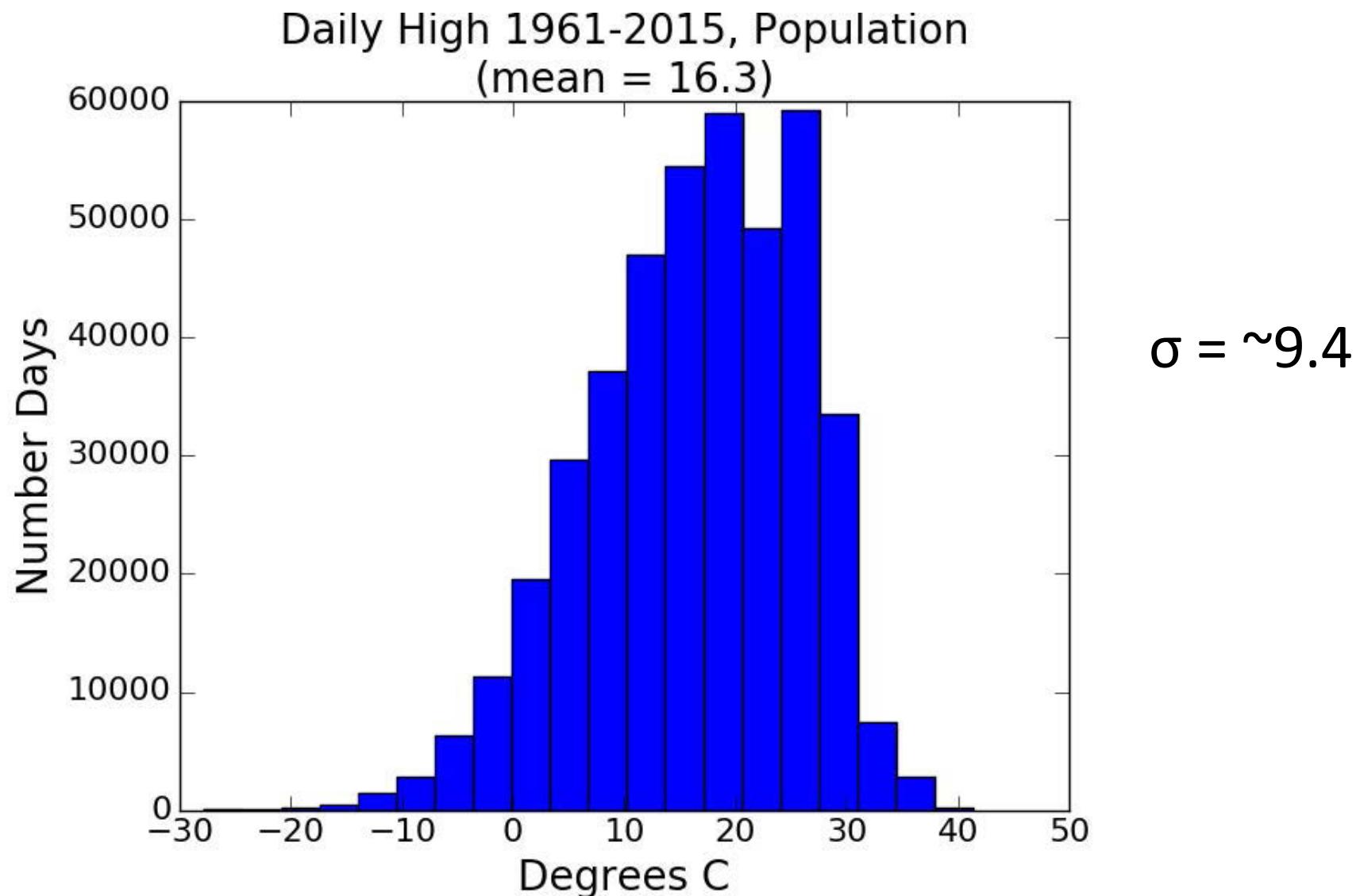
Data

- From U.S. National Centers for Environmental Information (NCEI)
- Daily high and low temperatures for
 - 21 different US cities
 - ALBUQUERQUE, BALTIMORE, BOSTON, CHARLOTTE, CHICAGO, DALLAS, DETROIT, LAS VEGAS, LOS ANGELES, MIAMI, NEW ORLEANS, NEW YORK, PHILADELPHIA, PHOENIX, PORTLAND, SAN DIEGO, SAN FRANCISCO, SAN JUAN, SEATTLE, ST LOUIS, TAMPA
 - 1961 – 2015
 - 421,848 data points (examples)
- Let's use some code to **look at the data**

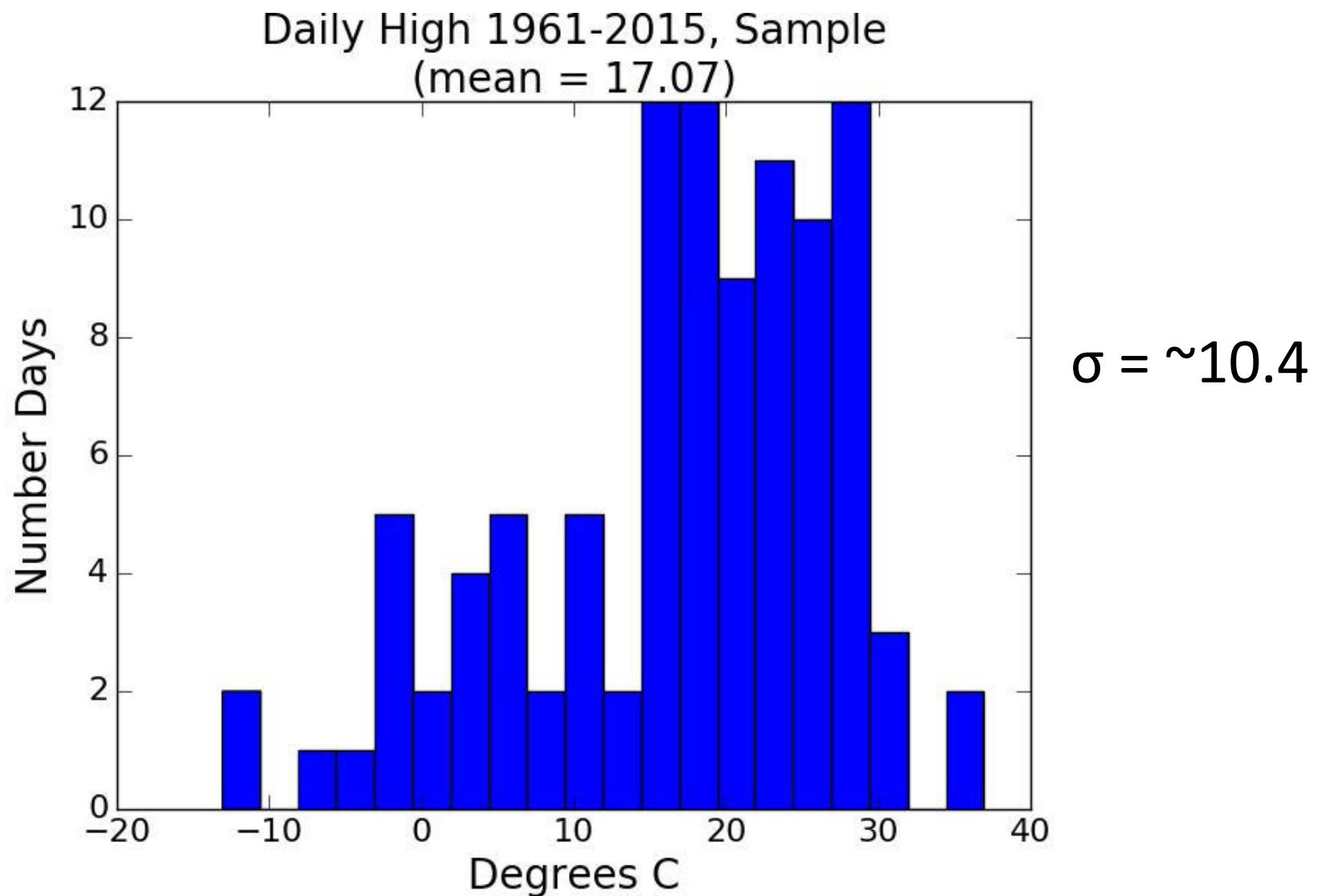
New in Code

- `numpy.std` is function in the `numpy` module that returns the standard deviation
- `random.sample(population, sampleSize)` returns a list containing `sampleSize` randomly chosen distinct elements of `population`
 - Sampling without replacement

Histogram of Entire Population



Histogram of Random Sample of Size 100



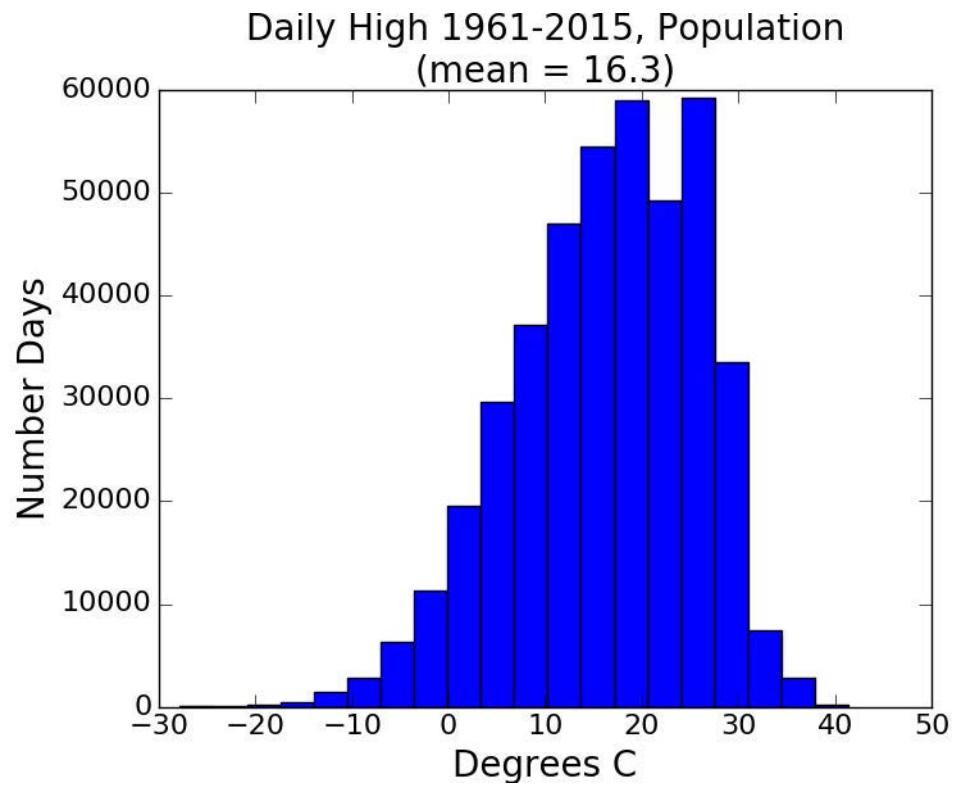
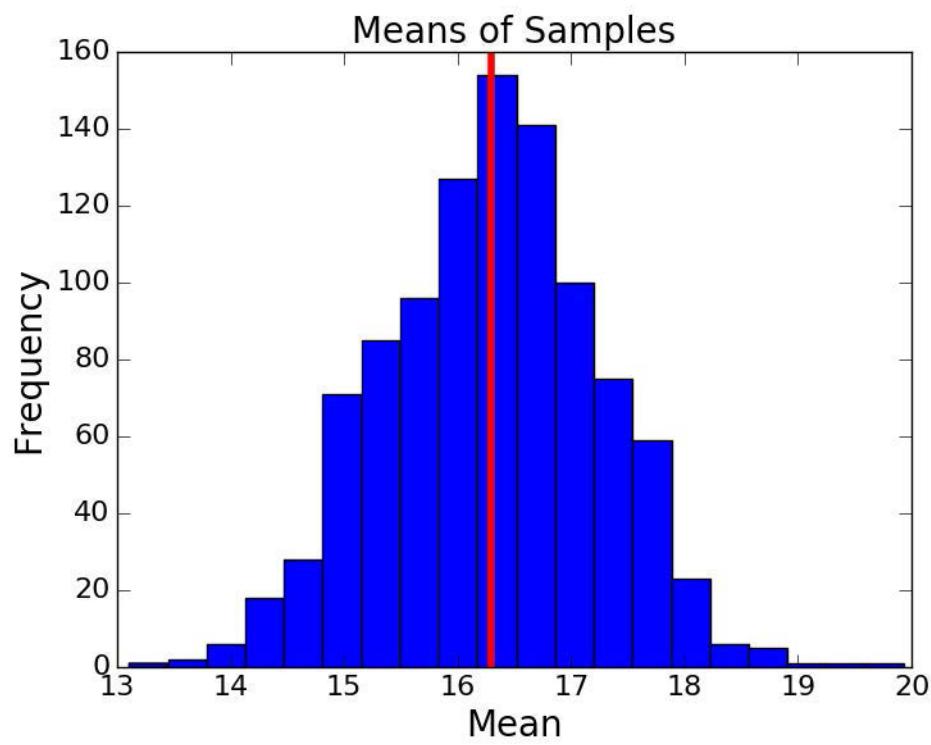
Means and Standard Deviations

- Population mean = 16.3
- Sample mean = 17.1
- Standard deviation of population = 9.44
- Standard deviation of sample = 10.4
- A happy accident, or something we should expect?
- Let's try it 1000 times and plot the results

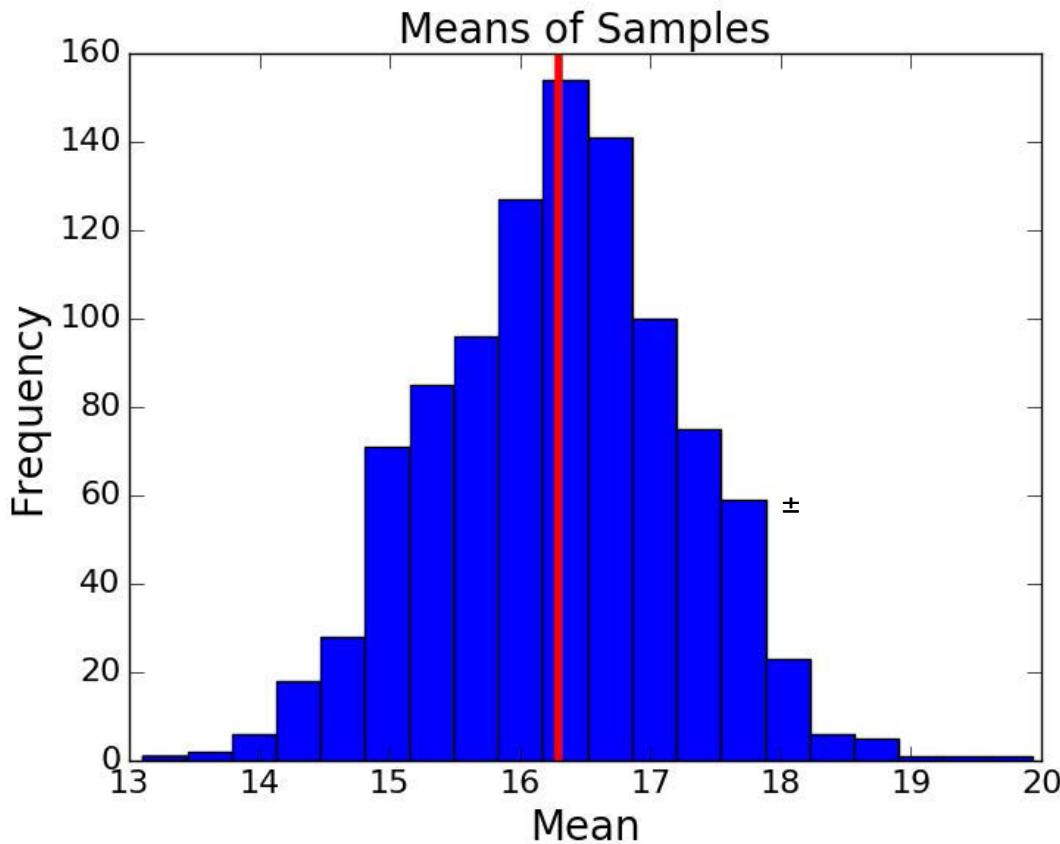
New in Code

- `pylab.axvline(x = popMean, color = 'r')` draws a red vertical line at `popMean` on the x-axis
- There's also a `pylab.axhline` function

Try It 1000 Times



Try It 1000 Times



Mean of sample Means = 16.3

Standard deviation of sample means = 0.94

What's the 95% confidence interval?

$$16.28 \pm 1.96 \cdot 0.94$$

$$14.5 - 18.1$$

Includes population mean, but pretty wide

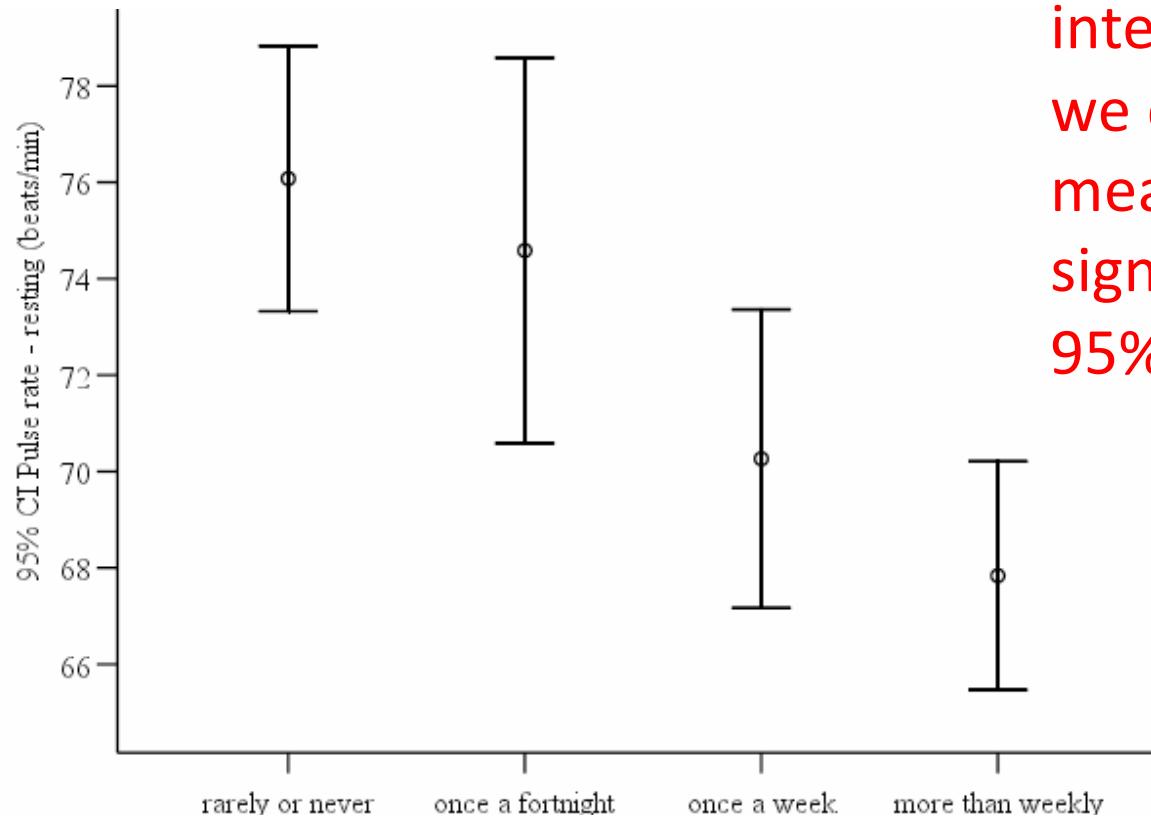
Suppose we want a tighter bound?

Getting a Tighter Bound

- Will drawing more samples help?
 - Let's try increasing from 1000 to 2000
 - Standard deviation goes from 0.943 to 0.946
- How about larger samples?
 - Let's try increasing sample size from 100 to 200
 - Standard deviation goes from 0.943 to 0.662

Error Bars, a Digression

- Graphical representation of the variability of data
- Way to visualize uncertainty



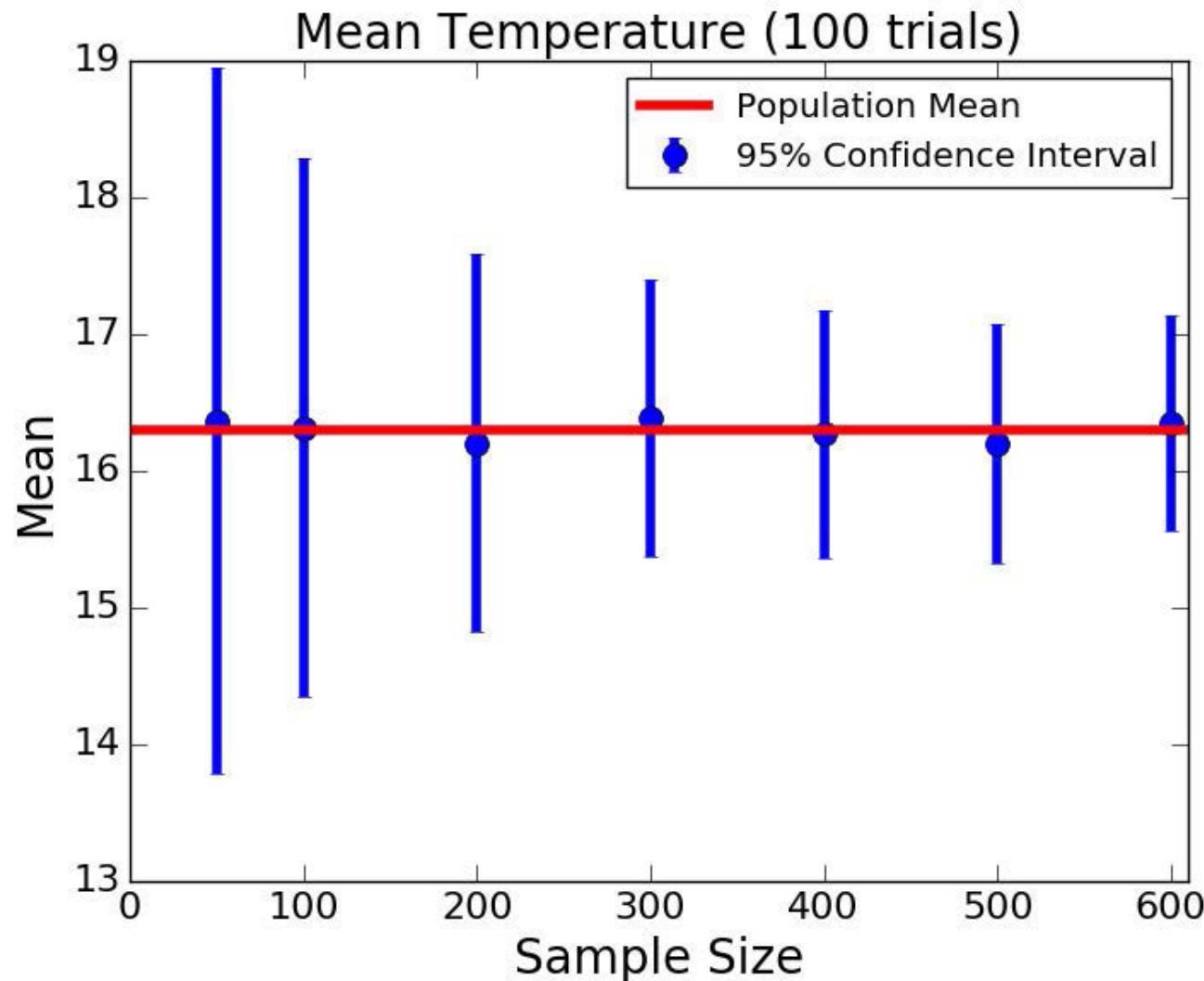
When confidence intervals don't overlap, we can conclude that means are statistically significantly different at 95% level.

https://upload.wikimedia.org/wikipedia/commons/1/1d/Pulse_Rate_Error_Bar_By_Exercise_Level.png

Let's Look at Error Bars for Temperatures

```
pylab.errorbar(xVals, sizeMeans,  
                yerr = 1.96*pylab.array(sizeSDs),  
                fmt = 'o',  
                label = '95% Confidence Interval')
```

Sample Size and Standard Deviation



Larger Samples Seem to Be Better

- Going from a sample size of 50 to 600 reduced the confidence interval from about 1.2C to about 0.34C.
- But we are now looking at $600 * 100 = 600k$ examples
 - What has sampling bought us?
 - Absolutely Nothing!
 - Entire population contained $\sim 422k$ samples

What Can We Conclude from 1 Sample?

- More than you might think
- Thanks to the Central Limit Theorem

Recall Central Limit Theorem

- Given a sufficiently large sample:
 - 1) The means of the samples in a set of samples (the sample means) will be approximately normally distributed,
 - 2) This normal distribution will have a mean close to the mean the population, and
 - 3) The variance of the sample means will be close to the variance of the population divided by the sample size.
- Time to use the 3rd feature
- Compute *standard error of the mean* (SEM or SE)

Standard Error of the Mean

$$SE = \frac{\sigma}{\sqrt{n}}$$

```
def sem(popSD, sampleSize):  
    return popSD/sampleSize**0.5
```

- Does it work?

Testing the SEM

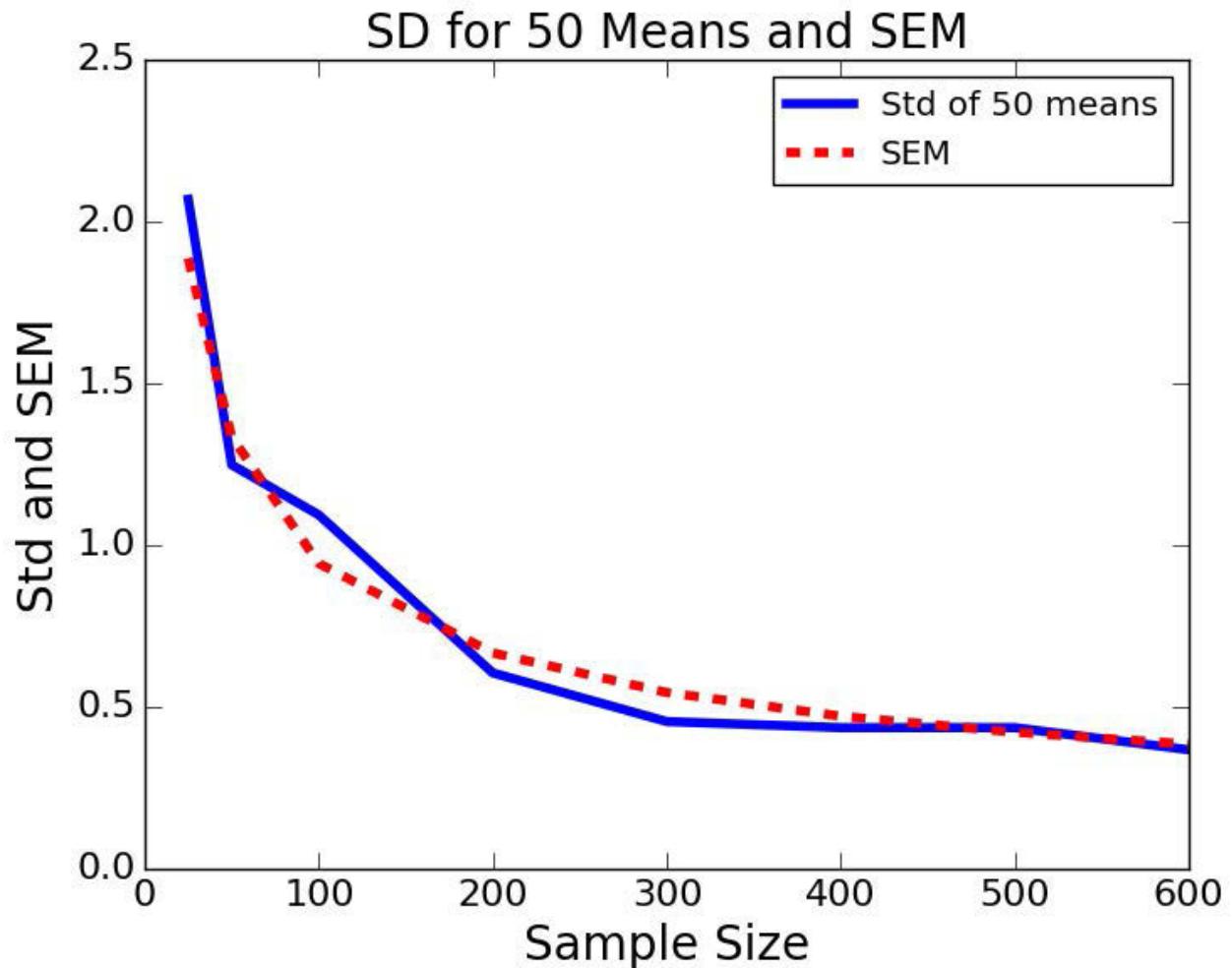
```
sampleSizes = (25, 50, 100, 200, 300, 400, 500, 600)
numTrials = 50
population = getHighs()
popSD = numpy.std(population)
sems = []
sampleSDs = []
for size in sampleSizes:
    sems.append(sem(popSD, size))
    means = []
    for t in range(numTrials):
        sample = random.sample(population, size)
        means.append(sum(sample)/len(sample))
    sampleSDs.append(numpy.std(means))
pylab.plot(sampleSizes, sampleSDs,
           label = 'Std of ' + str(numTrials) + ' means')
pylab.plot(sampleSizes, sems, 'r--', label = 'SEM')
pylab.xlabel('Sample Size')
pylab.ylabel('Std and SEM')
pylab.title('SD for ' + str(numTrials) + ' Means and SEM')
pylab.legend()
```

Standard Error of the Mean

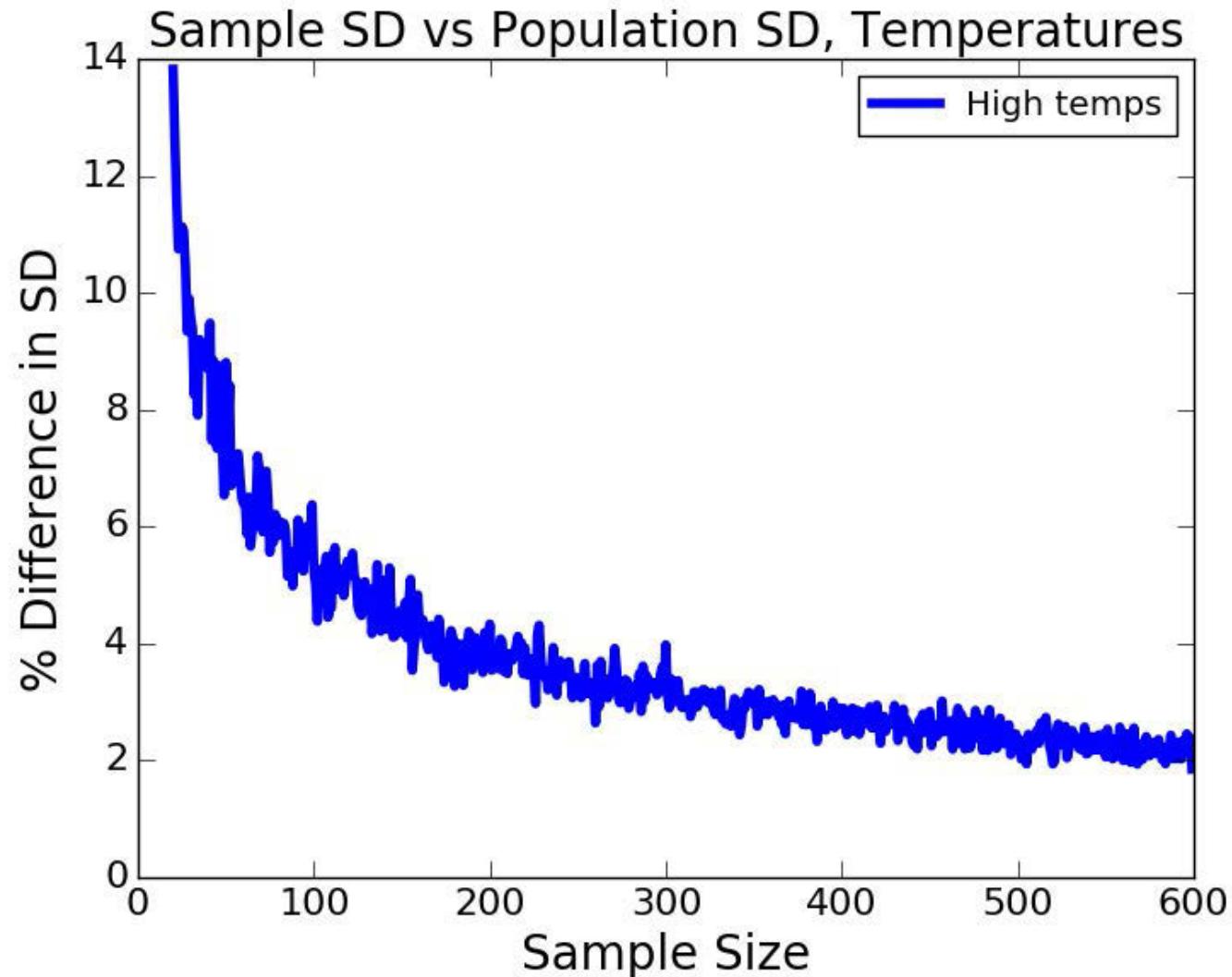
$$SE = \frac{\sigma}{\sqrt{n}}$$

But, we don't know standard deviation of population

How might we approximate it?

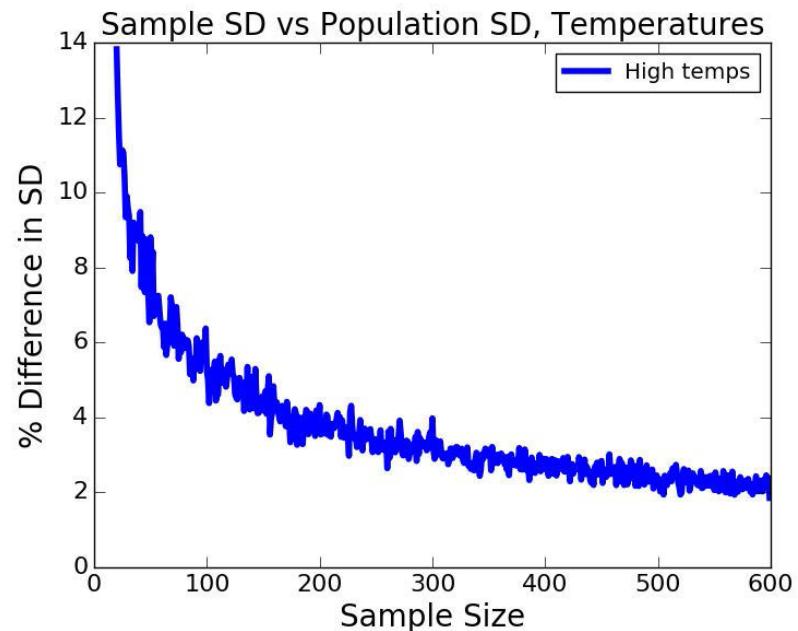


Sample SD vs. Population SD



The Point

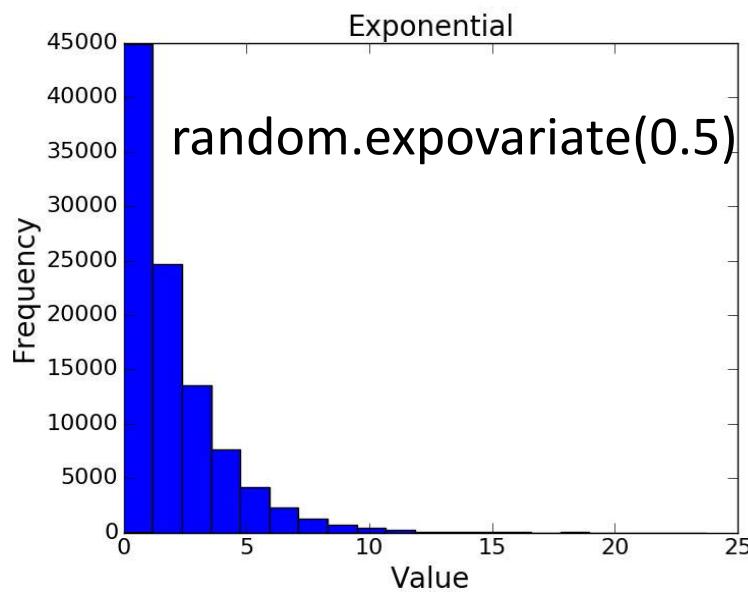
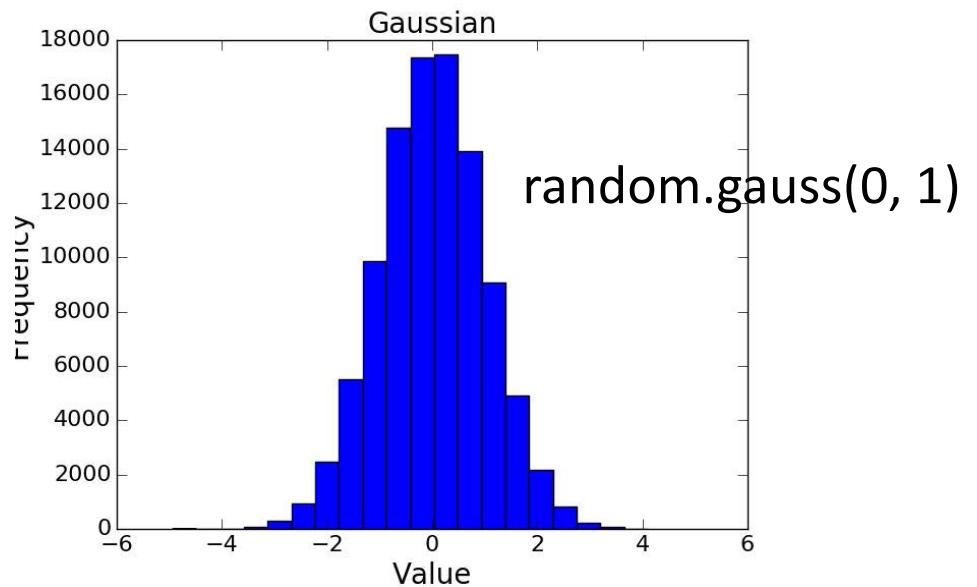
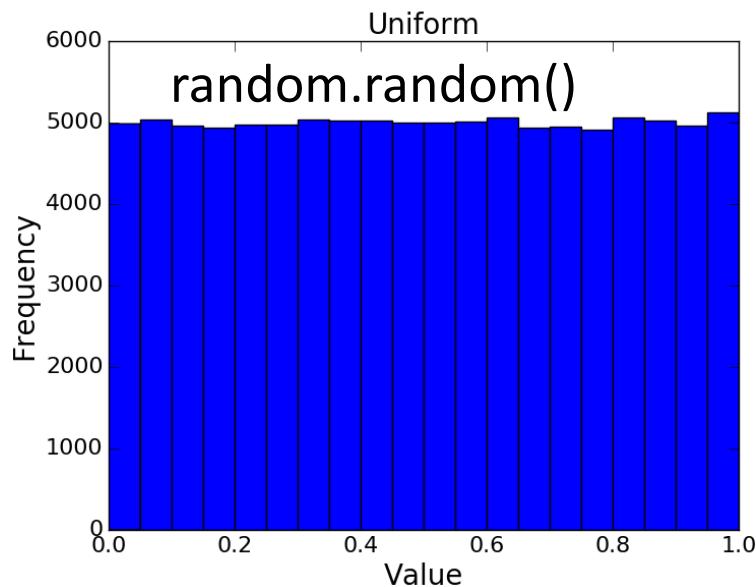
- Once sample reaches a reasonable size, sample standard deviation is a pretty good approximation to population standard deviation
- True only for this example?
 - Distribution of population?
 - Size of population?



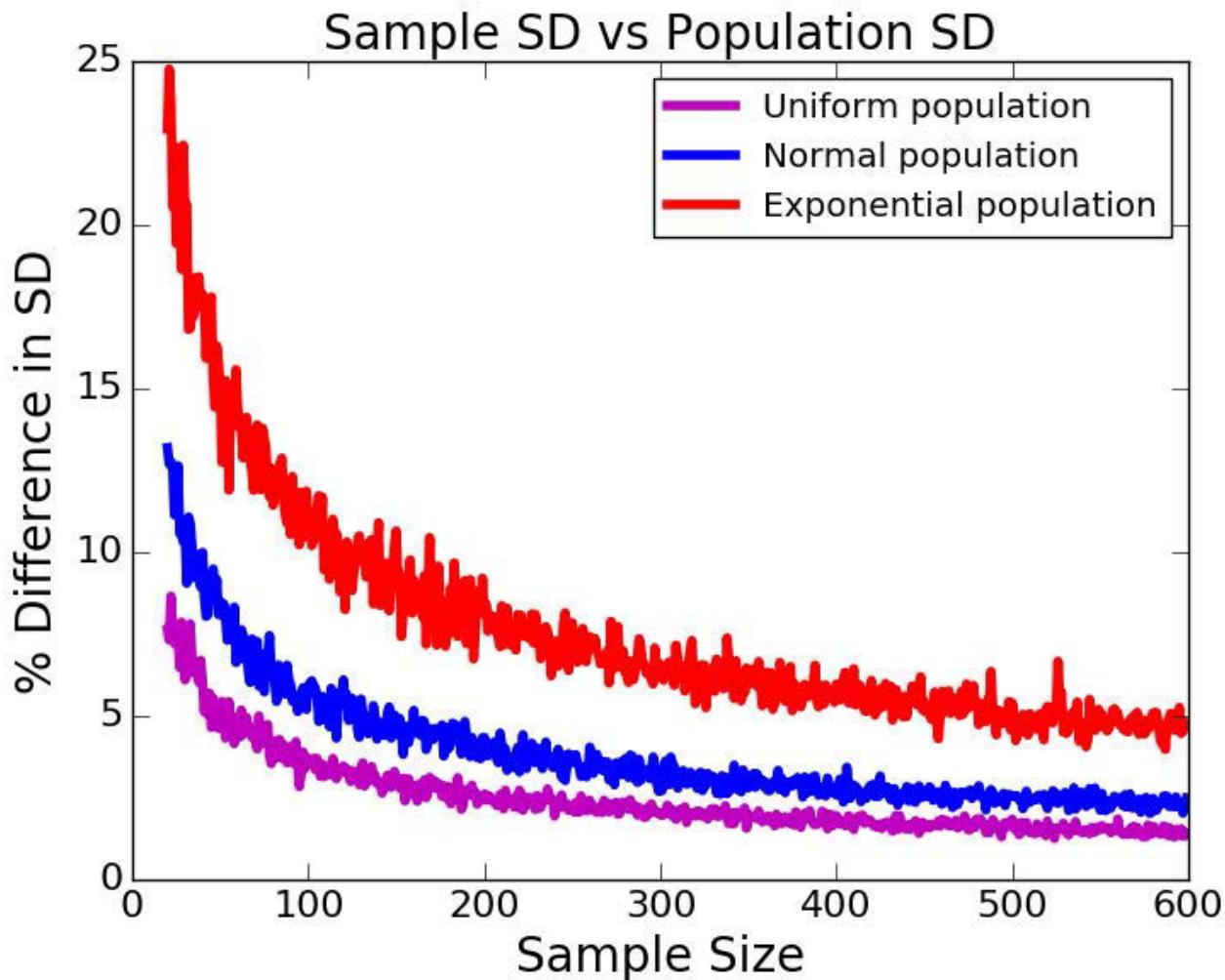
Looking at Distributions

```
def plotDistributions():
    uniform, normal, exp = [], [], []
    for i in range(100000):
        uniform.append(random.random())
        normal.append(random.gauss(0, 1))
        exp.append(random.expovariate(0.5))
    makeHist(uniform, 'Uniform', 'Value', 'Frequency')
    pylab.figure()
    makeHist(normal, 'Gaussian', 'Value', 'Frequency')
    pylab.figure()
    makeHist(exp, 'Exponential', 'Value', 'Frequency')
```

Three Different Distributions

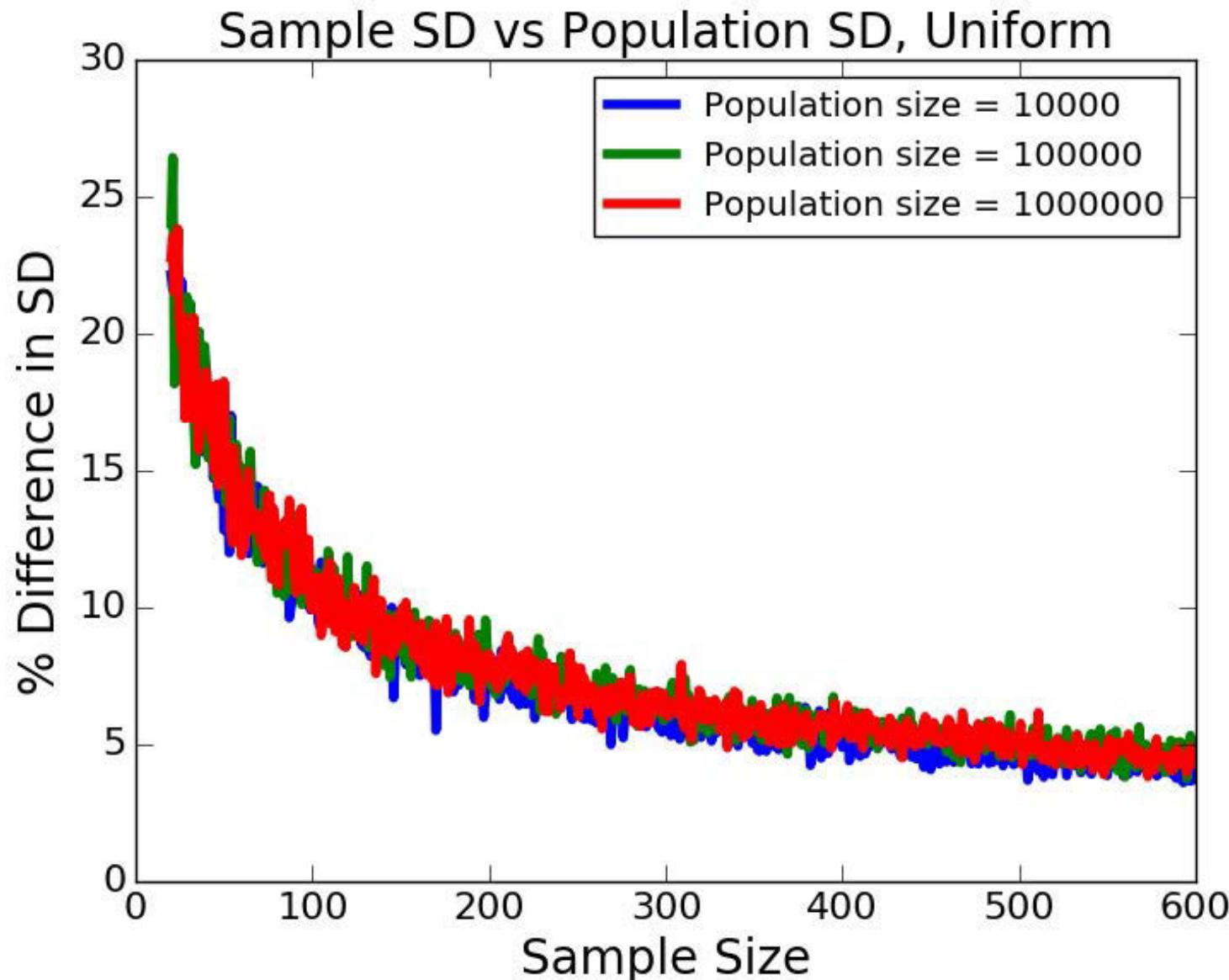


Does Distribution Matter?



Skew, a measure of the asymmetry of a probability distribution, matters

Does Population Size Matter?



To Estimate Mean from a Single Sample

- 1) Choose sample size based on estimate of skew in population
- 2) Chose a random sample from the population
- 3) Compute the mean and standard deviation of that sample
- 4) Use the standard deviation of that sample to estimate the SE
- 5) Use the estimated SE to generate confidence intervals around the sample mean

Works great when we choose independent random samples.

Not always so easy to do, as political pollsters keep learning.

Are 200 Samples Enough?

```
numBad = 0
for t in range(numTrials):
    sample = random.sample(tempList, sampleSize)
    sampleMean = sum(sample)/sampleSize
    se = numpy.std(sample)/sampleSize**0.5
    if abs(popMean - sampleMean) > 1.96*se:
        numBad += 1
print('Fraction outside 95% confidence interval =',
      numBad/numTrials)
```

Fraction outside 95% confidence interval = 0.0511

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Understanding Experimental Data

Eric Grimson

MIT Department Of Electrical Engineering and
Computer Science

Announcements

- Reading: Chapter 18
- No lecture on Wednesday

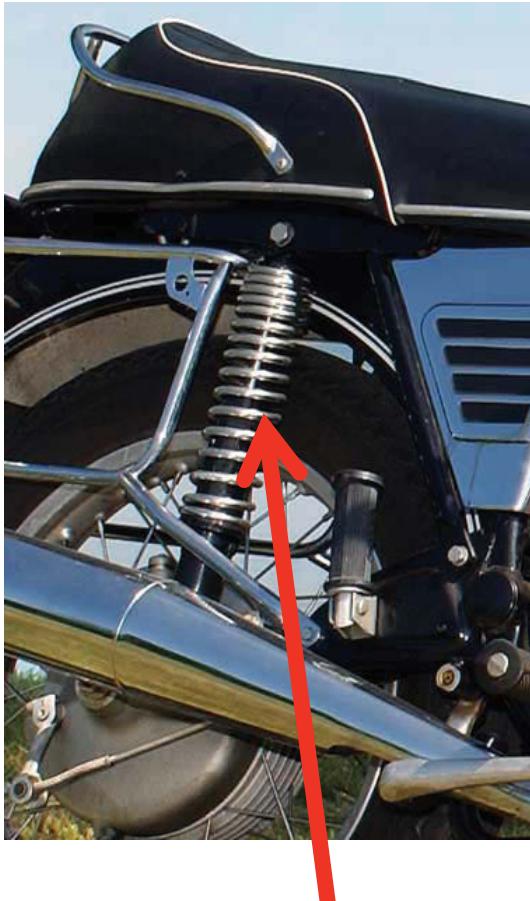
Statistics Meets Experimental Science

- Conduct an experiment to gather data
 - Physical (e.g., in a biology lab)
 - Social (e.g., questionnaires)
- Use theory to generate some questions about data
 - Physical (e.g., gravitational fields)
 - Social (e.g., people give inconsistent answers)
- Design a computation to help answer questions about data

Net Gain on a missed jump shot

$$= P(\text{off reb}) \times E[\text{pts for}] - P(\text{def reb}) \times E[\text{pts against}]$$
- Consider, for example, a spring

This Kind of Spring



$$k \approx 35,000 \text{ N/m}$$

$$k \approx 1 \text{ N/m} \rightarrow$$



Linear spring: amount of force needed to stretch or compress spring is linear in the distance the spring is stretched or compressed

Each spring has a spring constant, k , that determines how much force is needed

Newton = force to accelerate 1 kg mass 1 meter per second per second

Hooke's Law

- $F = -kd$
- How much does a rider have to weigh to compress spring 1cm?

$$F = 0.01m * 35,000 N/m$$

$$F = 350 N$$

$$mass * 9.8 m/s^2 = 350 N$$

$$mass = 350 N / 9.81 m/s^2$$

$$mass = 350 k / 9.81 \leftarrow \text{This } k \text{ refers to kilograms, not the spring constant!}$$

$$mass \approx 35.68 k$$

$$F = mass * acc$$

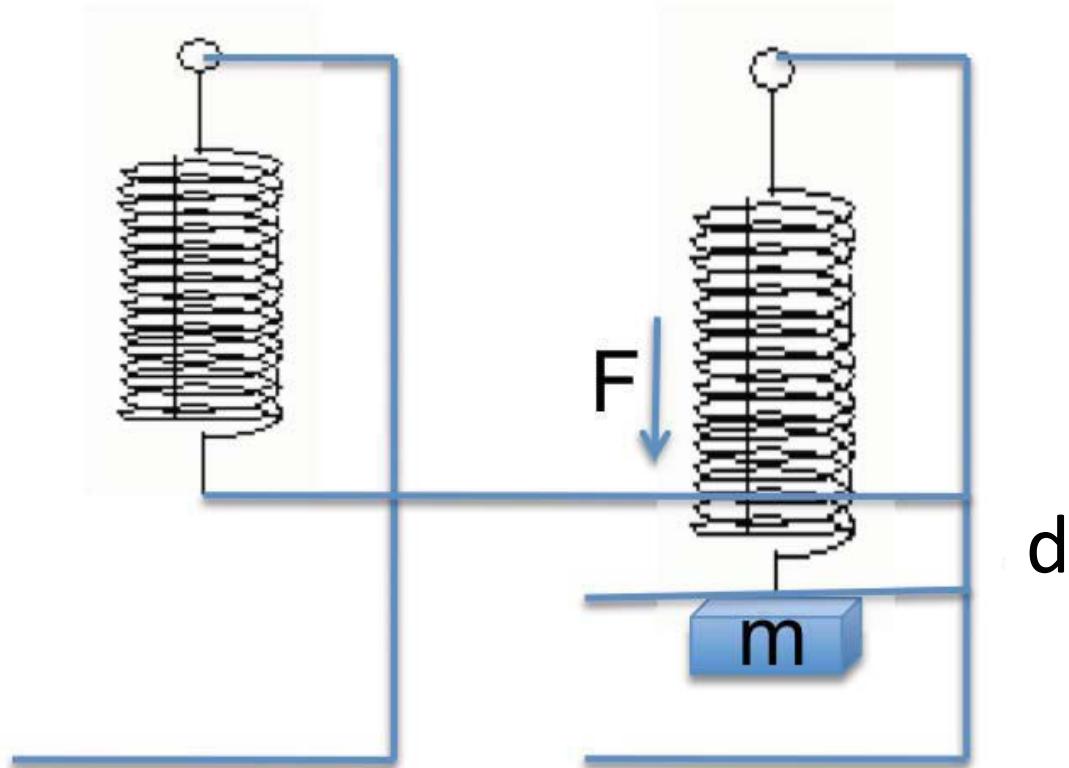
$$F = mass * 9.8 m/s^2$$



Images of suspension spring © source unknown. All rights reserved.
This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.

Finding k

- $F = -kd$
- $k = -F/d$
- $k = 9.81*m/d$



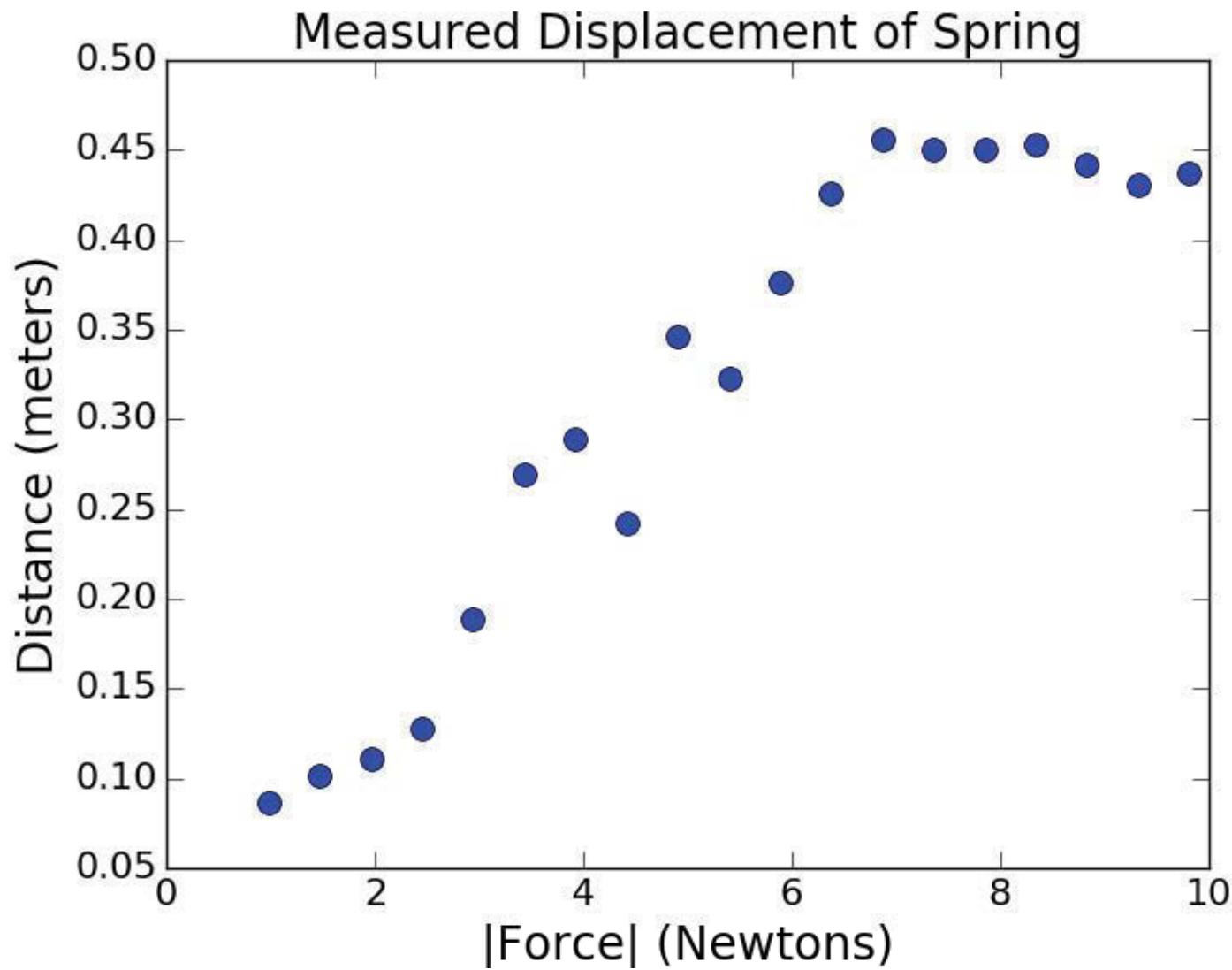
Some Data

Distance (m)	Mass (kg)
0.0865	0.1
0.1015	0.15
0.1106	0.2
0.1279	0.25
0.1892	0.3
0.2695	0.35
0.2888	0.4
0.2425	0.45
0.3465	0.5
0.3225	0.55
0.3764	0.6
0.4263	0.65
0.4562	0.7

Taking a Look at the Data

```
def plotData(fileName):
    xVals, yVals = getData(fileName)
    xVals = pylab.array(xVals)
    yVals = pylab.array(yVals)
    xVals = xVals*9.81 #acc. due to gravity
    pylab.plot(xVals, yVals, 'bo',
               label = 'Measured displacements')
labelPlot()
```

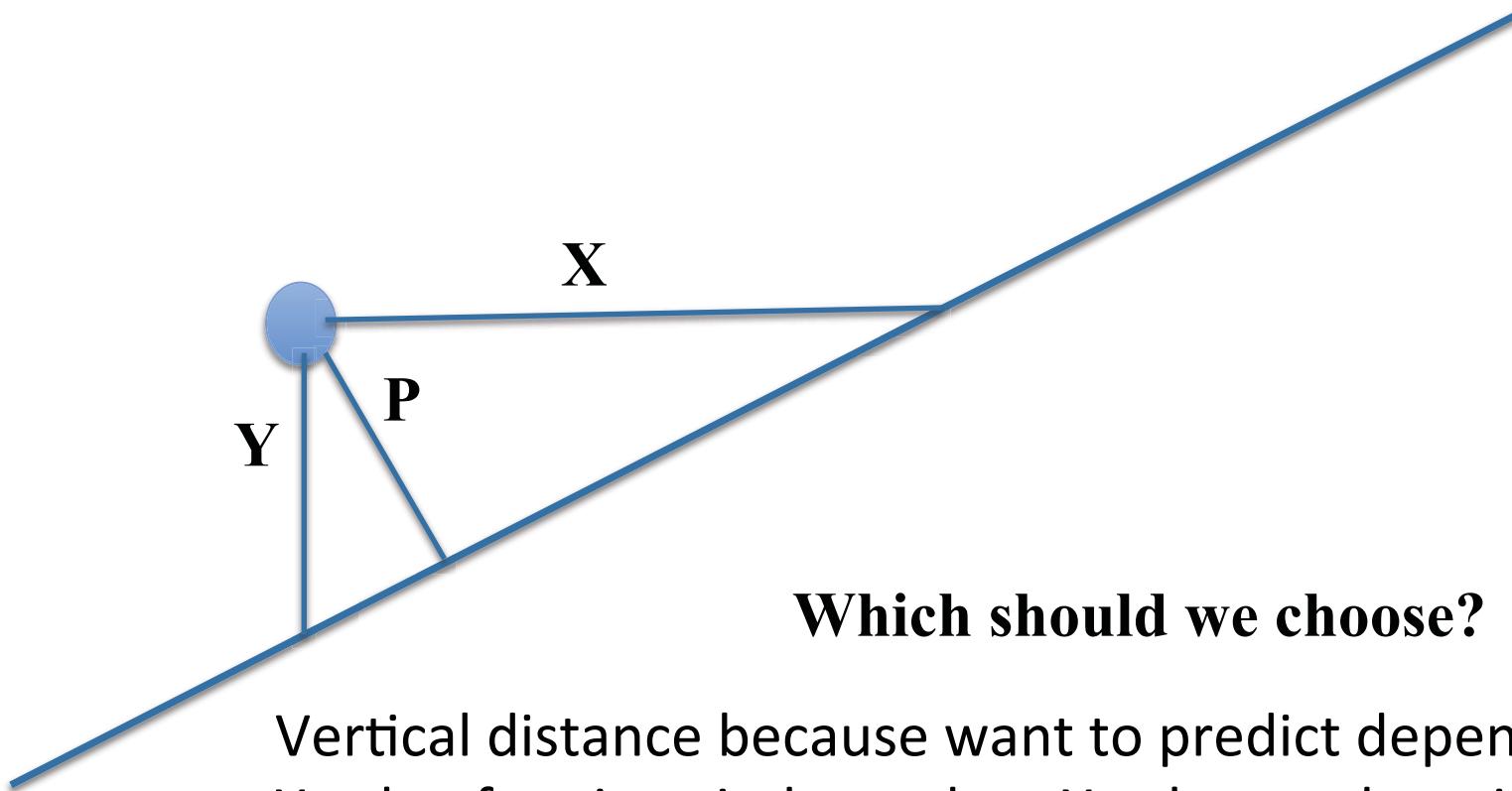
Taking a Look at the Data



Fitting Curves to Data

- When we fit a curve to a set of data, we are finding a fit that relates an independent variable (the mass) to an estimated value of a dependent variable (the distance)
- To decide how well a curve fits the data, we need a way to measure the goodness of the fit – called the **objective function**
- Once we define the objective function, we want to find the curve that minimizes it
- In this case, we want to find a line such that some function of the sum of the distances from the line to the measured points is minimized

Measuring Distance



Vertical distance because want to predict dependent Y value for given independent X value, and vertical distance measures error in that prediction

Least Squares Objective Function

$$\sum_{i=0}^{\text{len}(\text{observed})-1} (\text{observed}[i] - \text{predicted}[i])^2$$

- Look familiar?
 - This is variance times number of observations
 - So minimizing this will also minimize the variance

Solving for Least Squares

$$\sum_{i=0}^{\text{len}(\text{observed})-1} (\text{observed}[i] - \text{predicted}[i])^2$$

- To minimize this objective function, want to find a curve for the predicted observations that leads to minimum value
- Use **linear regression** to find a polynomial representation for the predicted model

Polynomials with One Variable (x)

- 0 or sum of finite number of non-zero terms
- Each term of the form cx^p
 - c , the coefficient, a real number
 - p , the degree of the term, a non-negative integer
- The degree of the polynomial is the largest degree of any term
- Examples
 - Line: $ax + b$
 - Parabola: $ax^2 + bx + c$

Solving for Least Squares

$$\sum_{i=0}^{\text{len}(\text{observed})-1} (\text{observed}[i] - \text{predicted}[i])^2$$

- Simple example:
 - Use a degree-one polynomial, $y = ax+b$, as model of our data (we want best fitting line)
- Find values of a and b such that when we use the polynomial to compute y values for all of the x values in our experiment, the squared difference of these **predicted** values and the corresponding **observed** values is minimized
- A **linear regression** problem
- Many algorithms for doing this, including one similar to Newton's method (which you saw in 6.0001)

polyFit

- Good news is that pylab provides built in functions to find these polynomial fits
- `pylab.polyfit(observedX, observedY, n)`
- Finds coefficients of a polynomial of degree n , that provides a best least squares fit for the observed data
 - $n = 1$ – best line $y = ax + b$
 - $n = 2$ – best parabola $y = ax^2 + bx + c$

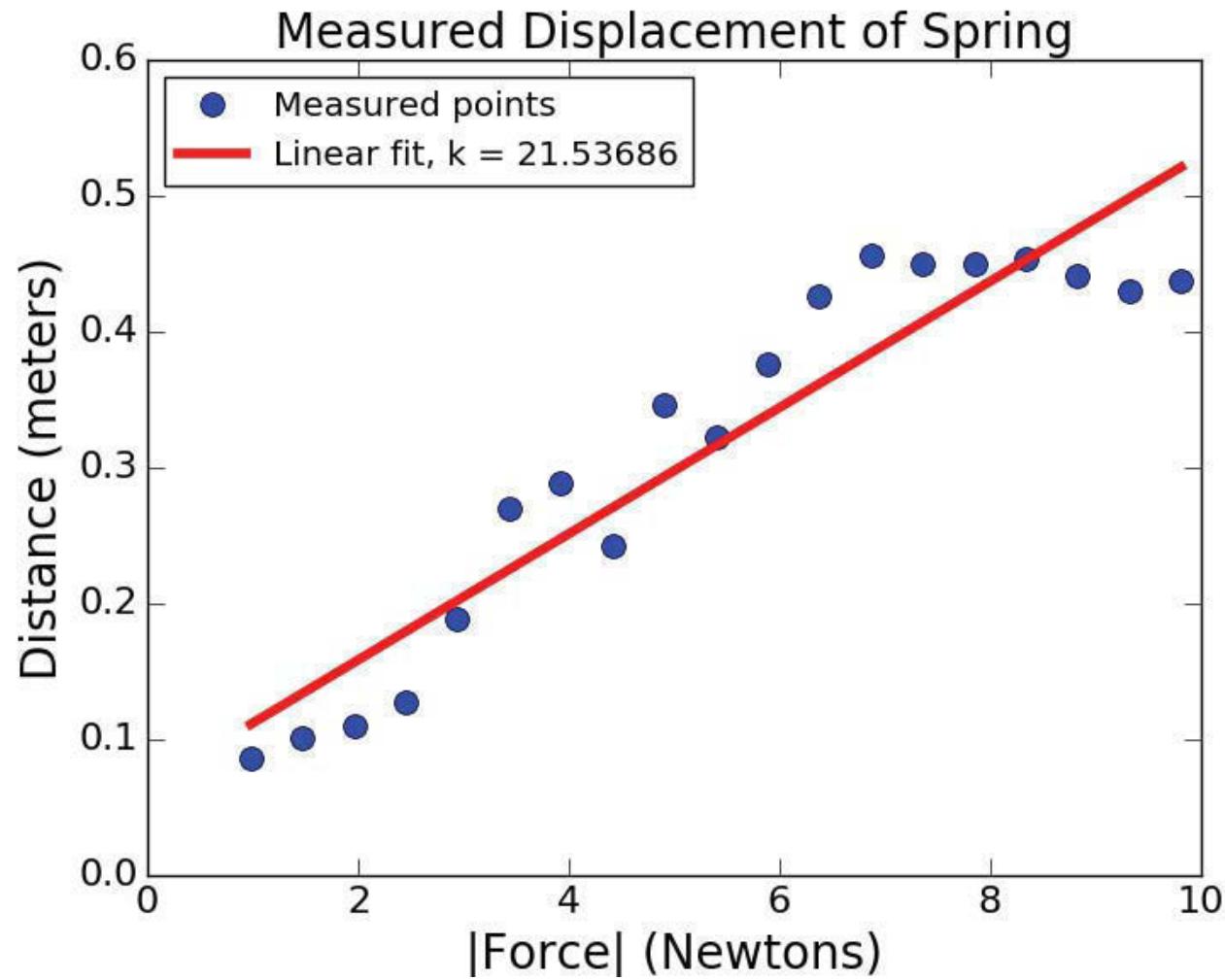
Using polyfit

```
def fitData(fileName):
    xVals, yVals = getData(fileName)
    xVals = pylab.array(xVals)
    yVals = pylab.array(yVals)
    xVals = xVals*9.81 #get force
    pylab.plot(xVals, yVals, 'bo',
               label = 'Measured points')
    labelPlot()
    a,b = pylab.polyfit(xVals, yVals, 1)
    estYVals = a*pylab.array(xVals) + b
    print('a = ', a, 'b = ', b)
    pylab.plot(xVals, estYVals, 'r',
               label = 'Linear fit, k = '
               + str(round(1/a, 5)))
    pylab.legend(loc = 'best')
```

plotData

Note that conversion to array is redundant here

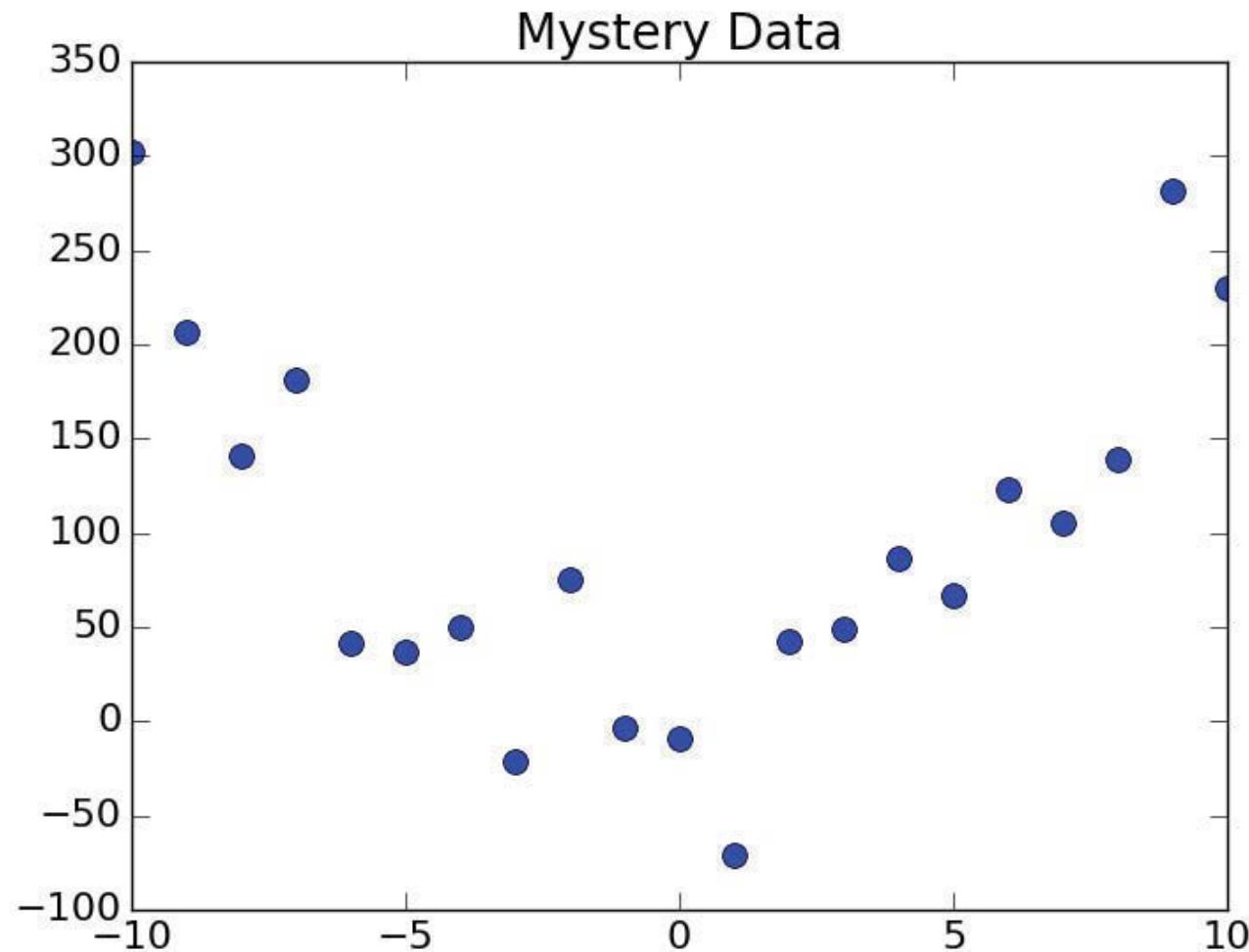
Visualizing the Fit



Version Using polyval

```
def fitData1(fileName):
    xVals, yVals = getData(fileName)
    xVals = pylab.array(xVals)
    yVals = pylab.array(yVals)
    xVals = xVals*9.81 #get force
    pylab.plot(xVals, yVals, 'bo',
                label = 'Measured points')
    labelPlot()
    model = pylab.polyfit(xVals, yVals, 1)
    estYVals = pylab.polyval(model, xVals)
    pylab.plot(xVals, estYVals, 'r',
                label = 'Linear fit, k = '
                + str(round(1/model[0], 5)))
    pylab.legend(loc = 'best')
```

Another Experiment



Fit a Line

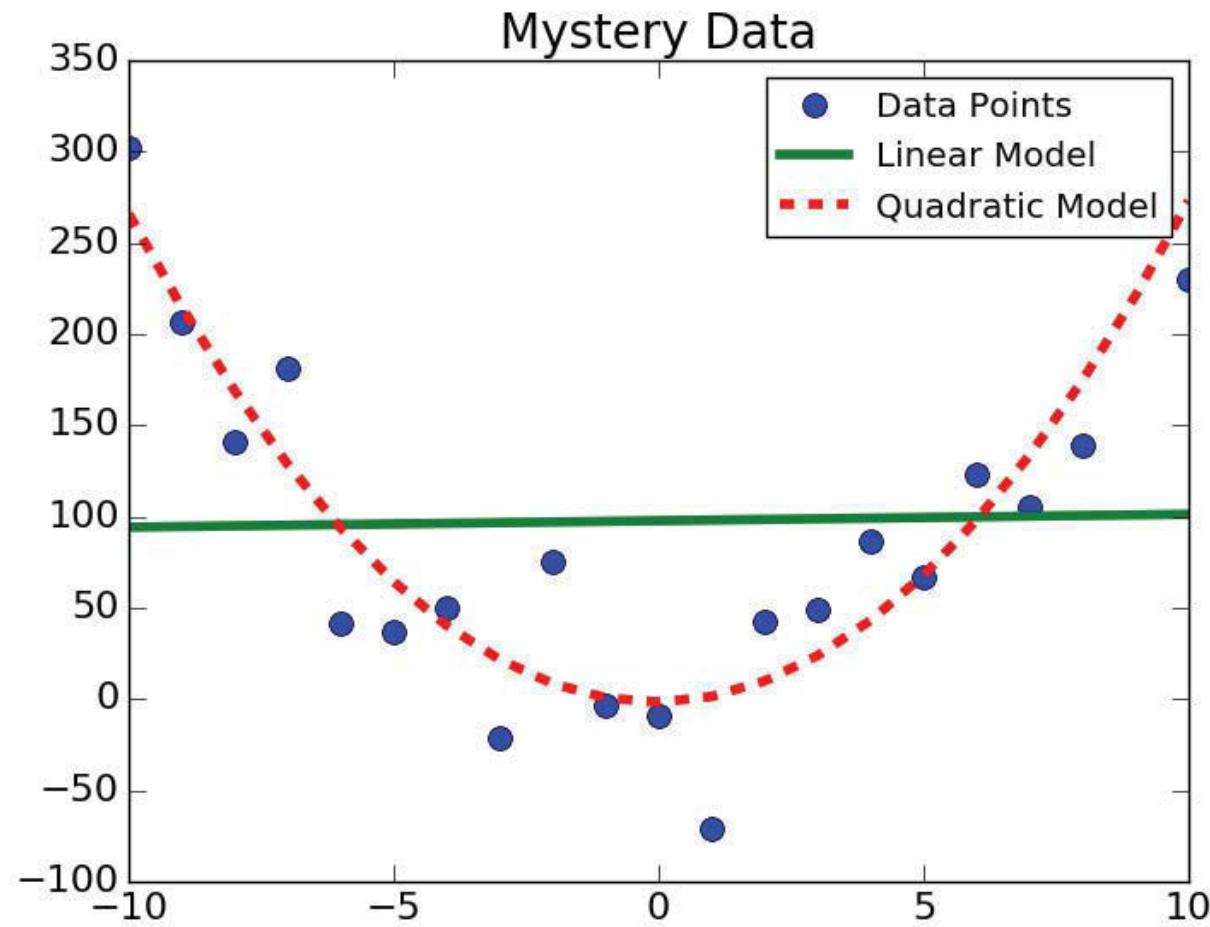


Let's Try a Higher-degree Model

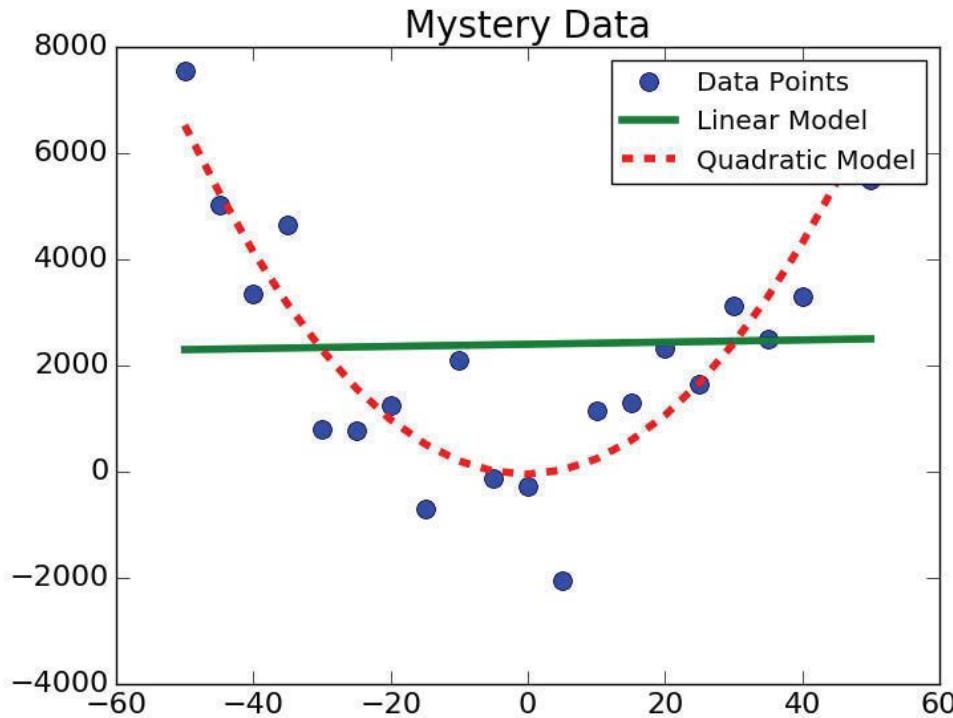
```
model2 = pylab.polyfit(xVals, yVals, 2)
pylab.plot(xVals, pylab.polyval(model2, xVals),
           'r--', label = 'Quadratic Model')
```

Note that this is still an example of linear regression,
even though we are not fitting a line to the data (in this
case we are finding the best parabola)

Quadratic Appears to be a Better Fit



How Good Are These Fits?



- Relative to each other
- In an absolute sense

Relative to Each Other

- Fit is a function from the independent variable to the dependent variable
- Given an independent value, provides an estimate of the dependent value
- Which fit provides better estimates?
- Since we found fit by minimizing mean square error, could just evaluate goodness of fit by looking at that error

Comparing Mean Squared Error

```
def aveMeanSquareError(data, predicted):
    error = 0.0
    for i in range(len(data)):
        error += (data[i] - predicted[i])**2
    return error/len(data)

estYVals = pylab.polyval(model1, xVals)
print('Ave. mean square error for linear model =',
      aveMeanSquareError(yVals, estYVals))
estYVals = pylab.polyval(model2, xVals)
print('Ave. mean square error for quadratic model =',
      aveMeanSquareError(yVals, estYVals))
```

Ave. mean square error for linear model = 9372.73078965
Ave. mean square error for quadratic model = 1524.02044718

In an Absolute Sense

- Mean square error useful for comparing two different models for the same data
- Useful for getting a sense of absolute goodness of fit?
 - Is 1524 good?
- Hard to know, since there is no upper bound and not scale independent
- Instead we use coefficient of determination, R^2 ,

$$R^2 = 1 - \frac{\sum_i (y_i - p_i)^2}{\sum_i (y_i - \mu)^2}$$

← Error in estimates
← Variability in measured data

y_i are measured values
 p_i are predicted values
 μ is mean of measured values

If You Prefer Code

$$R^2 = 1 - \frac{\sum_i (y_i - p_i)^2}{\sum_i (y_i - \mu)^2}$$

```
def rSquared(observed, predicted):
    error = ((predicted - observed)**2).sum()
    meanError = error/len(observed)
    return 1 - (meanError/numpy.var(observed))
```

I am playing a clever trick here:

- Numerator is sum of squared errors
- Dividing by number of samples gives average sum-squared-error
- Denominator is variance times number of samples
- So mean SSE/variance is same as R^2 ratio

R²

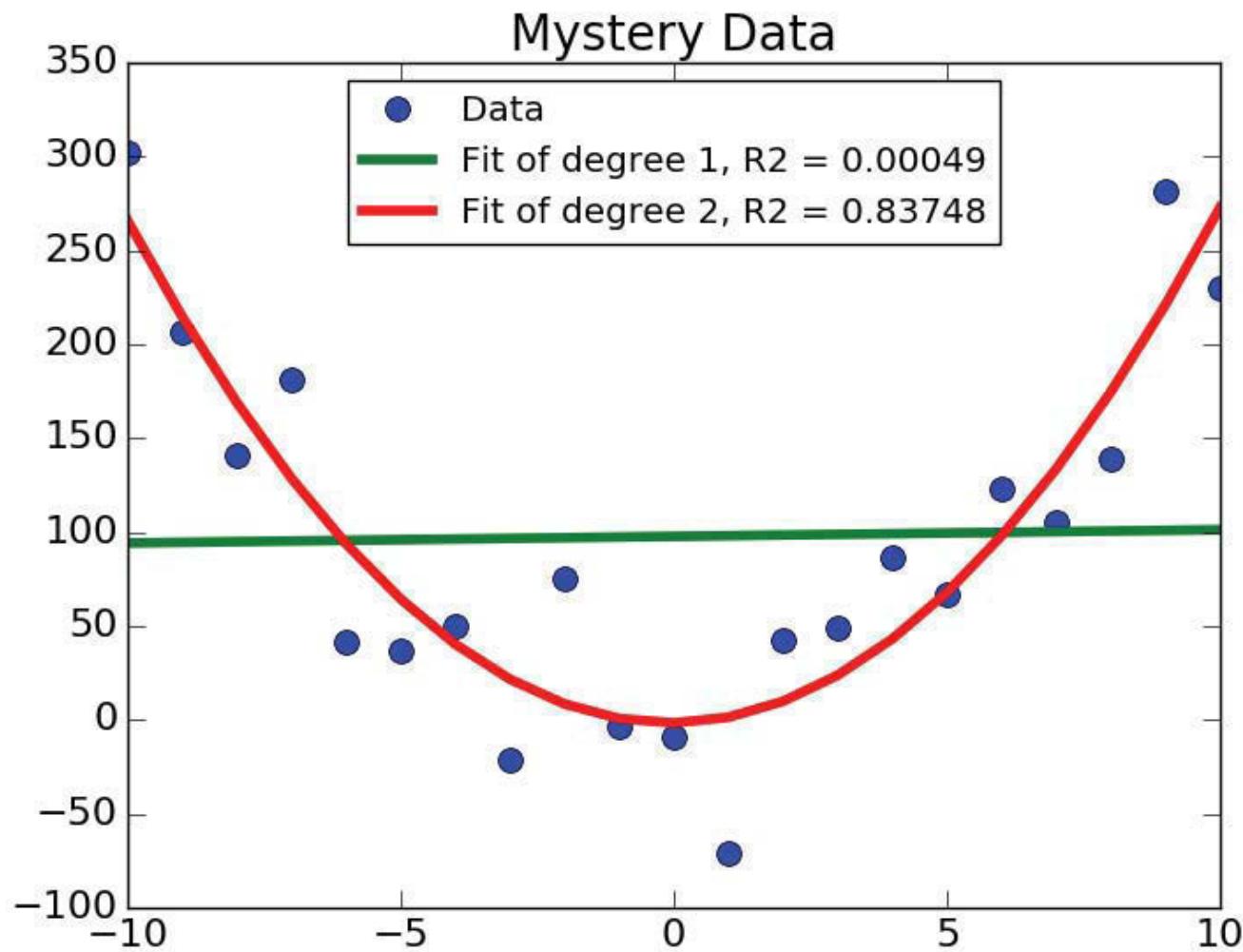
- By comparing the estimation errors (the numerator) with the variability of the original values (the denominator), R² is intended to capture the proportion of variability in a data set that is accounted for by the statistical model provided by the fit
- Always between 0 and 1 when fit generated by a linear regression and tested on training data
 - If R² = 1, the model explains all of the variability in the data.
 - If R² = 0, there is no relationship between the values predicted by the model and the actual data.
 - If R² = 0.5, the model explains half the variability in the data.

Testing Goodness of Fits

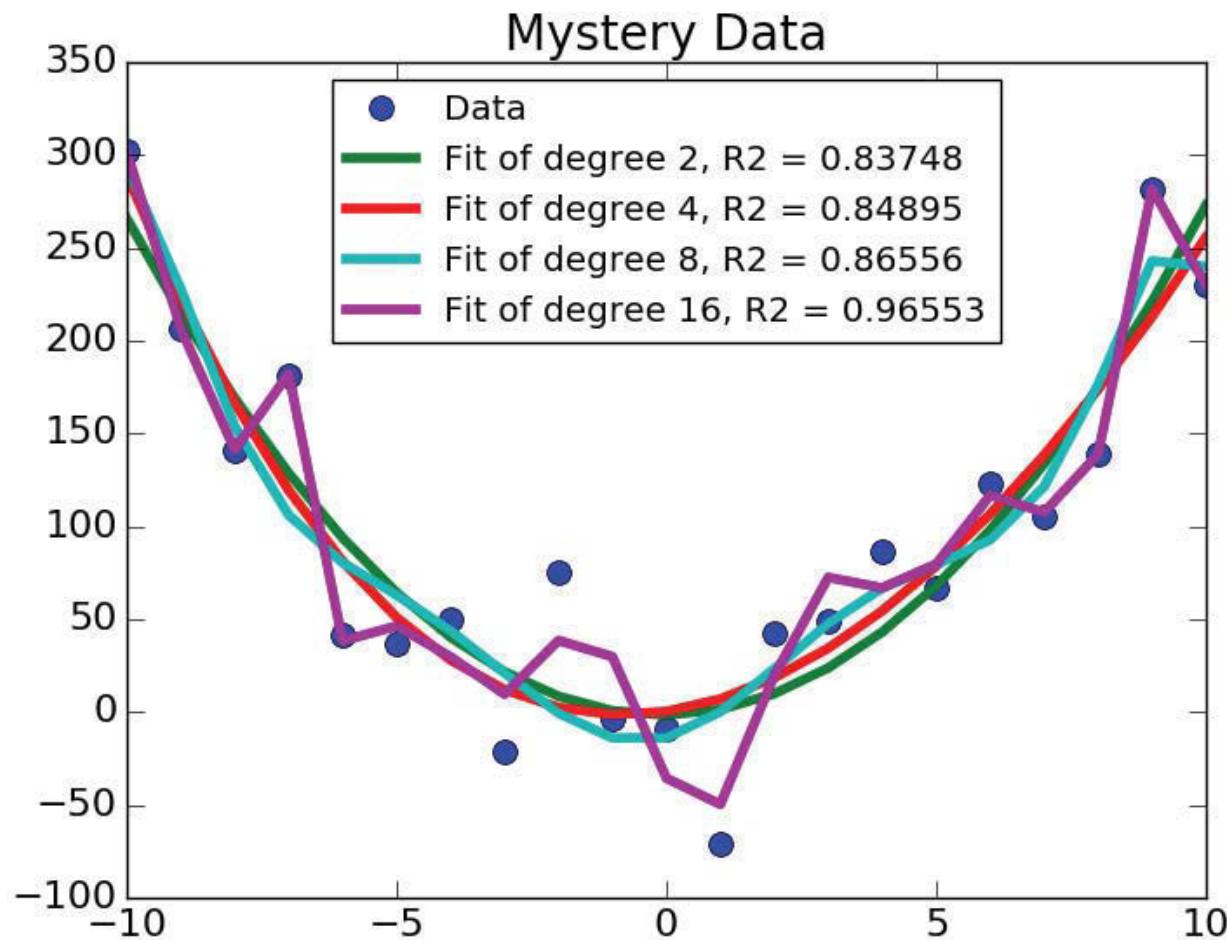
```
def genFits(xVals, yVals, degrees):
    models = []
    for d in degrees:
        model = pylab.polyfit(xVals, yVals, d)
        models.append(model)
    return models

def testFits(models, degrees, xVals, yVals, title):
    pylab.plot(xVals, yVals, 'o', label = 'Data')
    for i in range(len(models)):
        estYVals = pylab.polyval(models[i], xVals)
        error = rSquared(yVals, estYVals)
        pylab.plot(xVals, estYVals,
                   label = 'Fit of degree ' \
                   + str(degrees[i]) \
                   + ', R2 = ' + str(round(error, 5)))
    pylab.legend(loc = 'best')
    pylab.title(title)
```

How Well Fits Explain Variance



Can We Get a Tighter Fit?



MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Understanding Experimental Data, cont.

Eric Grimson

MIT Department Of Electrical Engineering and
Computer Science

Remember our goal

- Want to find a model that fits experimental data well
- Model will then allow us to explain phenomena, and to make predictions about behavior in new settings
- Know that data is unlikely to be perfect, so have to account for uncertainty in measurements or observations
- Sometimes have theoretical knowledge of structure of model, but not always
 - In latter case, want to try to find best model from class of options

Solving for Least Squares (Recap)

$$\sum_{i=0}^{\text{len}(\text{observed})-1} (\text{observed}[i] - \text{predicted}[i])^2$$

- Given observed data, and model prediction of expected values, can measure goodness of fit of model to observation using sum-of-squared-differences (or mean-squared-error)
- Want to find best model for predicting values
- Predicted values often come from mathematical expression, with set of parameters that can vary – typically a polynomial expression
- Use linear regression to find best model that minimizes difference – for polynomial model, this include coefficients, and may include order of polynomial

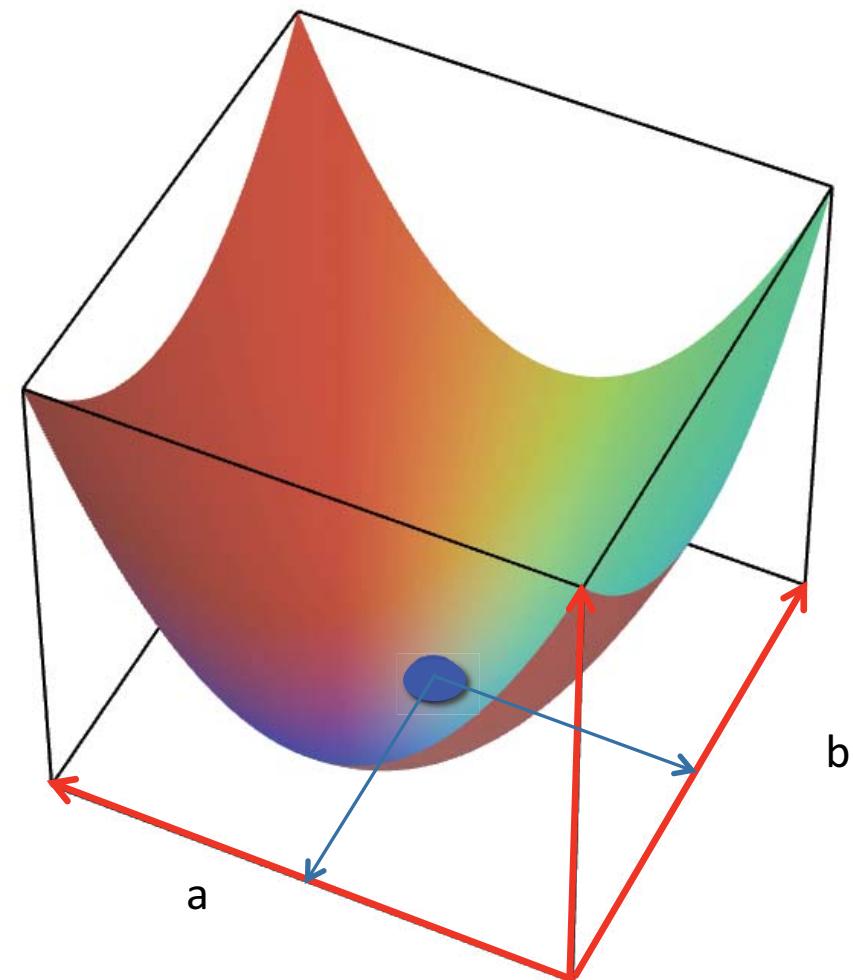
Solving for Least Squares (Recap)

$$\sum_{i=0}^{\text{len}(\text{observed})-1} (\text{observed}[i] - \text{predicted}[i])^2$$

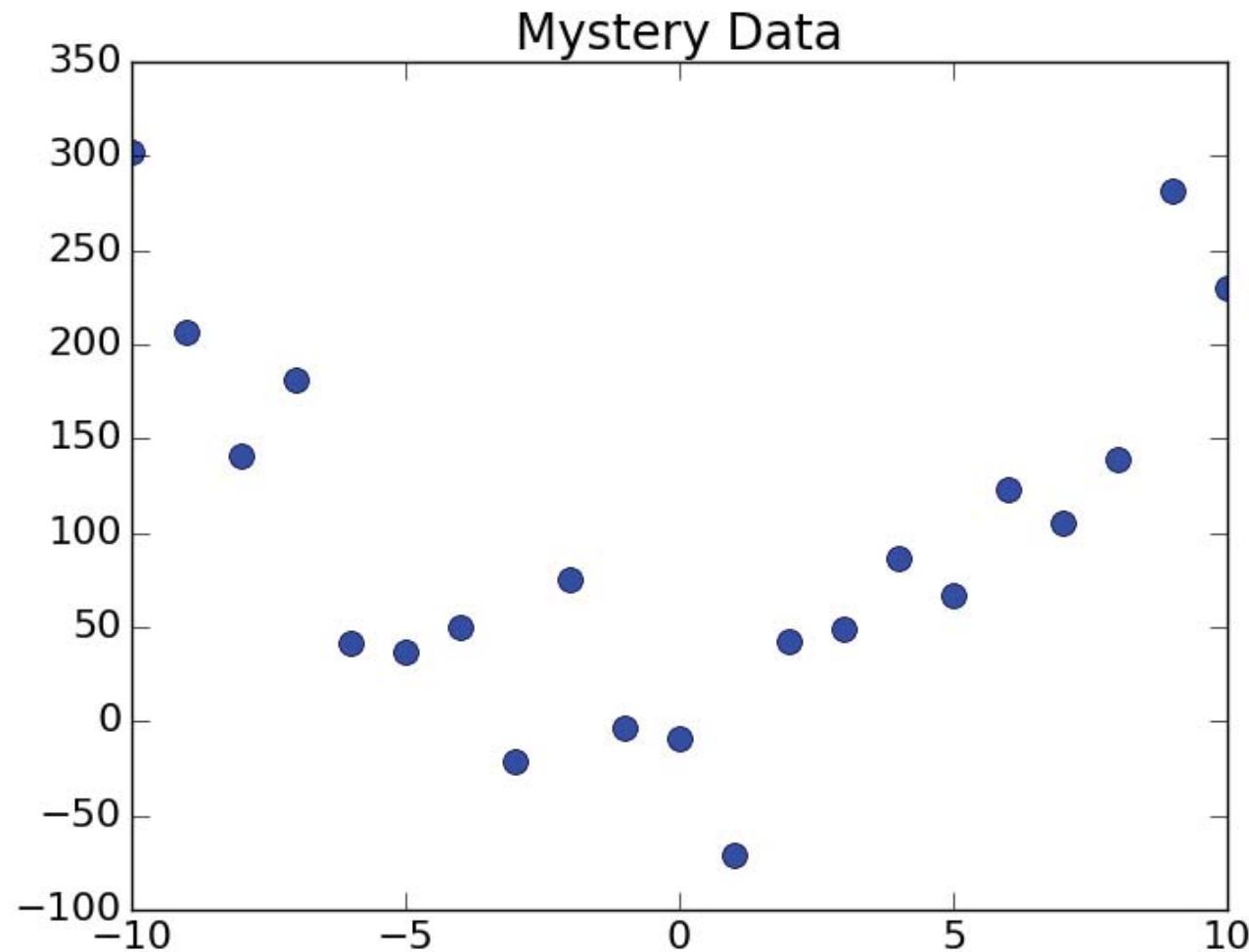
- Simple example:
 - Use a degree-one polynomial, $y = ax+b$, as model of our data (we want best fitting line)
- Find values of a and b such that when we use the polynomial to predict y values for all of the x values in our experiment, the squared difference of these values and the corresponding observed values is minimized
- A **linear regression** problem

Finding the best curve (simplest case)

- The set of all possible lines can be represented by a point in a-b space
- Imagine a surface in this space, where height of the surface is the value of the objective function
- Starting at any point on the surface, walk “downhill”, until you reach the “bottom”
- Corresponding point is best line to fit to data
- Can generalize to higher order models



Another Experiment (Recap)

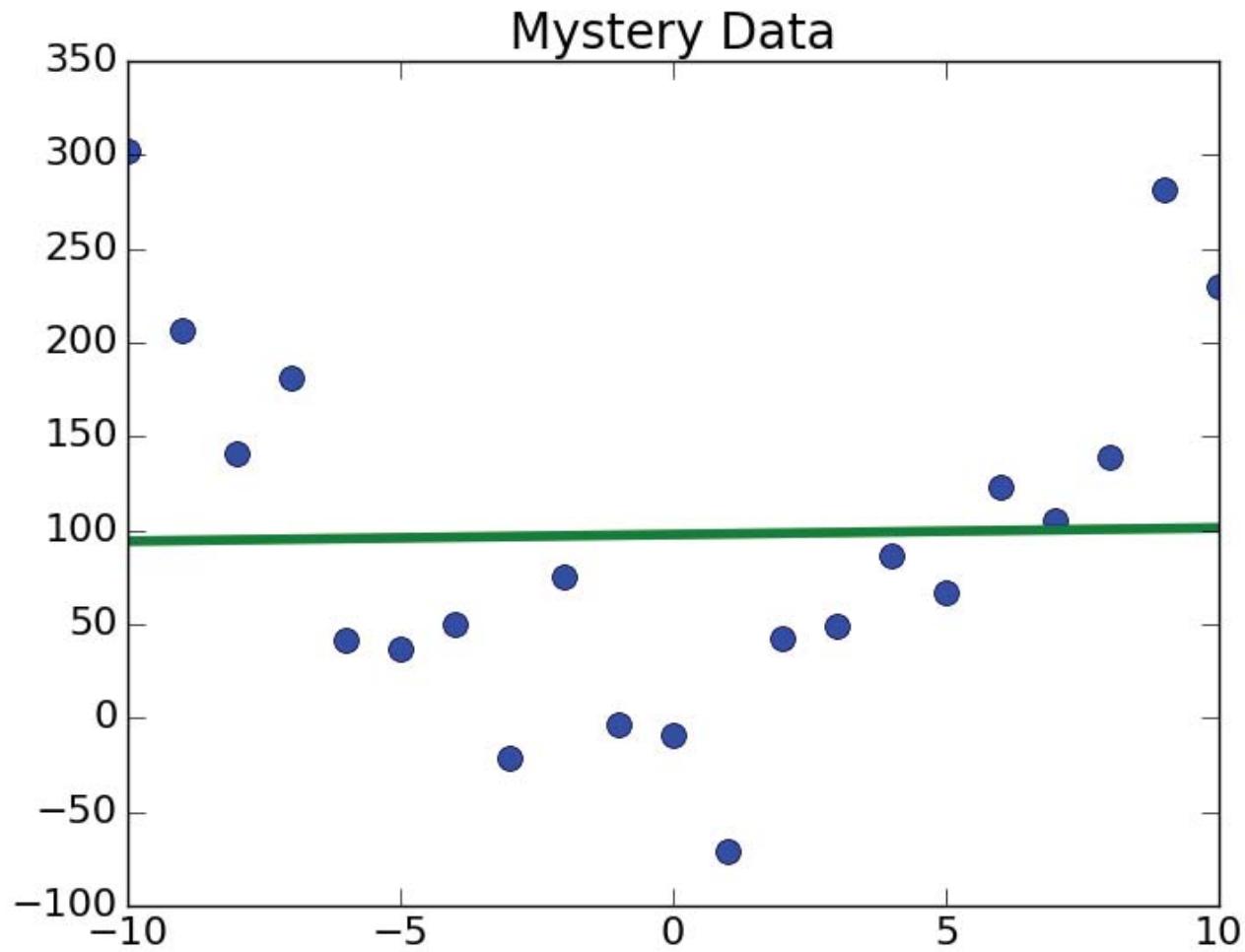


Fit a Line

```
model1 = pylab.polyfit(xVals, yVals, 1)
pylab.plot(xVals, pylab.polyval(model1, xVals),
            'r--', label = 'Linear Model')
```

- Remember that `pylab.polyfit` will find parameters of best fitting polynomial of described order
 - In this case (with argument $n = 1$), find the values of a and b , such that $y = ax + b$ best matches the observed $yVals$
- Remember that `pylab.polyval` will generate predicted $yVals$ given parameters of model

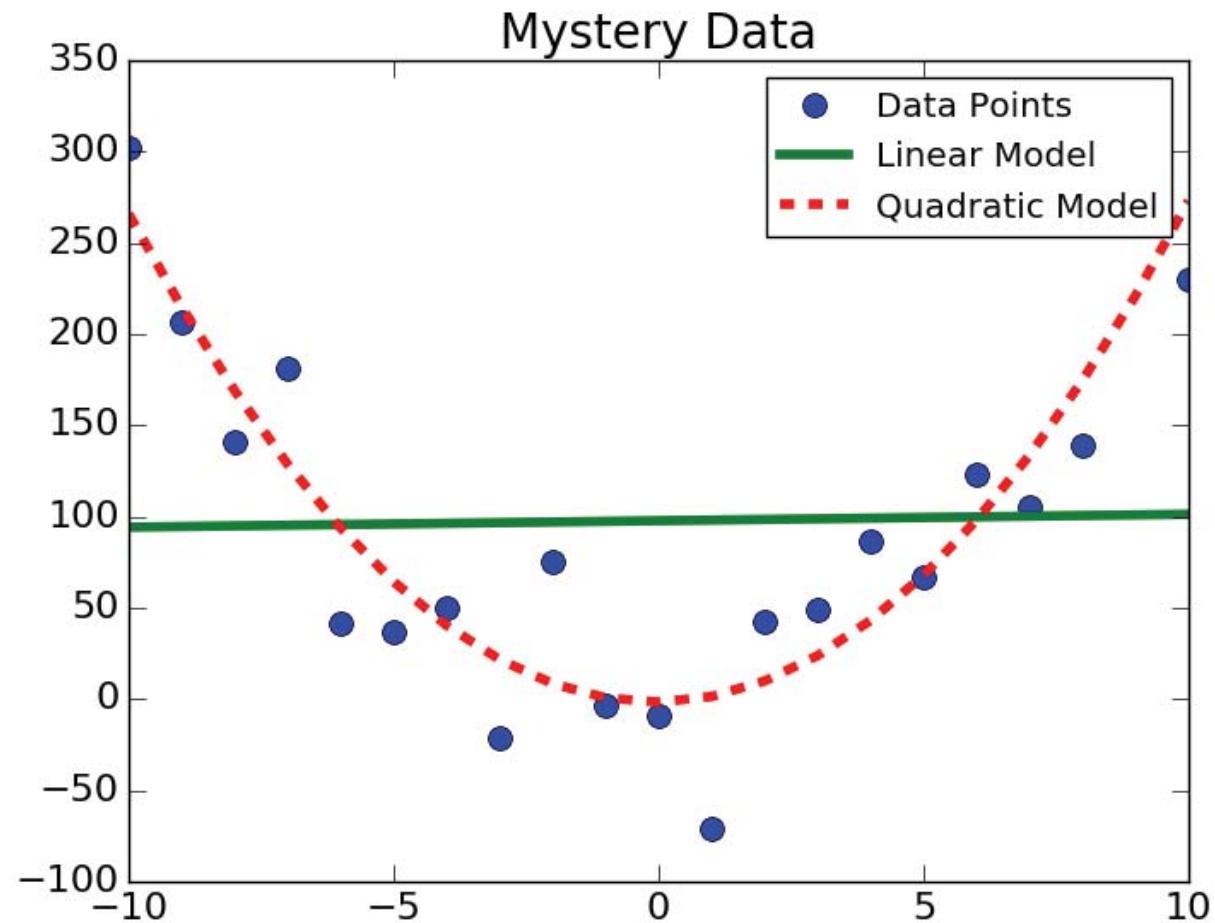
Fit a Line



Let's Try a Higher-degree Model

```
model2 = pylab.polyfit(xVals, yVals, 2)
pylab.plot(xVals, pylab.polyval(model2, xVals),
           'r--', label = 'Quadratic Model')
```

Quadratic Appears to be a Better Fit



Can We Get a Tighter Fit?

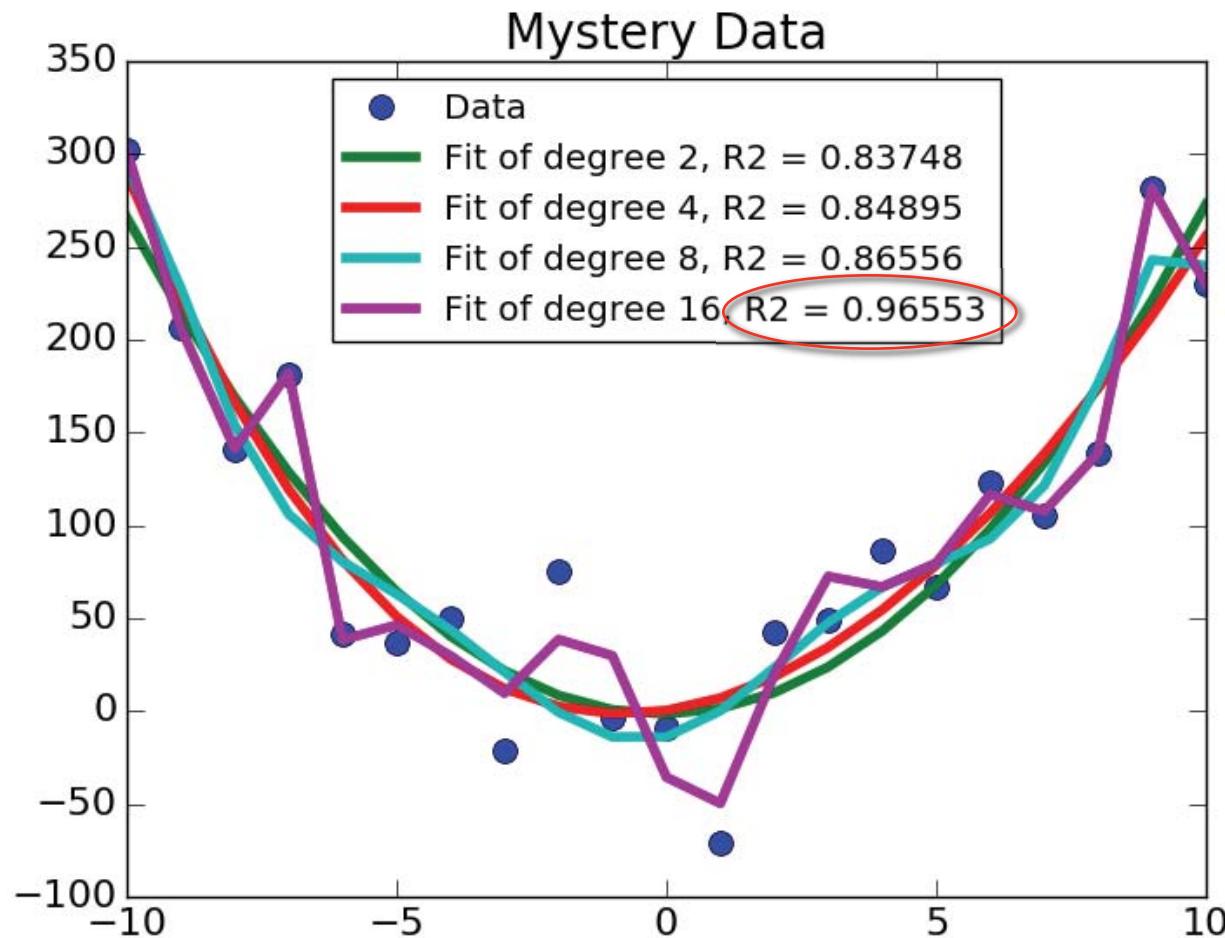
- What if we try fitting higher order polynomials to the data?
 - Does this give us a better fit?
- How would we measure that?
 - In absence of other information (e.g., theoretical insights into order of model), R^2 (**coefficient of determination**) gives us decent measure of the tightness of the model fit
 - In principle, a model with a higher R^2 value is a “better” fit

$$R^2 = 1 - \frac{\sum_i (y_i - p_i)^2}{\sum_i (y_i - \mu)^2}$$

← Error in estimates
← Variability in measured data

y_i are measured values
 p_i are predicted values
 μ is mean of measured values

Can We Get a Tighter Fit?

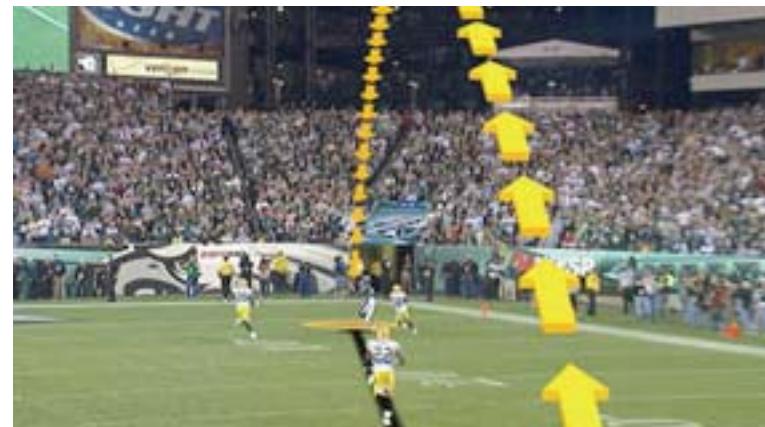
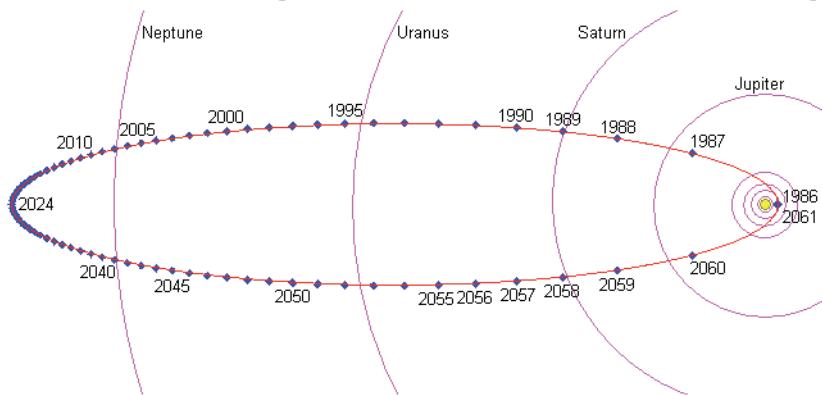


Why We Build Models

- Looks like an order 16 fit is really good – so should we just use this as our model?
 - To answer, need to ask – why build models in first place?
- Help us understand process that generated the data
 - E.g., the properties of a particular linear spring
- Help us make predictions about out-of-sample data
 - E.g., predict the displacement of a spring when a force is applied to it
 - E.g., predict the effect of treatment on a patient
 - E.g., predict the outcome of an election
- A good model helps us do both of these things

Motivation for Mystery Data – Parabola

- Trajectory of a particle under the influence of a uniform gravitational field (e.g. Halley's Comet)
- Position of center of mass of a football pass
- Design of a load-bearing arch



Images of particle trajectory, load-bearing arch, football pass center of mass diagram © sources unknown. All rights reserved.
This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.

How Mystery Data Was Generated

```
def genNoisyParabolicData(a, b, c, xVals, fName):
    yVals = []
    for x in xVals:
        theoreticalVal = a*x**2 + b*x + c
        yVals.append(theoreticalVal + random.gauss(0, 35))
    f = open(fName, 'w')
    f.write('x      y\n')
    for i in range(len(yVals)):
        f.write(str(yVals[i]) + ' ' + str(xVals[i]) + '\n')
    f.close()

#parameters for generating data
xVals = range(-10, 11, 1)
a, b, c = 3, 0, 0
genNoisyParabolicData(a, b, c, xVals, 'Mystery Data.txt')
```

If data was generated by quadratic, why was 16th order polynomial the “best” fit?

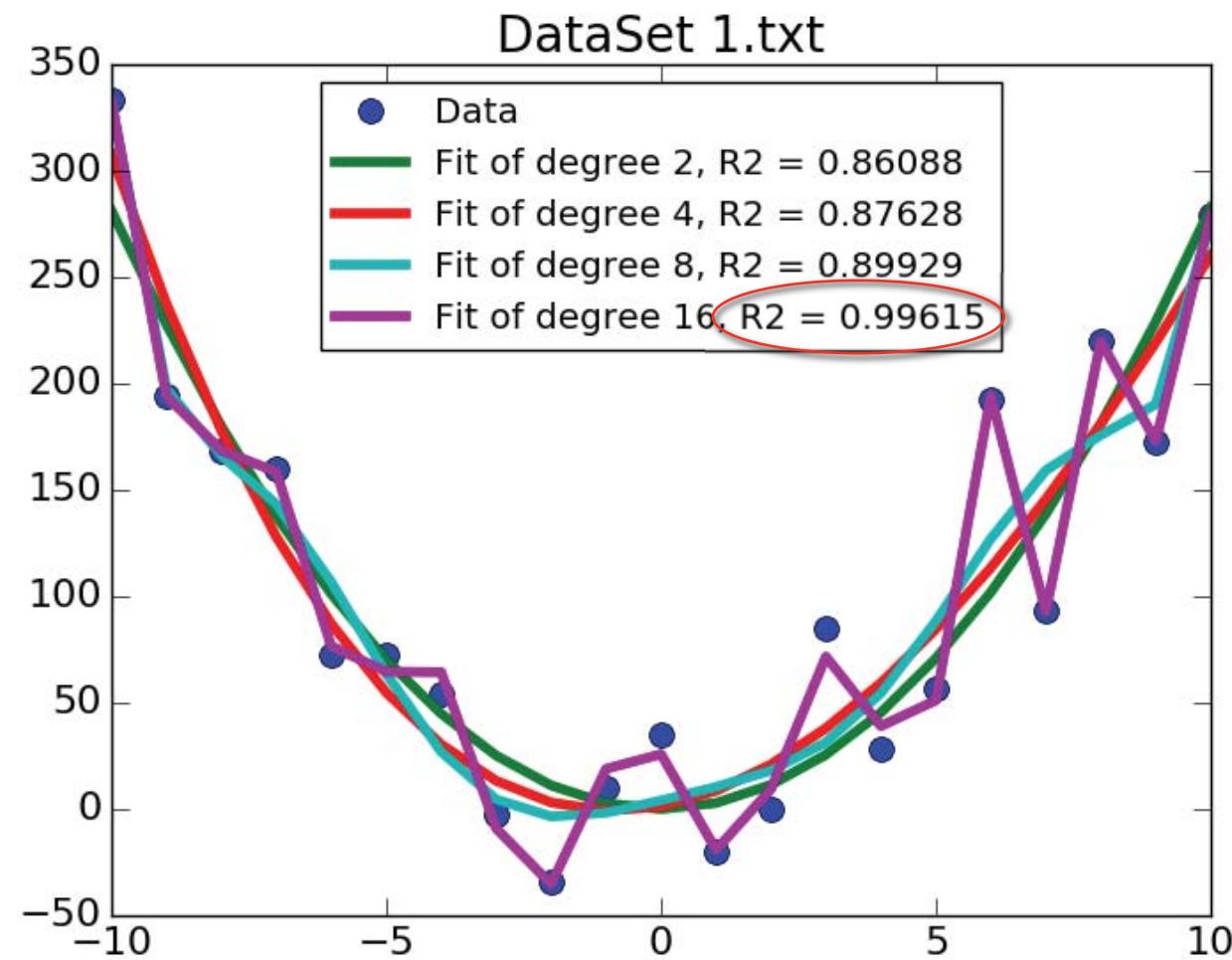
Let's Look at Two Data Sets

```
degrees = (2, 4, 8, 16)
```

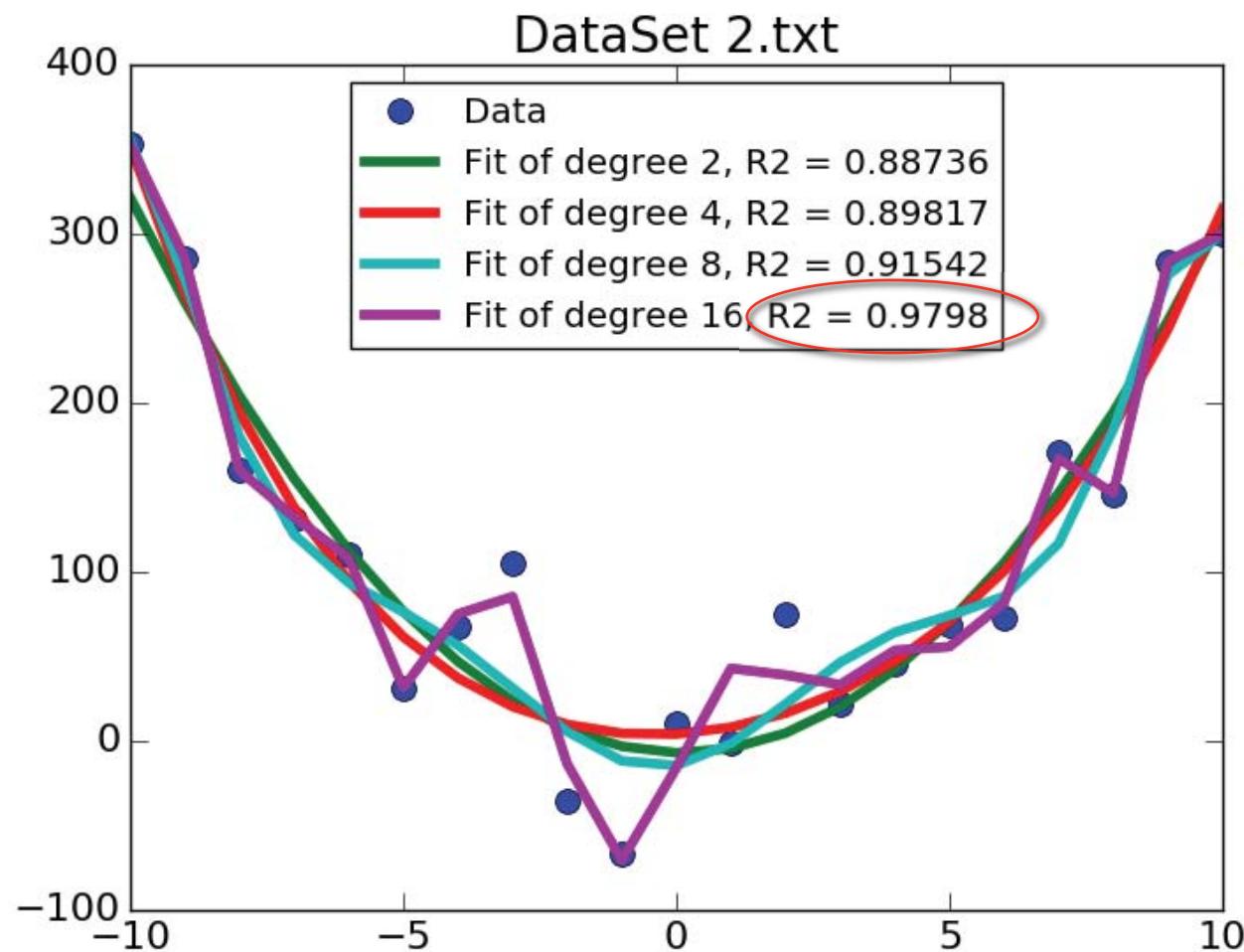
```
random.seed(0)
xVals1, yVals1 = getData('Dataset 1.txt')
models1 = genFits(xVals1, yVals1, degrees)
testFits(models1, degrees, xVals1, yVals1,
          'DataSet 1.txt')
```

```
pylab.figure()
xVals2, yVals2 = getData('Dataset 2.txt')
models2 = genFits(xVals2, yVals2, degrees)
testFits(models2, degrees, xVals2, yVals2,
          'DataSet 2.txt')
```

Fits for Dataset 1



Fits for Dataset 2



Hence Degree 16 Is Tightest Fit

- “Best” fitting model is still order 16 polynomial for both data sets, **but** we know data was generated using an order 2 polynomial?
- What we are seeing comes from training error
 - How well the model performs on the data from which it was learned
 - Small training error a necessary condition for a great model, **but not a sufficient one**
- We want model to work well on other data generated by the same process
 - Measurements for other weights on the spring
 - Positions of comets under different forces
 - Voters other than those surveyed
- In other words, the model needs to generalize

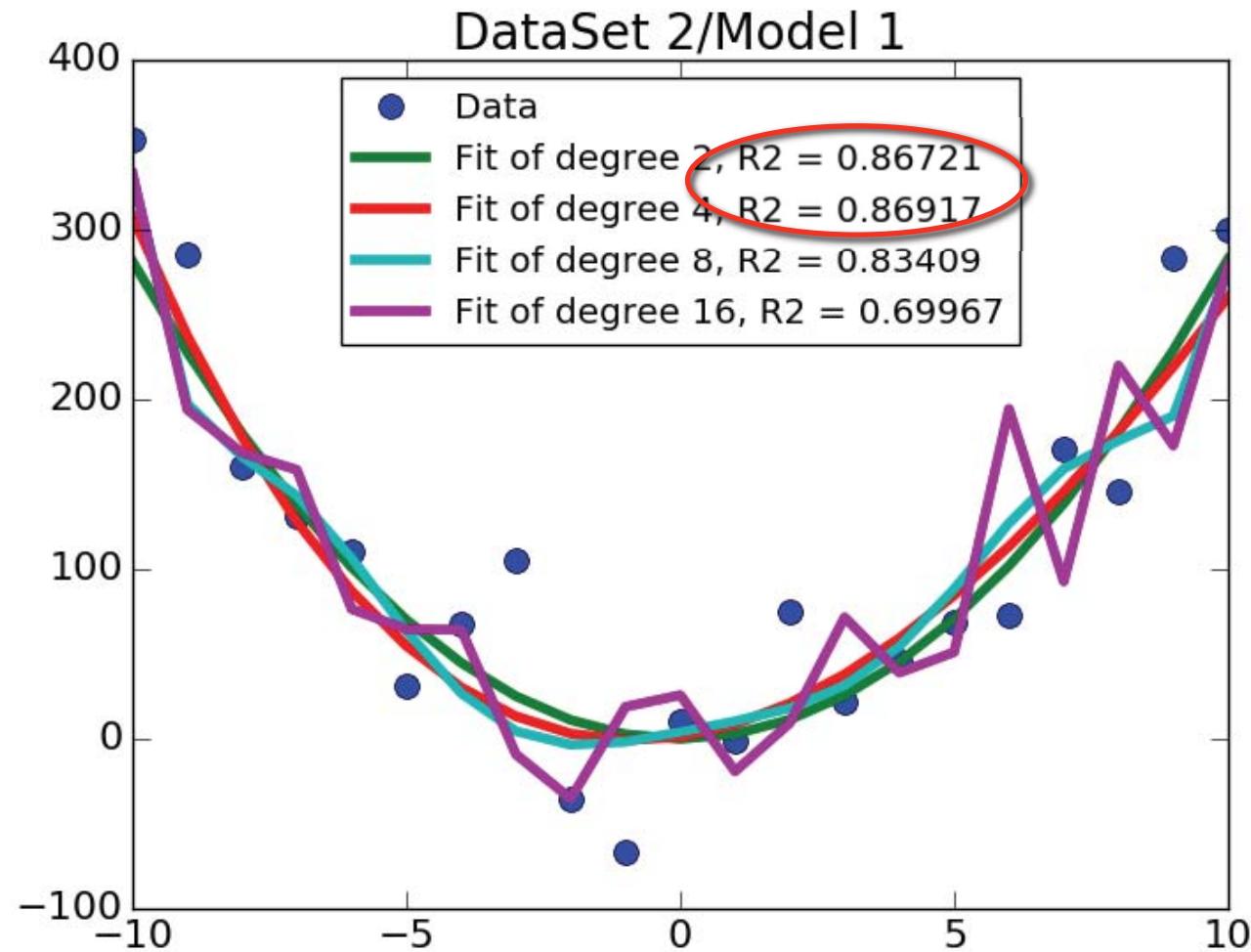
Cross Validate

- Generate models using one dataset, and then test them on another dataset
 - Use models for Dataset 1 to predict points for Dataset 2
 - Use models for Dataset 2 to predict points for Dataset 1
- Expect testing error to be larger than training error
- A better indication of generalizability than training error

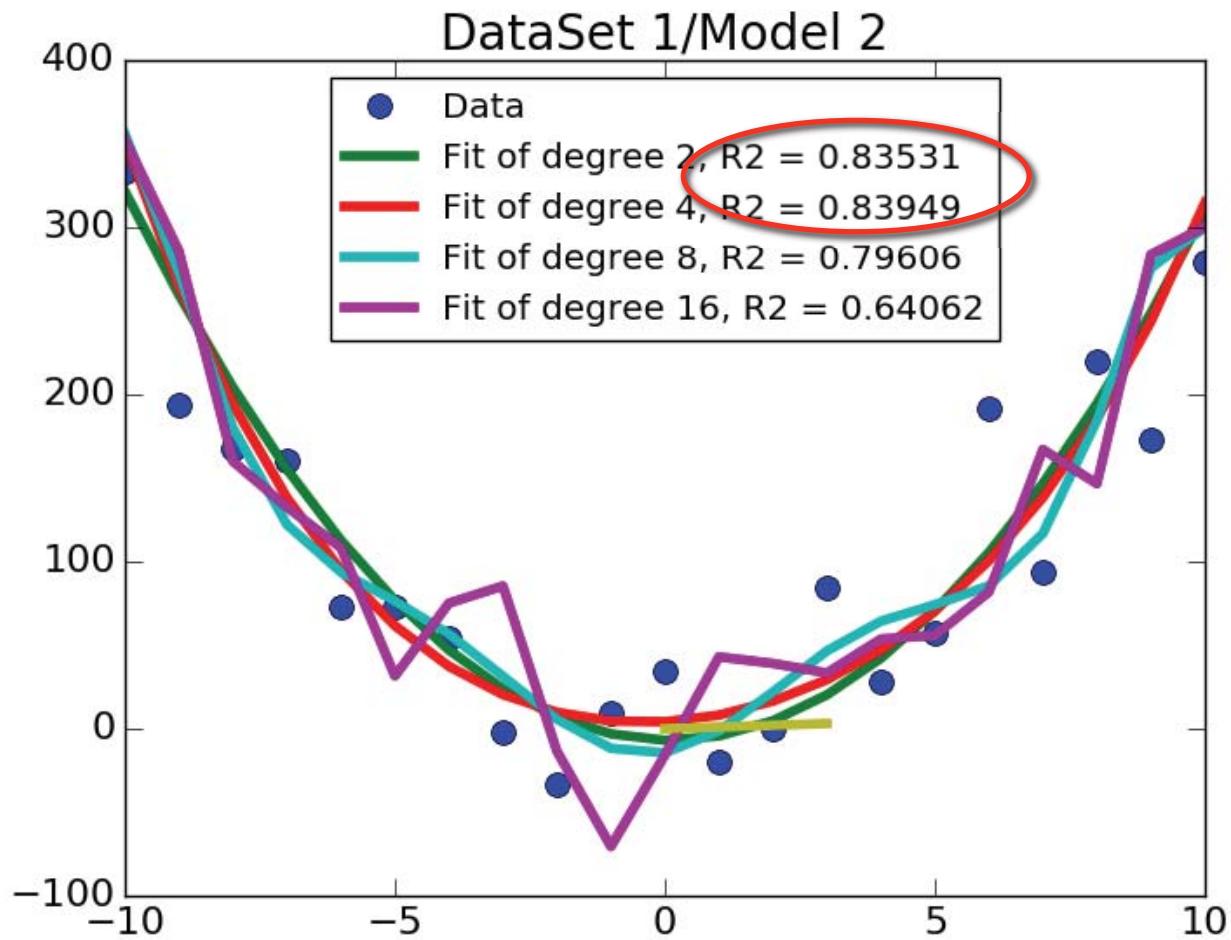
Test Code

```
pylab.figure()  
testFits(models1, degrees, xVals2, yVals2,  
          'DataSet 2/Model 1')  
pylab.figure()  
testFits(models2, degrees, xVals1, yVals1,  
          'DataSet 1/Model 2')
```

Train on Dataset 1, Test on Dataset 2



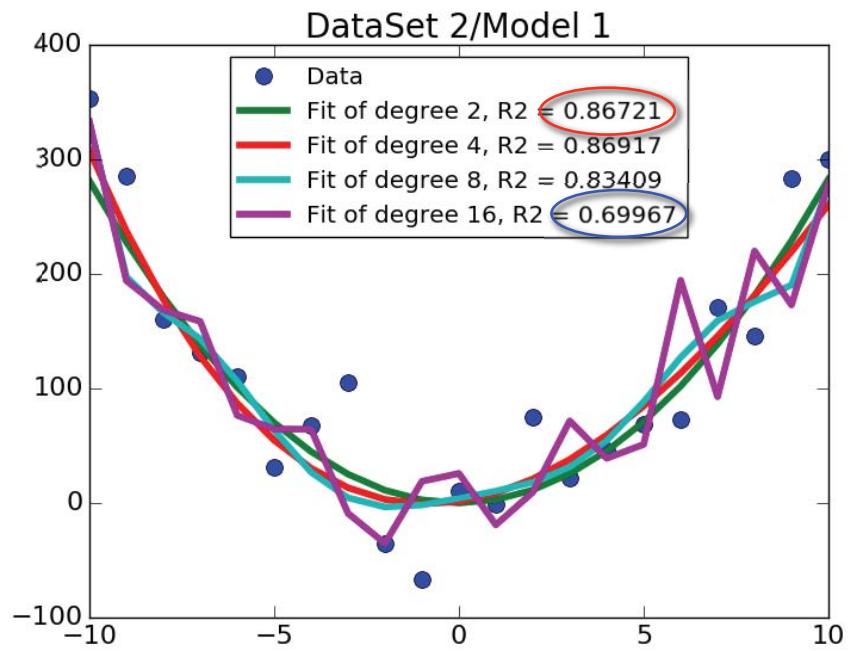
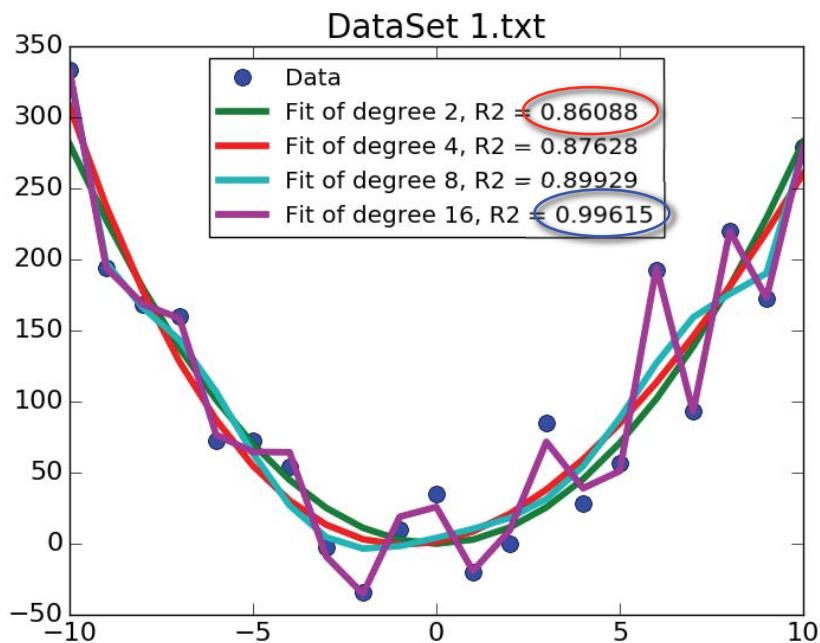
Train on Dataset 2, Test on Dataset 1



Cross Validation

- Now can see that based on R^2 numbers, best model is more likely to be 2nd order or 4th order polynomial (we know it is actually 2nd order, and difference in R^2 values is pretty small), but certainly not 16th order
- Example of **over fitting** to the data
- Can see that if we only fit model to training data, we may not detect that model is too complex; but training on one data set, then testing on a second helps expose this problem

Training and Testing Errors

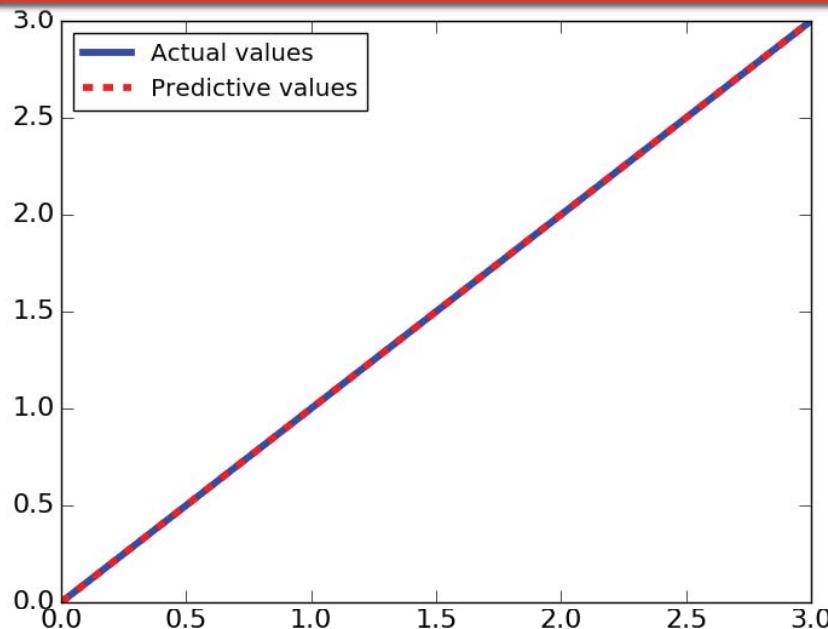


Increasing the Complexity

- Why do we get a “better” fit on training data with higher order model, but then do less well on handling new data?
- What happens when we increase order of polynomial during training?
 - Can we get a worse fit to training data?
- If extra term is useless, coefficient will merely be zero
- But if data is noisy, can fit the noise rather than the underlying pattern in the data
 - May lead to a “better” R^2 value, but not really a “better” fit

Fitting a Quadratic to a Perfect Line

```
xVals = (0,1,2,3)
yVals = xVals
pylab.plot(xVals, yVals, label = 'Actual values')
a,b,c = pylab.polyfit(xVals, yVals, 2)
print('a =', round(a, 4), 'b =', round(b, 4),
      'c =', round(c, 4))
estYVals = pylab.polyval((a,b,c), xVals)
pylab.plot(xVals, estYVals, 'r--', label = 'Predictive values')
print('R-squared = ', rSquared(yVals, estYVals))
```



$$y = ax^2 + bx + c$$

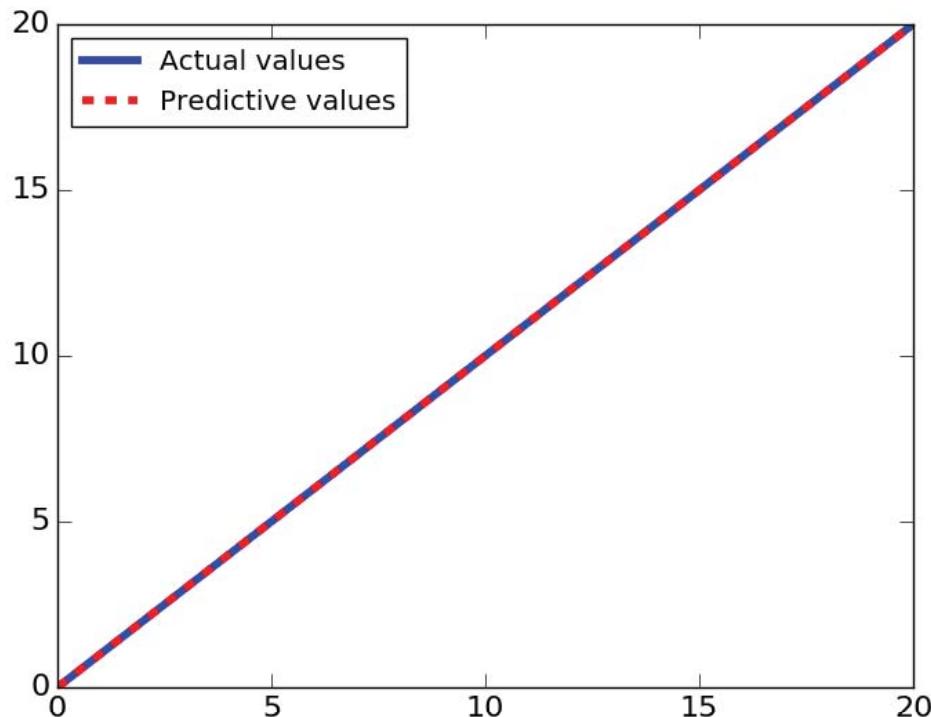
$$y = 0x^2 + 1x + 0$$

$$y = x$$

R-squared = 1.0

Predict Another Point Using Same Model

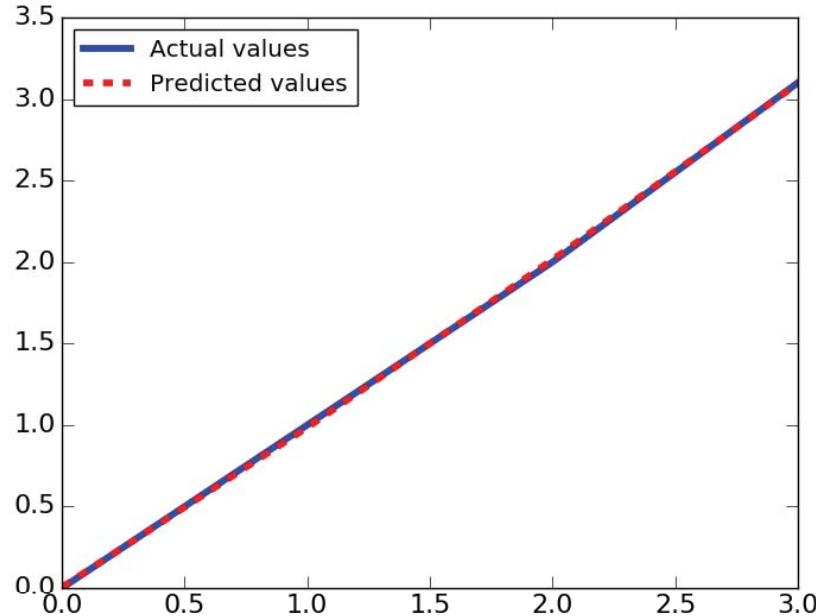
```
xVals = xVals + (20,)  
yVals = xVals  
pylab.plot(xVals, yVals, label = 'Actual values')  
estYVals = pylab.polyval((a,b,c), xVals)  
pylab.plot(xVals, estYVals, 'r--', label = 'Predictive values')  
print('R-squared = ', rSquared(yVals, estYVals))
```



R-squared = 1.0

Simulate a Small Measurement Error

```
xVals = (0,1,2,3)
yVals = (0,1,2,3.1)
pylab.plot(xVals, yVals, label = 'Actual values')
model = pylab.polyfit(xVals, yVals, 2)
print(model)
estYVals = pylab.polyval(model, xVals)
pylab.plot(xVals, estYVals, 'r--', label = 'Predicted values')
print('R-squared = ', rSquared(yVals, estYVals))
```



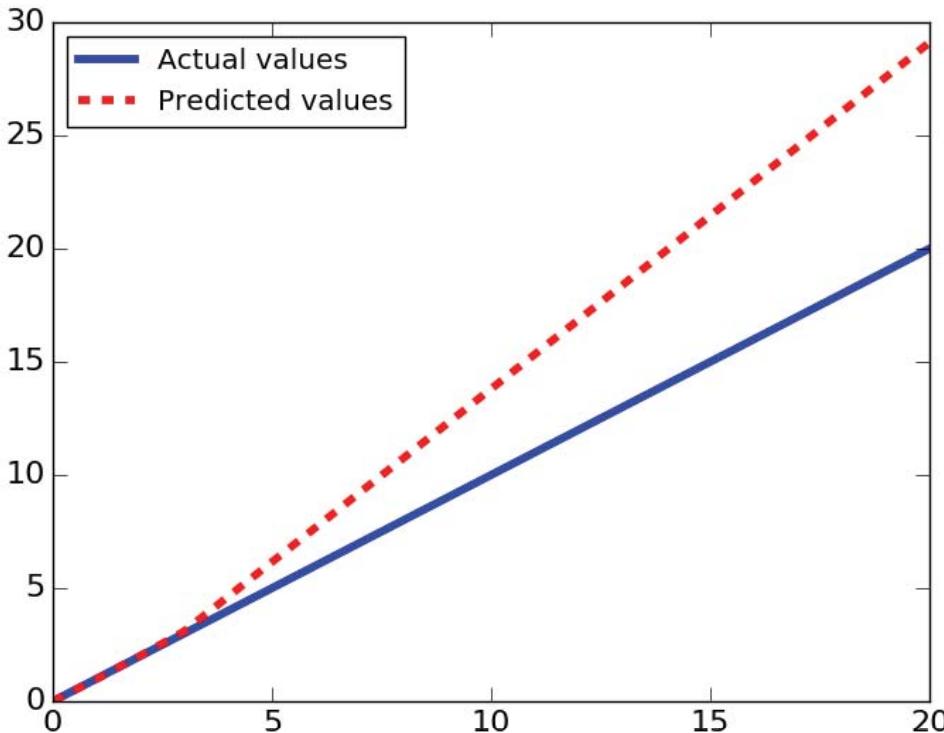
$$y = ax^2 + bx + c$$

$$y = .025x^2 + .955x + .005$$

$$R\text{-squared} = 0.9994$$

Predict Another Point Using Same Model

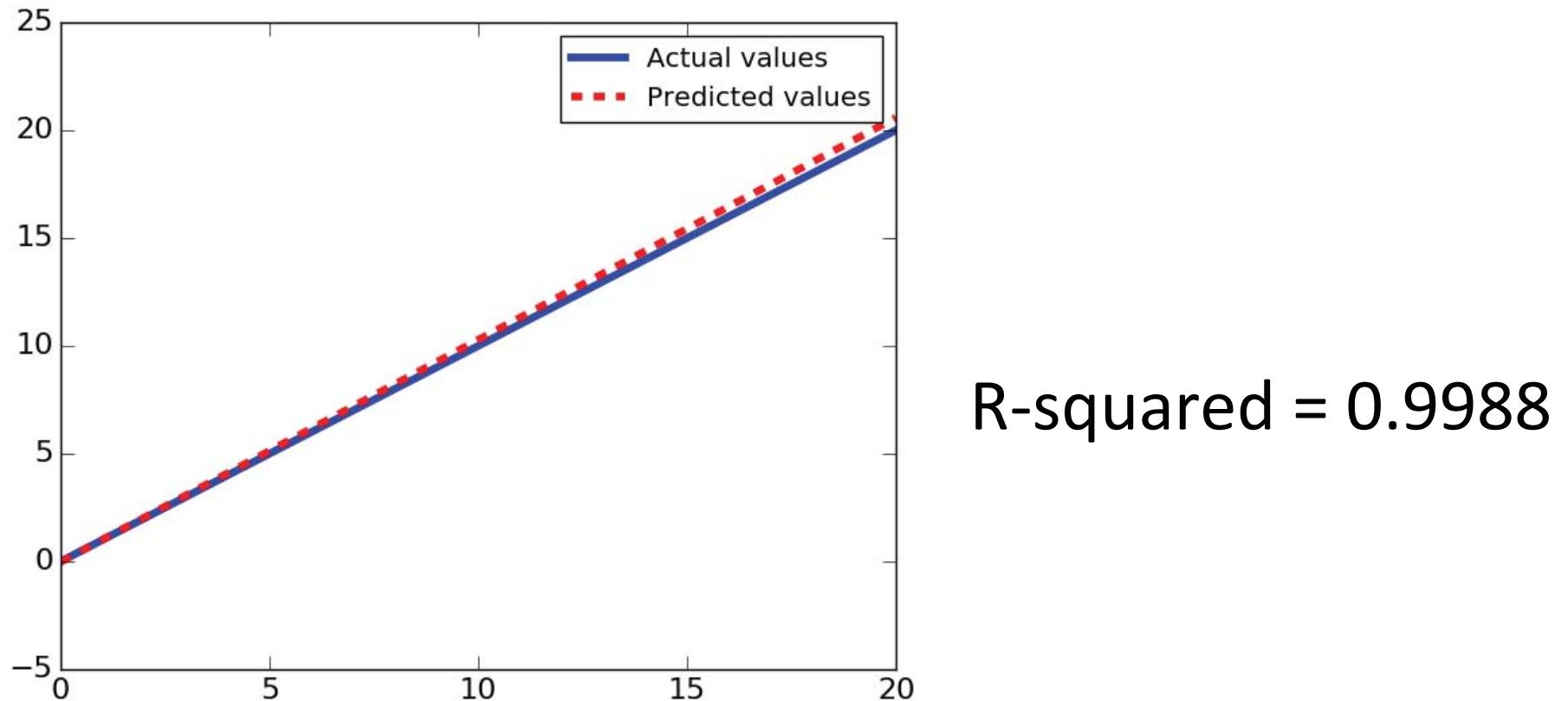
```
xVals = xVals + (20,)  
yVals = xVals  
estYVals = pylab.polyval(model, xVals)  
print('R-squared = ', rSquared(yVals, estYVals))  
pylab.figure()  
pylab.plot(xVals, estYVals)
```



R-squared = 0.7026

Suppose We Had Used a First-degree Fit

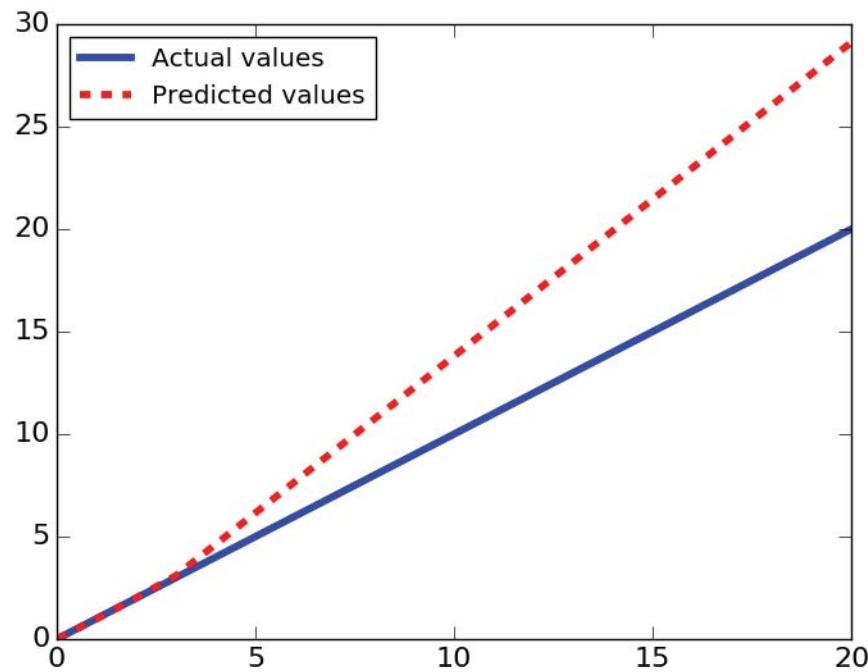
- model = pylab.polyfit(xVals, yVals, 1)



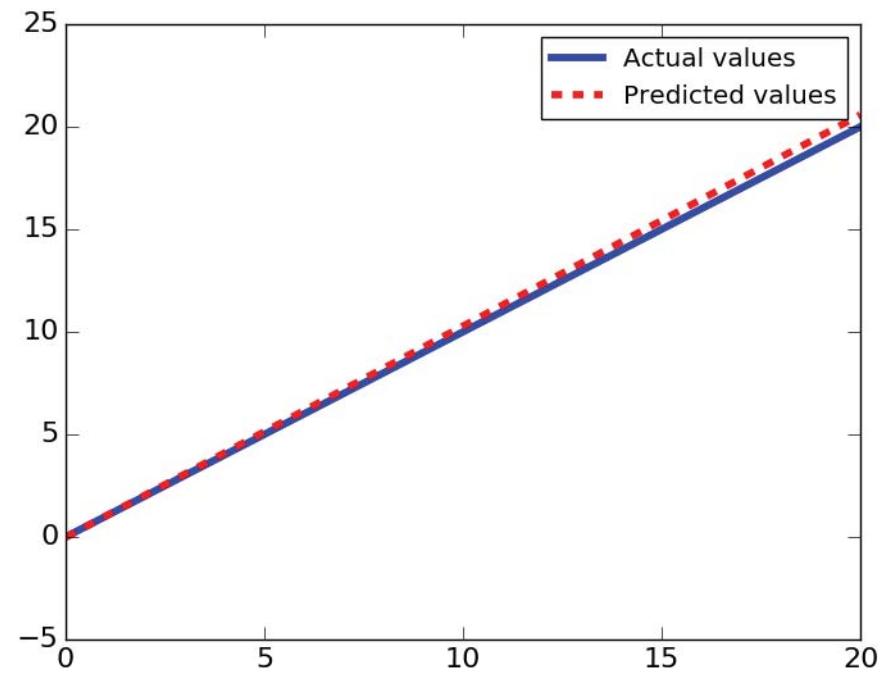
Comparing first and second degree fits

- Predictive ability of first order fit much better than second order fit

Degree 2 polynomial



Degree 1 polynomial

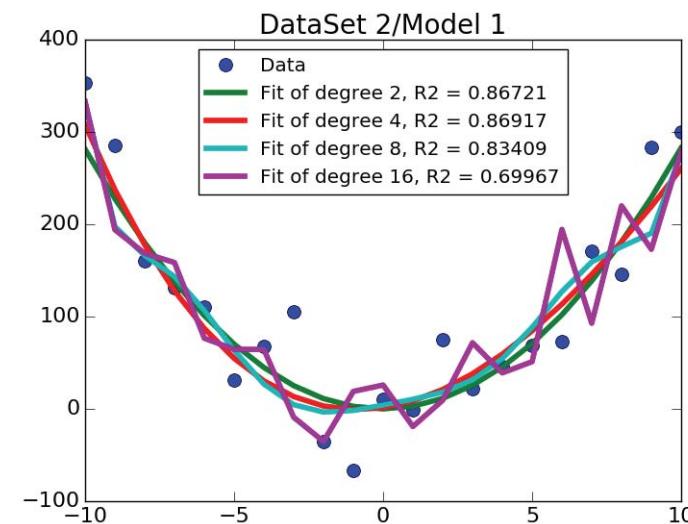
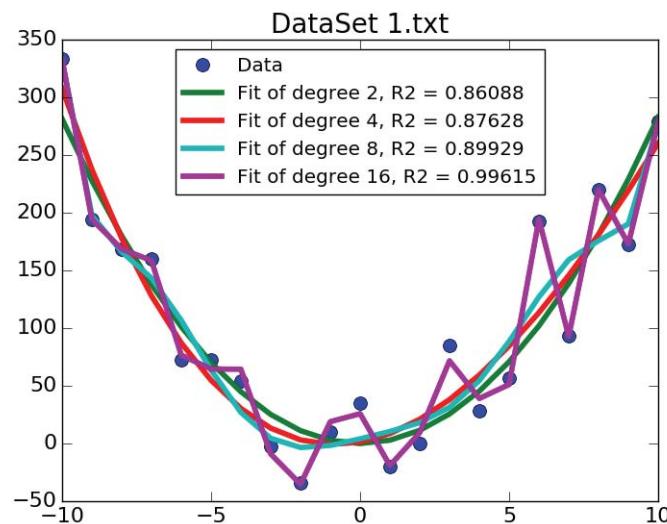


The Take Home Message

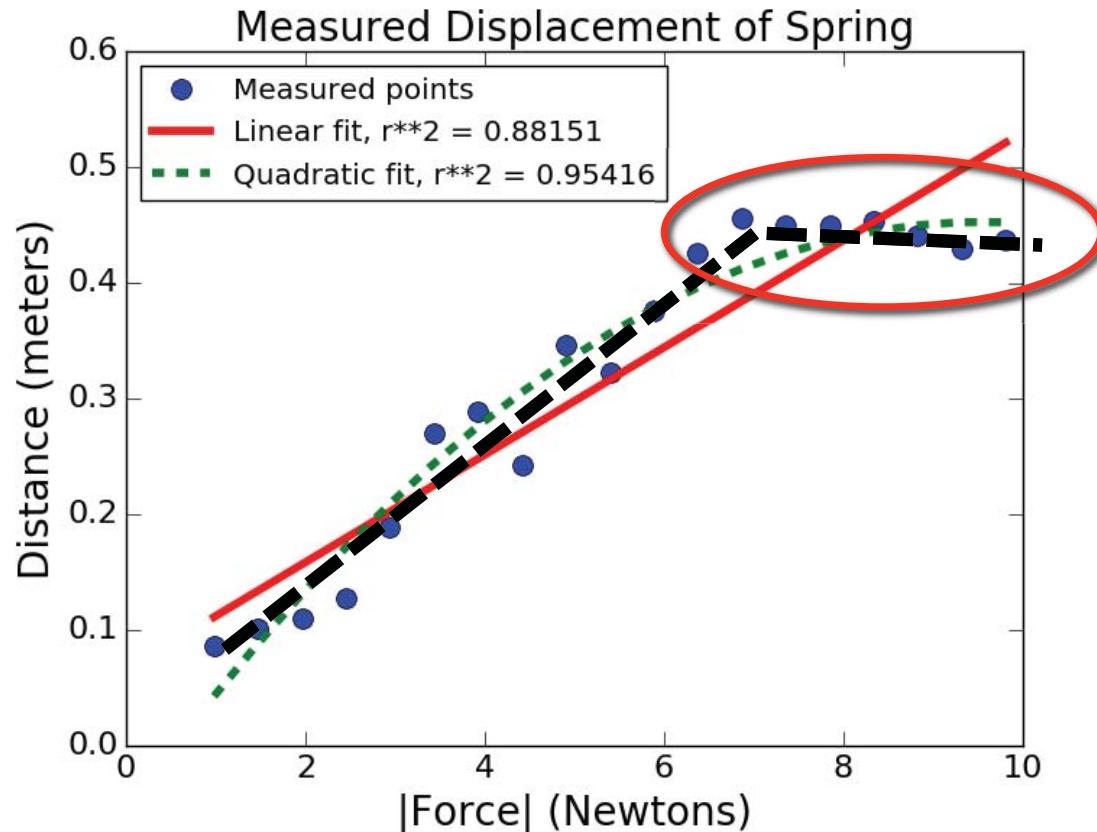
- Choosing an overly-complex model leads to **overfitting** to the training data
- Increases the risk of a model that works poorly on data not included in the training set
- On the other hand choosing an insufficiently complex model has other problems
 - As we saw when we fit a line to data that was basically parabolic
 - “**Everything should be made as simple as possible, but not simpler**” – Albert Einstein

Balancing Fit with Complexity

- In absence of theory predicting order of model, can engage in a search process
 - Fit a low order model to training data
 - Test on new data and record R^2 value
 - Increase order of model and repeat
 - Continue until fit on test data begins to decline



Returning to Where We Started



Quadratic fit tighter

But remember Hooke

Unless we believe
theory is wrong, that
should guide us

Model holds until
reach elastic limit of
spring

Should probably fit different models to different segments of data

Can visualize as search process – find best place to break into two parts, such that both linear segments have high R^2 fits

Suppose We Don't Have a Solid Theory

- Use cross-validation results to guide the choice of model complexity
- If dataset small, use leave-one-out cross validation
- If dataset large enough, use k-fold cross validation or repeated-random-sampling validation

Leave-one-out Cross Validation

Let D be the original data set

```
testResults = []
for i in range(len(D)):
    training = D[:].pop(i)
    model = buildModel(training)
    testResults.append(test(model, D[i]))
```

Average testResults

k-fold very similar

Applies when we have large amount of data

D partitioned into k equal size sets

Model trained on k-1 sets, and tested on remaining set

Repeated Random Sampling

Let D be the original data set
n be the number of random samples
usually n between 20% and 50%
k be number of trials

```
testResults = []
for i in range(k)
    randomly select n elements for testSet,
        keep rest for training
    model = buildModel(training)
    testResults.append(test(model, testSet))
```

Average testResults

An Example, Temperature By Year

- Task: Model how the mean daily high temperature in the U.S. varied from 1961 through 2015
- Get means for each year and plot them
- Randomly divide data in half n times
 - For each dimensionality to be tried
 - Train on one half of data
 - Test on other half
 - Record r-squared on test data
- Report mean r-squared for each dimensionality

A Boring Class

```
class tempDatum(object):
    def __init__(self, s):
        info = s.split(',')
        self.high = float(info[1])
        self.year = int(info[2][0:4])
    def getHigh(self):
        return self.high
    def getYear(self):
        return self.year
```

Read Data

```
def getTempData():
    inFile = open('temperatures.csv')
    data = []
    for l in inFile:
        data.append(tempDatum(l))
    return data
```

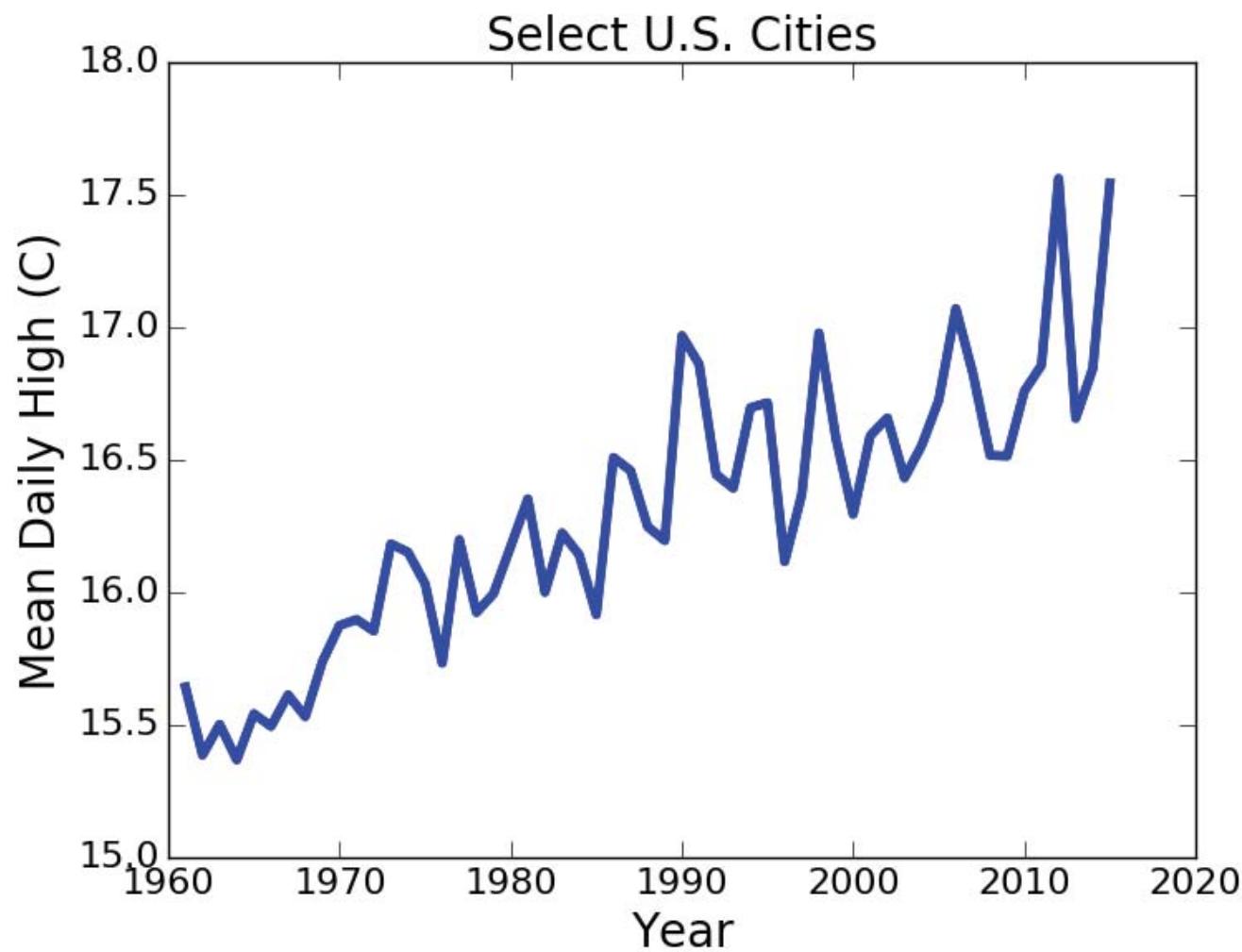
Get Means

```
def getYearlyMeans(data):
    years = {}
    for d in data:
        try:
            years[d.getYear()].append(d.getHigh())
        except:
            years[d.getYear()] = [d.getHigh()]
    for y in years:
        years[y] = sum(years[y])/len(years[y])
    return years
```

Get and Plot Data

```
data = getTempData()
years = getYearlyMeans(data)
xVals, yVals = [], []
for e in years:
    xVals.append(e)
    yVals.append(years[e])
pylab.plot(xVals, yVals)
pylab.xlabel('Year')
pylab.ylabel('Mean Daily High (C)')
pylab.title('Select U.S. Cities')
```

The Whole Data Set



Initialize Things

```
numSubsets = 10
dimensions = (1, 2, 3, 4)
rSquares = []
for d in dimensions:
    rSquares[d] = []
```

Split Data

```
def splitData(xVals, yVals):
    toTrain = random.sample(range(len(xVals)),
                           len(xVals)//2)
    trainX, trainY, testX, testY = [],[],[],[]
    for i in range(len(xVals)):
        if i in toTrain:
            trainX.append(xVals[i])
            trainY.append(yVals[i])
        else:
            testX.append(xVals[i])
            testY.append(yVals[i])
    return trainX, trainY, testX, testY
```

Train, Test, and Report

```
for f in range(numSubsets):
    trainX,trainY,testX,testY = splitData(xVals, yVals)
    for d in dimensions:
        model = pylab.polyfit(trainX, trainY, d)
        #estYVals = pylab.polyval(model, trainX)
        estYVals = pylab.polyval(model, testX)
        rSquares[d].append(rSquared(testY, estYVals))

print('Mean R-squares for test data')
for d in dimensions:
    mean = round(sum(rSquares[d])/len(rSquares[d]), 4)
    sd = round(numpy.std(rSquares[d]), 4)
    print('For dimensionality', d, 'mean =', mean,
          'Std =', sd)
```

Results

Mean R-squares for test data

For dimensionality 1 mean = 0.7535 Std = 0.0656

For dimensionality 2 mean = 0.7291 Std = 0.0744

For dimensionality 3 mean = 0.7039 Std = 0.0684

For dimensionality 4 mean = 0.7169 Std = 0.0777

- Line seems to be the winner
 - Highest average r-squared
 - Smallest deviation across trials
 - Simplest model

Why we should run multiple sets

- Note that deviations are a decimal order of magnitude smaller than means
 - Suggests that while there is good agreement, deviations are large enough there could be a noticeable range of variation across trials
- Suppose we had just run one trial
 - Here are the R^2 values for each trial of linear fit
 - [0.7828002156420516, 0.80637964025052067, 0.79637132757274265, 0.78433885743211906, 0.76001112024853124, 0.57088936507035748, 0.72115408562589023, 0.74358276762149023, 0.79031455375148507, 0.77920238586399471]
 - If we had only run one split, and happened to get **this** result, we might have reached a different conclusion about validity of linear model

Wrapping Up Curve Fitting

- We can use linear regression to fit a curve to data
 - Mapping from independent values to dependent values
- That curve is a model of the data that can be used to predict the value associated with independent values we haven't seen (out of sample data)
- R-squared used to evaluate model
 - Higher not always “better” because of risk of over fitting
- Choose complexity of model based on
 - Theory about structure of data
 - Cross validation
 - Simplicity

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Introduction to Machine Learning

Eric Grimson

MIT Department Of Electrical Engineering and
Computer Science

Reading assignment

- Chapter 22

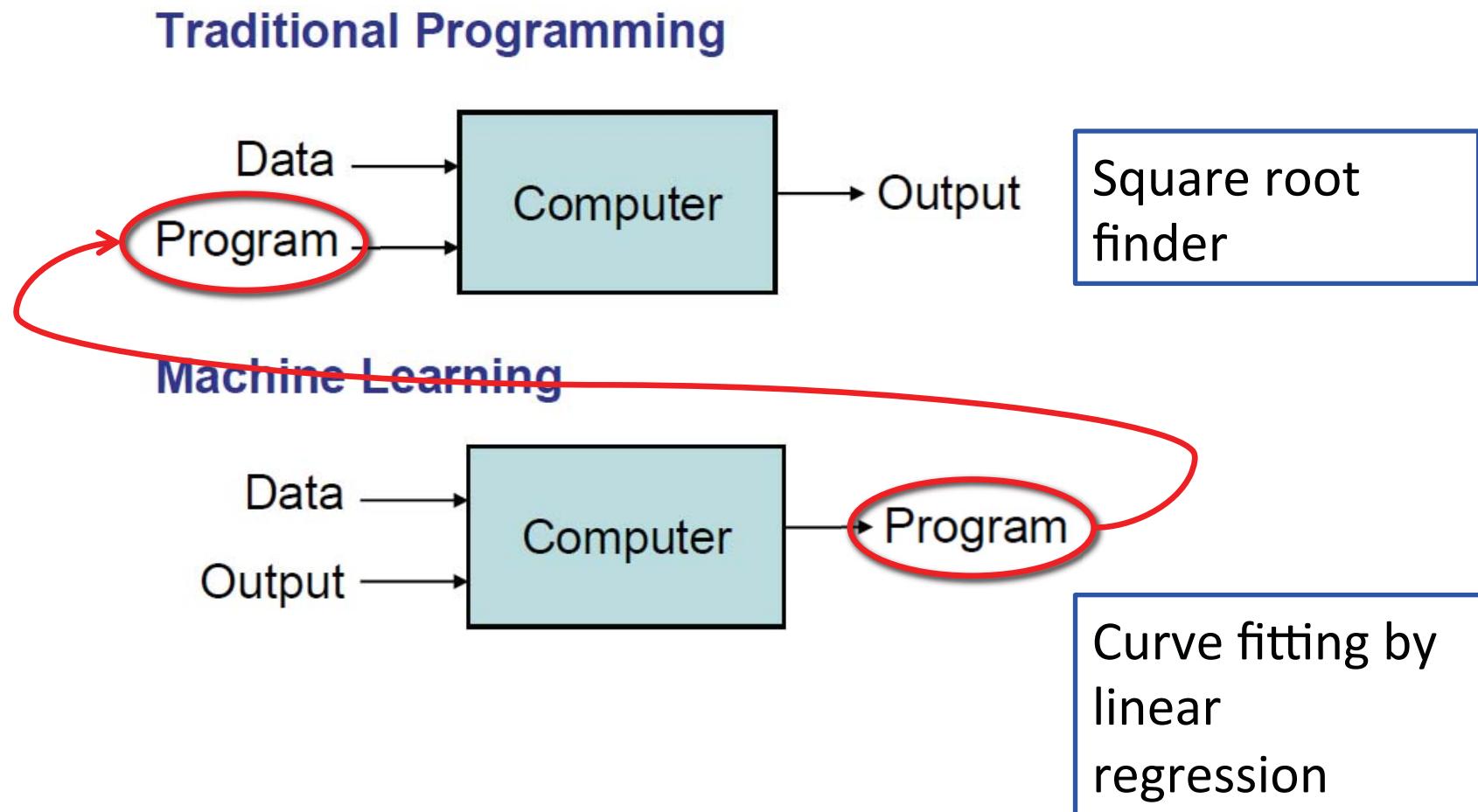
The plan ahead

- Machine learning is a huge topic – with whole courses devoted to it
 - e.g., 6.008, 6.036, 6.860, 6.862, 6.867, and as central part of courses in natural language processing, computational biology, computer vision, robotics, other areas
- In 6.0002, we will
 - Provide an introduction to the basic ideas, including ways to measure distances between examples, and how to group examples based on distance to create models
 - Introduce classification methods, such as “k nearest neighbor” methods
 - Introduce clustering methods, such as “k-means”

What Is Machine Learning?

- All useful programs “learn” something
- In the first lecture of 6.0001 we looked at an algorithm for finding square roots
- Last week we looked at using linear regression to find a model of a collection of points
- Early definition of machine learning:
 - *“Field of study that gives computers the ability to learn without being explicitly programmed.”* Arthur Samuel (1959)
 - Computer pioneer who wrote first self-learning program, which played checkers – learned from “experience”
 - Invented alpha-beta pruning – widely used in decision tree searching

What Is Machine Learning?



How Are Things Learned?

- Memorization

- Accumulation of individual facts
- Limited by
 - Time to observe facts
 - Memory to store facts

Declarative knowledge

- Generalization

- Deduce new facts from old facts
- Limited by accuracy of deduction process
 - Essentially a predictive activity
 - Assumes that the past predicts the future

Imperative knowledge

- Interested in extending to programs that can infer useful information from **implicit** patterns in data

Basic Paradigm

- Observe set of examples: **training data**
Spatial deviations relative to mass displacements of spring
- Infer something about process that generated that data
Fit polynomial curve using linear regression
- Use inference to make predictions about previously unseen data: **test data**
Predict displacements for other weights

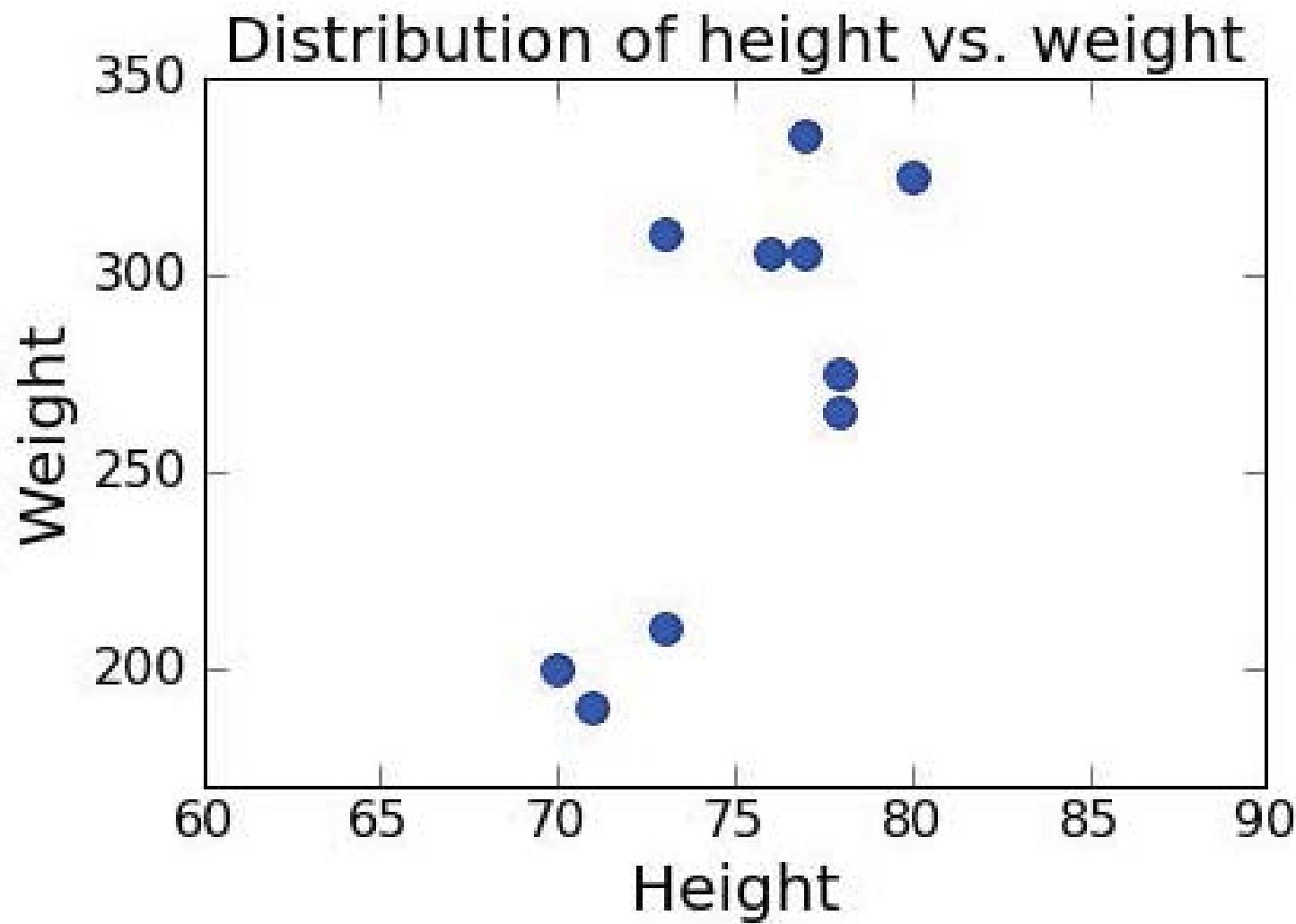
Basic Paradigm

- Observe set of examples: **training data**
Football players, labeled by position, with height and weight data
- Infer something about process that generated that data
Find canonical model of position, by statistics
- Use inference to make predictions about previously unseen data: **test data**
Predict position of new players
- Variations on paradigm
 - **Supervised:** given a set of feature/label pairs, find a rule that predicts the label associated with a previously unseen input
 - **Unsupervised:** given a set of feature vectors (without labels) group them into “natural clusters” (or create labels for groups)

Some Examples of Classifying and Clustering

- Here are some data on the New England Patriots
 - Name, height, weight
 - Labeled by type of position
- Receivers:
 - edelman = ['edelman', 70, 200]
 - hogan = ['hogan', 73, 210]
 - gronkowski = ['gronkowski', 78, 265]
 - amendola = ['amendola', 71, 190]
 - bennett = ['bennett', 78, 275]
- Linemen:
 - cannon = ['cannon', 77, 335]
 - solder = ['solder', 80, 325]
 - mason = ['mason', 73, 310]
 - thuney = ['thuney', 77, 305]
 - karras = ['karras', 76, 305]

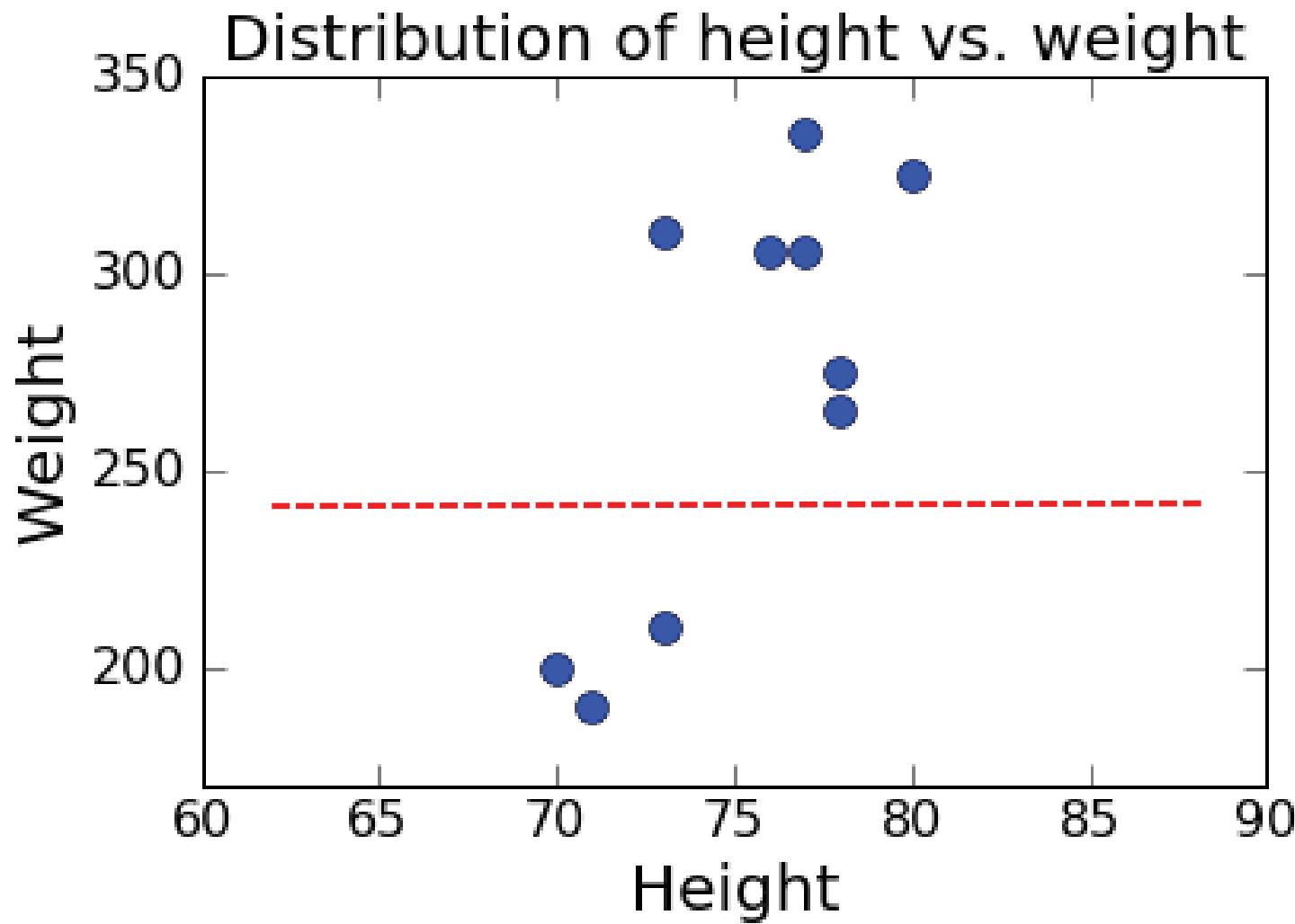
Unlabeled Data



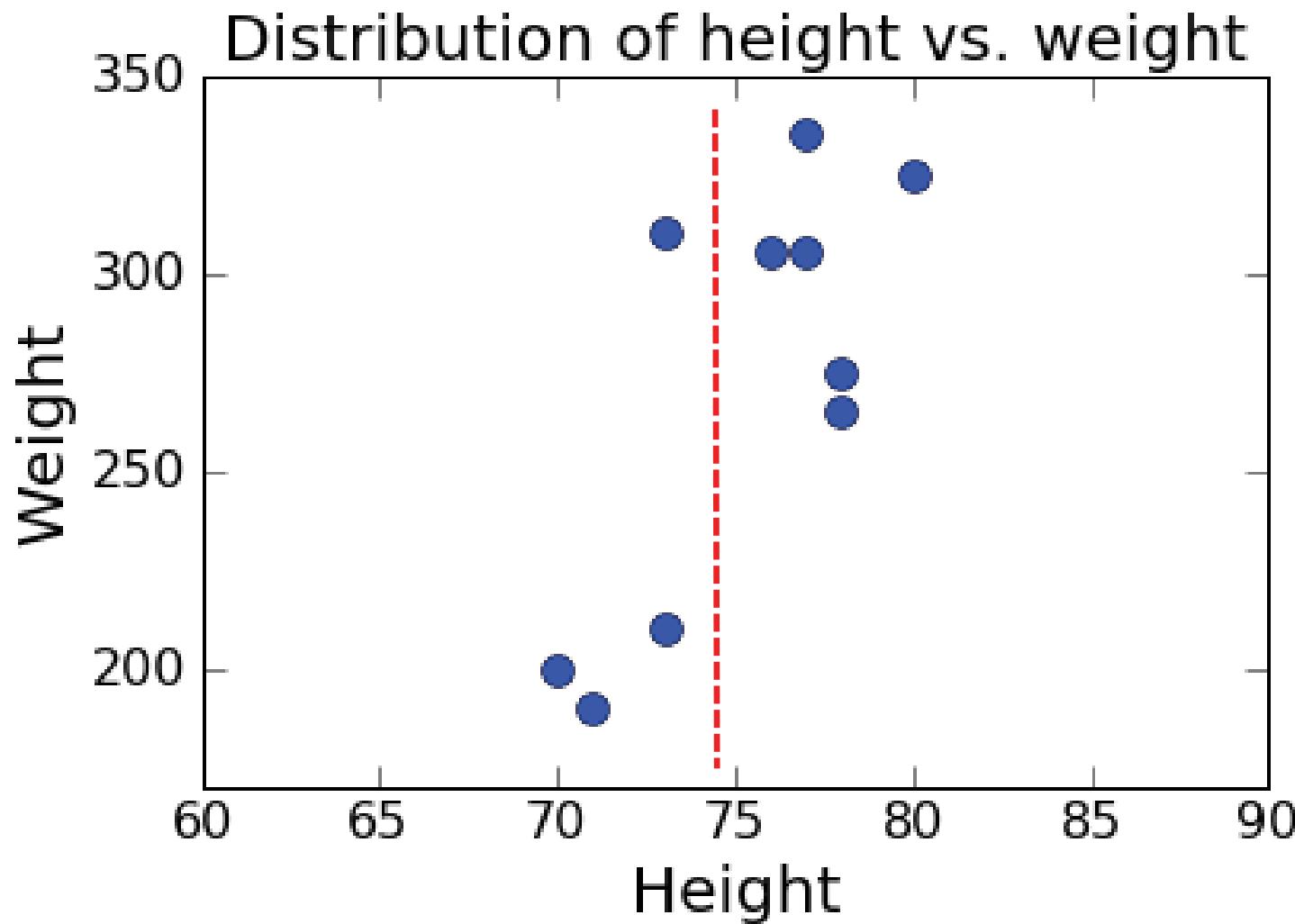
Clustering examples into groups

- Want to decide on “similarity” of examples, with goal of separating into distinct, “natural”, groups
 - Similarity is a **distance measure**
- Suppose we know that there are k different groups in our training data, but don’t know labels (here $k = 2$)
 - Pick k samples (at random?) as exemplars
 - Cluster remaining samples by minimizing distance between samples in same cluster (**objective function**) – put sample in group with closest exemplar
 - Find median example in each cluster as new exemplar
 - Repeat until no change

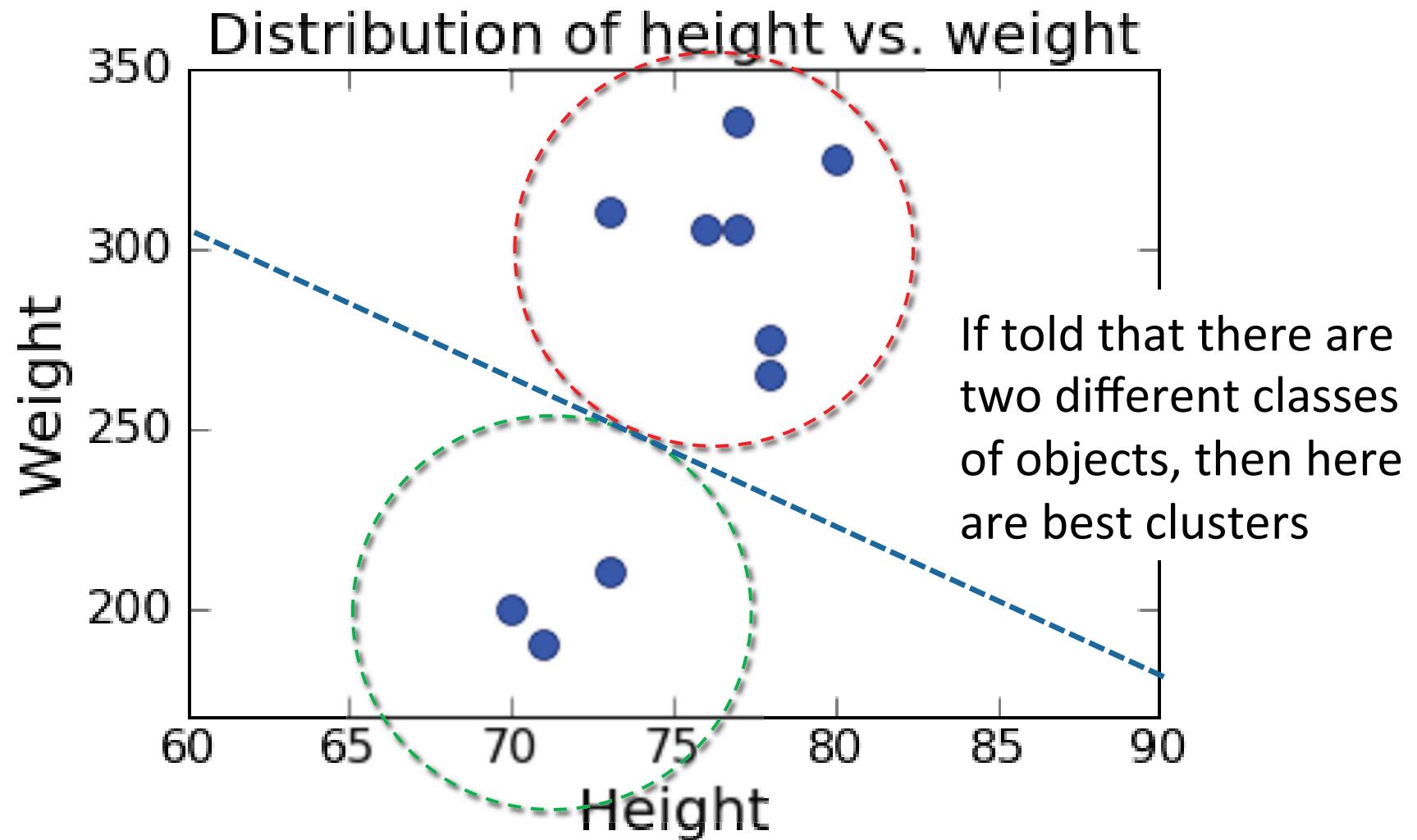
Similarity Based on Weight



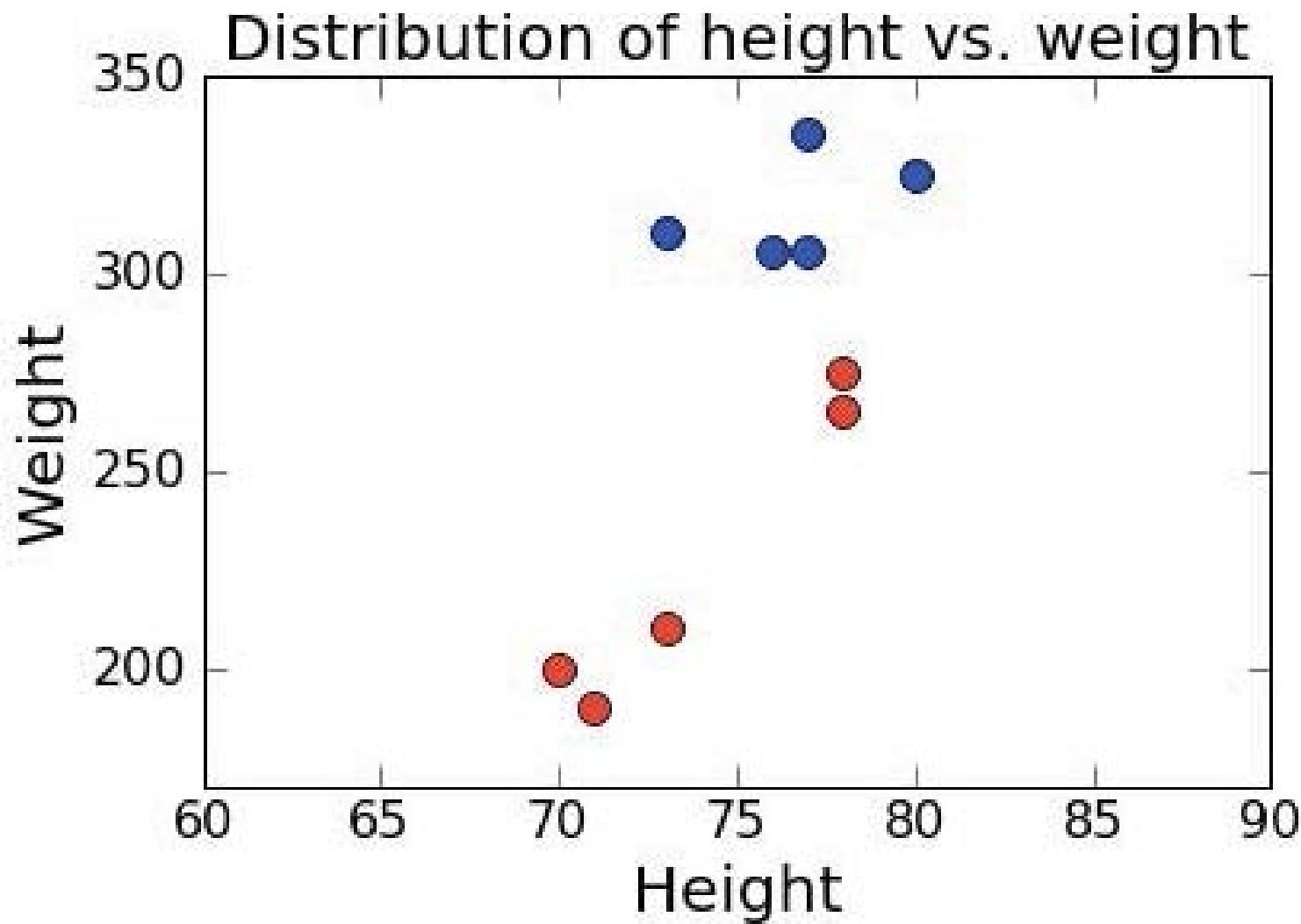
Similarity Based on Height



Cluster into Two Groups Using Both Attributes



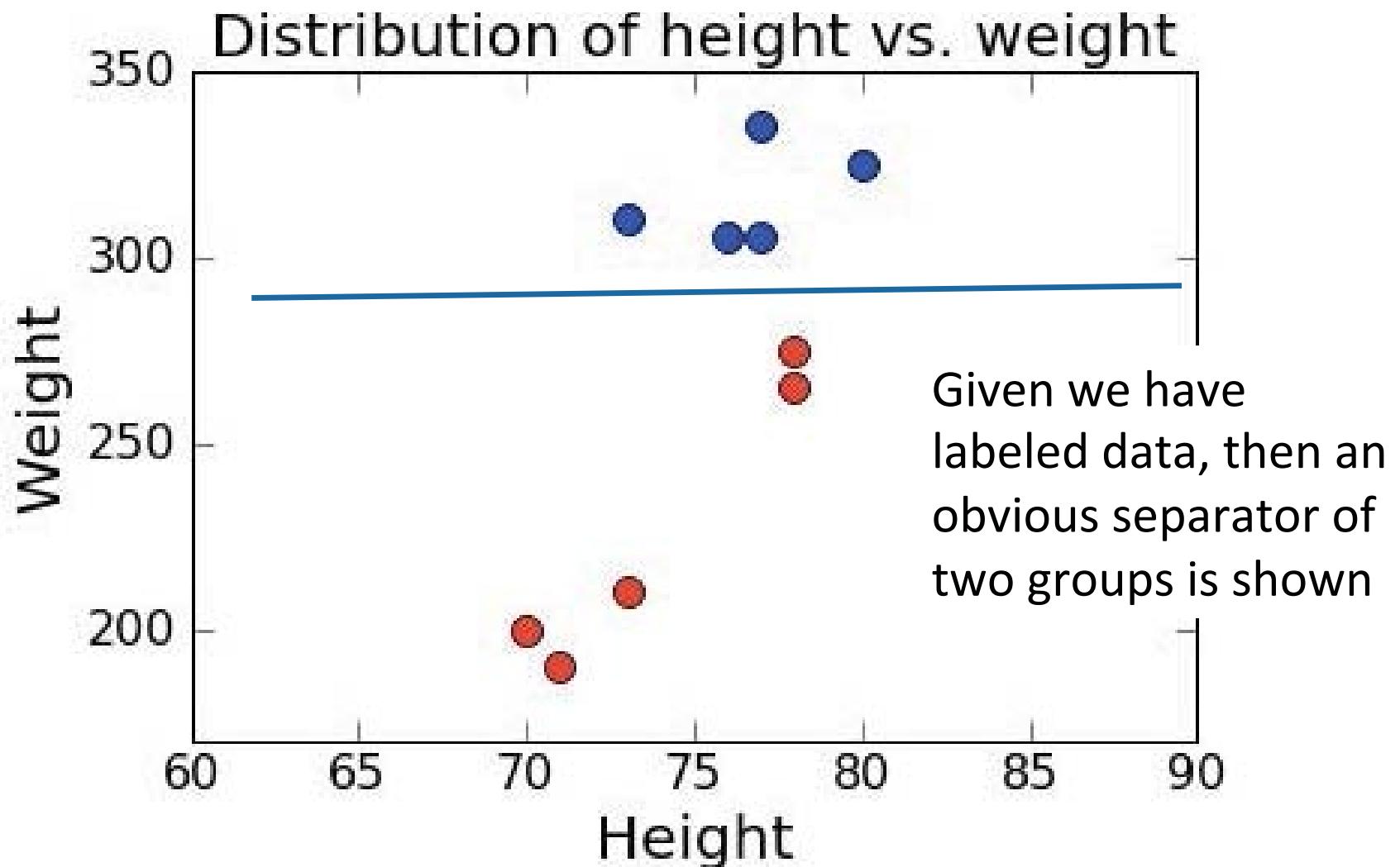
Suppose Data Was Labeled



Finding Classifier Surfaces

- Given labeled groups in feature space, want to find subsurface in that space that separates the groups
 - Subject to constraints on complexity of subsurface
- In this example, have 2D space, so find line (or connected set of line segments) that best separates the two groups
- When examples well separated, this is straightforward
- When examples in labeled groups overlap, may have to trade off false positives and false negatives

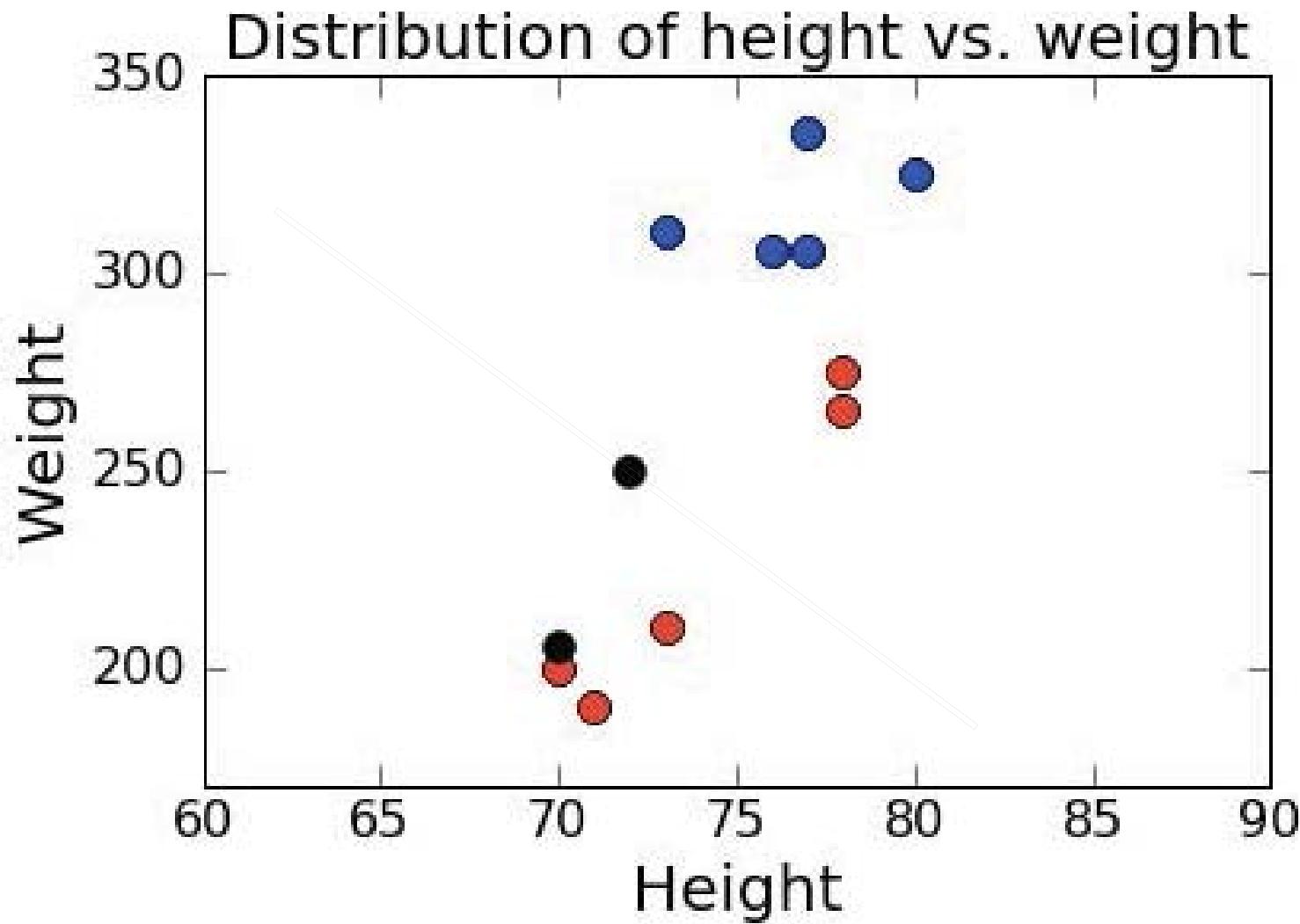
Suppose Data Was Labeled



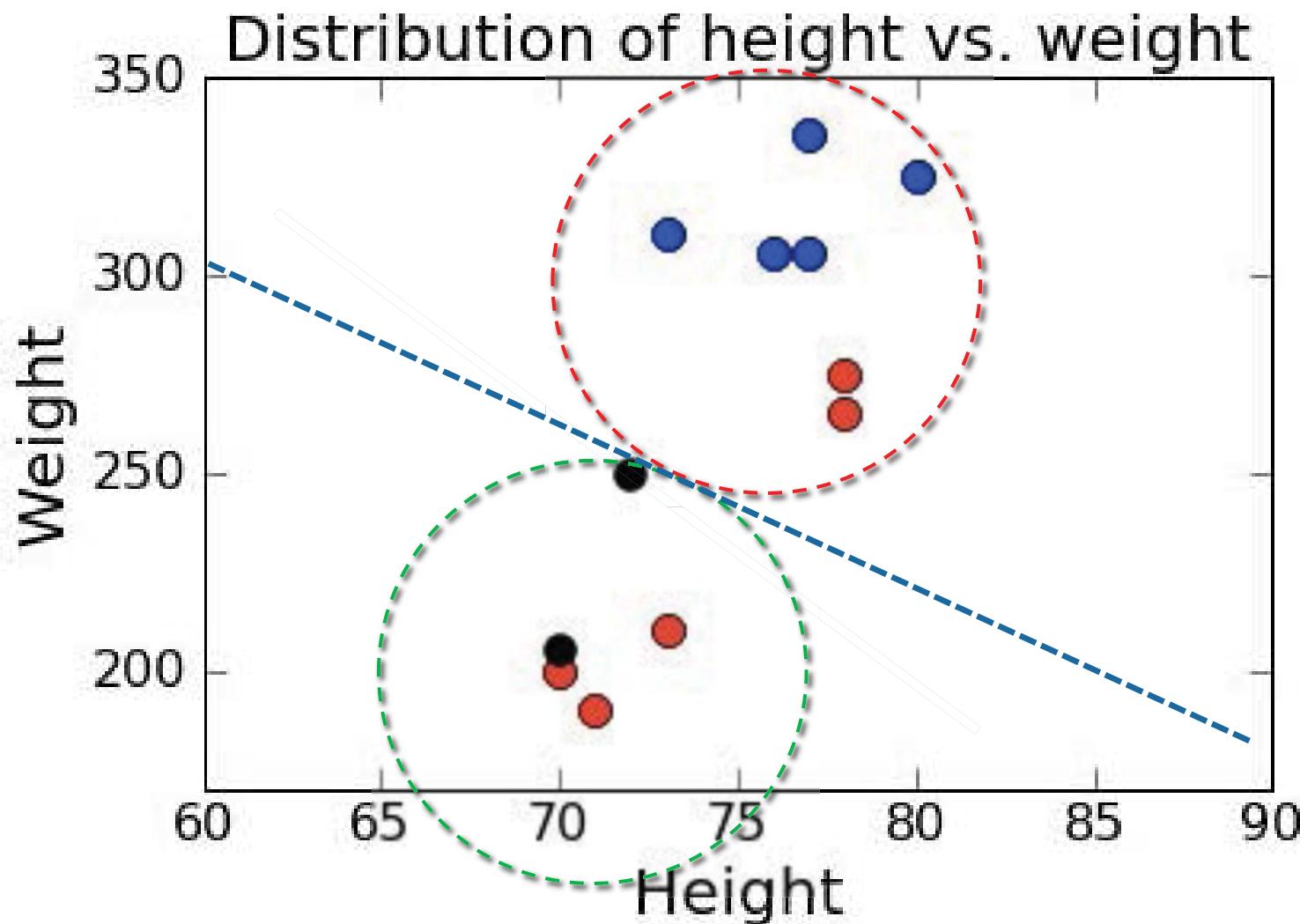
Adding Some New Data

- Suppose we have learned to separate receivers versus linemen
- Now we are given some running backs, and want to use model to decide if they are more like receivers or linemen
 - `blount = ['blount', 72, 250]`
 - `white = ['white', 70, 205]`

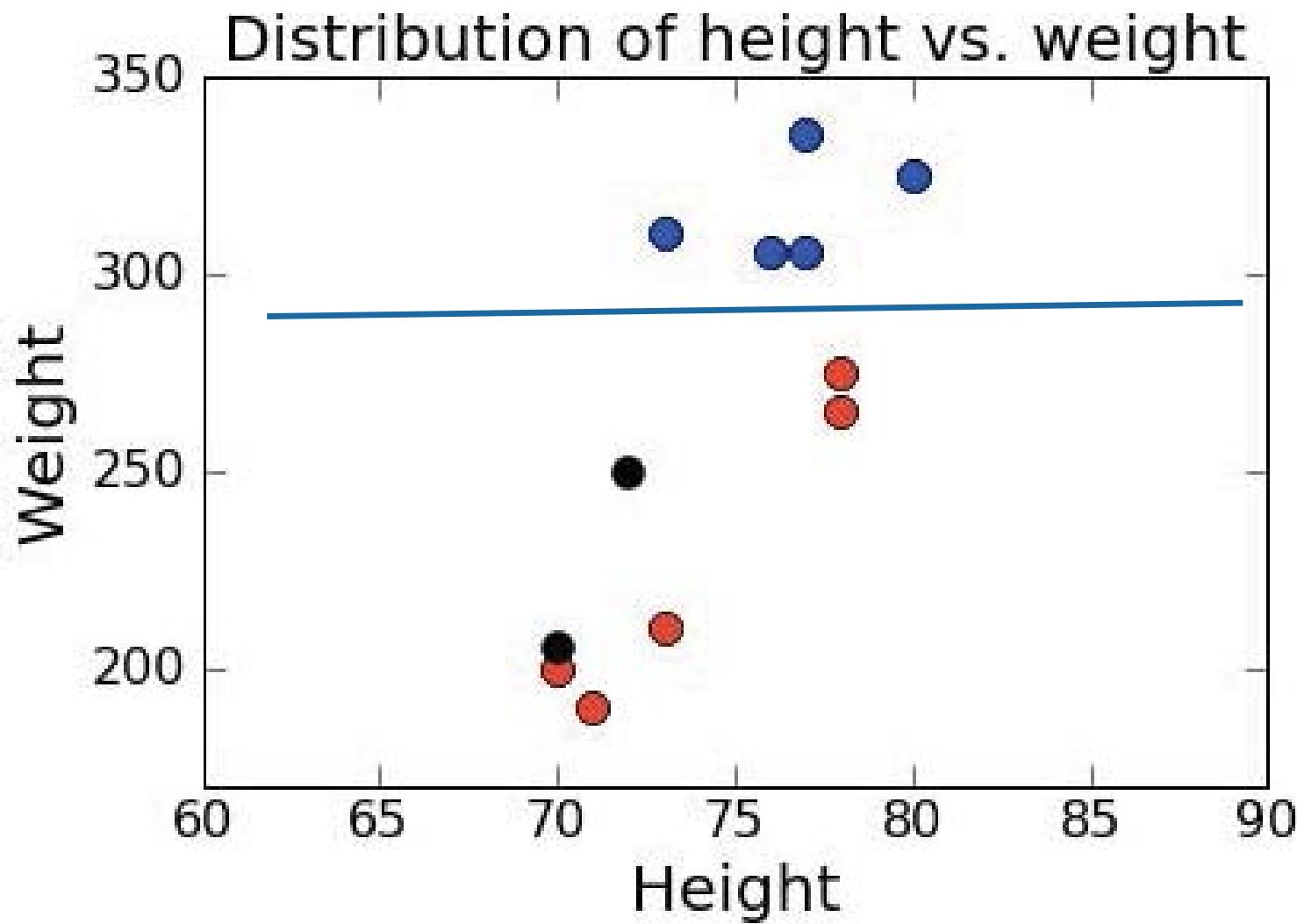
Adding Some New Data



Clustering using Unlabeled Data



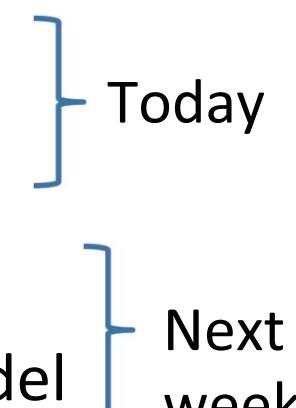
Classified using Labeled Data



Machine Learning Methods

- We will see some examples of machine learning methods:
- Learn models based on unlabeled data, by clustering training data into groups of nearby points
 - Resulting clusters can assign labels to new data
- Learn models that separate labeled groups of similar data from other groups
 - May not be possible to perfectly separate groups, without “over fitting”
 - But can make decisions with respect to trading off “false positives” versus “false negatives”
 - Resulting classifiers can assign labels to new data

All ML Methods Require:

- Choosing training data and evaluation method
 - Representation of the features
 - Distance metric for feature vectors
 - Objective function and constraints
 - Optimization method for learning the model
- 

Feature Representation

- Features never fully describe the situation
 - “All models are wrong, but some are useful.” – George Box
- Feature engineering
 - Represent examples by feature vectors that will facilitate generalization
 - Suppose I want to use 100 examples from past to predict, at the start of the subject, which students will get an A in 6.0002
 - Some features surely helpful, e.g., GPA, prior programming experience (not a perfect predictor)
 - Others might cause me to overfit, e.g., birth month, eye color
- Want to maximize ratio of useful input to irrelevant input
 - Signal-to-Noise Ratio (SNR)

An Example

	Features						Label
Name	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile	
Cobra	True	True	True	True	0	Yes	

Initial model:

- Not enough information to generalize

An Example

	Features						Label
Name	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile	
Cobra	True	True	True	True	0	Yes	
Rattlesnake	True	True	True	True	0	Yes	

Initial model:

- Egg laying
- Has scales
- Is poisonous
- Cold blooded
- No legs

An Example

Name	Features					Label
	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	
Cobra	True	True	True	True	0	Yes
Rattlesnake	True	True	True	True	0	Yes
Boa constrictor	False	True	False	True	0	Yes

Current model:

- Has scales
- Cold blooded
- No legs

Boa doesn't fit model, but is labeled as reptile.
Need to refine model

An Example

	Features					Label
Name	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile
Cobra	True	True	True	True	0	Yes
Rattlesnake	True	True	True	True	0	Yes
Boa constrictor	False	True	False	True	0	Yes
Chicken	True	True	False	False	2	No

Current model:

- Has scales
- Cold blooded
- No legs

An Example

Name	Egg-laying	Features				Label
		Scales	Poisonous	Cold-blooded	# legs	
Cobra	True	True	True	True	0	Yes
Rattlesnake	True	True	True	True	0	Yes
Boa constrictor	False	True	False	True	0	Yes
Chicken	True	True	False	False	2	No
Alligator	True	True	False	True	4	Yes

Current model:

- Has scales
- Cold blooded
- Has 0 or 4 legs

Alligator doesn't fit model, but is labeled as reptile.
Need to refine model

An Example

Name	Egg-laying	Features				Label
		Scales	Poisonous	Cold-blooded	# legs	
Cobra	True	True	True	True	0	Yes
Rattlesnake	True	True	True	True	0	Yes
Boa constrictor	False	True	False	True	0	Yes
Chicken	True	True	False	False	2	No
Alligator	True	True	False	True	4	Yes
Dart frog	True	False	True	False	4	No

Current model:

- Has scales
- Cold blooded
- Has 0 or 4 legs

An Example

Name	Egg-laying	Features				Label
		Scales	Poisonous	Cold-blooded	# legs	
Cobra	True	True	True	True	0	Yes
Rattlesnake	True	True	True	True	0	Yes
Boa constrictor	False	True	False	True	0	Yes
Chicken	True	True	False	False	2	No
Alligator	True	True	False	True	4	Yes
Dart frog	True	False	True	False	4	No
Salmon	True	True	False	True	0	No
Python	True	True	False	True	0	Yes

Current model:

- Has scales
- Cold blooded
- Has 0 or 4 legs

No (easy) way to add to rule that will correctly classify salmon and python (since identical feature values)

An Example

Name	Egg-laying	Features			Label	
		Scales	Poisonous	Cold-blooded	# legs	Reptile
Cobra	True	True	True	True	0	Yes
Rattlesnake	True	True	True	True	0	Yes
Boa constrictor	False	True	False	True	0	Yes
Chicken	True	True	False	False	2	No
Alligator	True	True	False	True	4	Yes
Dart frog	True	False	True	False	4	No
Salmon	True	True	False	True	0	No
Python	True	True	False	True	0	Yes

Good model:

- Has scales
- Cold blooded

Not perfect, but no false negatives (anything classified as not reptile is correctly labeled); some false positives (may incorrectly label some animals as reptile)

Need to Measure Distances between Features

- Feature engineering:

- Deciding which features to include and which are merely adding noise to classifier
- Defining how to measure distances between training examples (and ultimately between classifiers and new instances)
- Deciding how to weight relative importance of different dimensions of feature vector, which impacts definition of distance

Measuring Distance Between Animals

- We can think of our animal examples as consisting of four binary features and one integer feature
- One way to learn to separate reptiles from non-reptiles is to measure the distance between pairs of examples, and use that:
 - To cluster nearby examples into a common class (unlabeled data), or
 - To find a classifier surface in space of examples that optimally separates different (labeled) collections of examples from other collections

```
rattlesnake = [1,1,1,1,0]  
boa constrictor = [0,1,0,1,0]  
dart Frog = [1,0,1,0,4]
```

Can convert examples
into feature vectors

Minkowski Metric

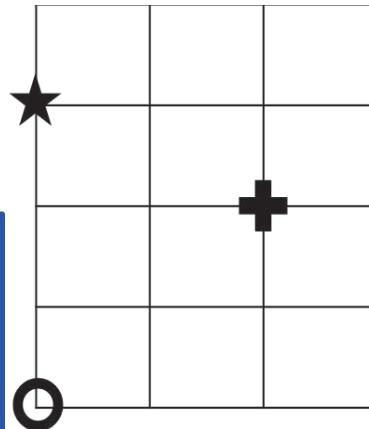
$$dist(X1, X2, p) = \left(\sum_{k=1}^{\text{len}} abs(X1_k - X2_k)^p \right)^{1/p}$$

p = 1: Manhattan Distance

p = 2: Euclidean Distance

Need to measure distances between feature vectors

Typically use Euclidean metric; Manhattan may be appropriate if different dimensions are not comparable



Is circle closer to star or cross?

- Euclidean distance
 - Cross – 2.8
 - Star – 3
- Manhattan Distance
 - Cross – 4
 - Star - 3

Euclidean Distance Between Animals

rattlesnake = [1,1,1,1,0]

boa constrictor = [0,1,0,1,0]

dartFrog = [1,0,1,0,4]



Patrick K. Campbell/Shutterstock.com



Images of rattlesnake, dart frog, boa constrictor © sources unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.

Euclidean Distance Between Animals

```
rattlesnake = [1,1,1,1,0]
```

```
boa constrictor = [0,1,0,1,0]
```

```
dartFrog = [1,0,1,0,4]
```

	rattlesnake	boa constrictor	dart frog
rattlesnake	--	1.414	4.243
boa constrictor	1.414	--	4.472
dart frog	4.243	4.472	--

Using Euclidean distance, rattlesnake and boa constrictor are much closer to each other, than they are to the dart frog

Add an Alligator

- alligator = Animal('alligator', [1,1,0,1,4])
- animals.append(alligator)
- compareAnimals(animals, 3)



Image of alligator © source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.

Add an Alligator

```
▪ alligator = Animal('alligator', [1,1,0,1,4])
▪ animals.append(alligator)
▪ compareAnimals(animals, 3)
```

	rattlesnake	boa constrictor	dart frog	alligator
rattlesnake	--	1.414	4.243	4.123
boa constrictor	1.414	--	4.472	4.123
dart frog	4.243	4.472	--	1.732
alligator	4.123	4.123	1.732	--

Alligator is closer to dart frog than to snakes – why?

- Alligator differs from frog in 3 features, from boa in only 2 features
- But scale on “legs” is from 0 to 4, on other features is 0 to 1
- “legs” dimension is disproportionately large

Using Binary Features

```
rattlesnake = [1,1,1,1,0]
```

```
boa constrictor = [0,1,0,1,0]
```

```
dartFrog = [1,0,1,0,1]
```

```
Alligator = [1,1,0,1,1]
```

	rattlesnake	boa constrictor	dart frog	alligator
rattlesnake	--	1.414	1.732	1.414
boa constrictor	1.414	--	2.236	1.414
dart frog	1.732	2.236	--	1.732
alligator	1.414	1.414	1.732	--

Now alligator is closer to snakes than it is to dart frog
– makes more sense

Feature Engineering Matters

Supervised versus Unsupervised Learning

- In the next few lectures, we will see examples of learning algorithms:
- When given unlabeled data, try to find clusters of examples near each other
 - Use centroids of clusters as definition of each learned class
 - New data assigned to closest cluster
- When given labeled data, learn mathematical surface that “best” separates labeled examples, subject to constraints on complexity of surface (don’t over fit)
 - New data assigned to class based on portion of feature space carved out by classifier surface in which it lies

Issues of Concern When Learning Models

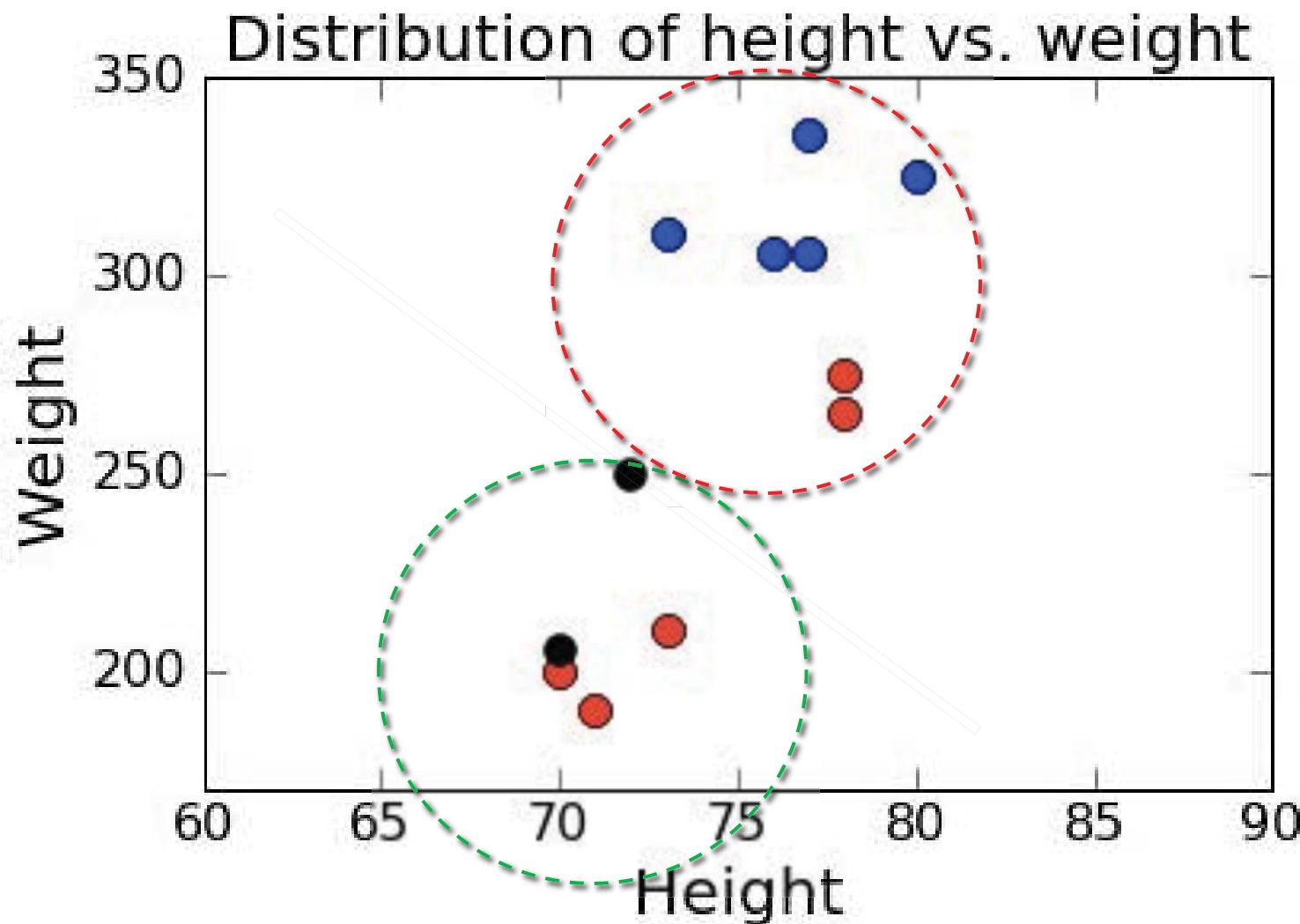
Learned models will depend on:

- Distance metric between examples
- Choice of feature vectors
- Constraints on complexity of model
 - Specified number of clusters
 - Complexity of separating surface
 - Want to avoid over fitting problem (each example is its own cluster, or a complex separating surface)

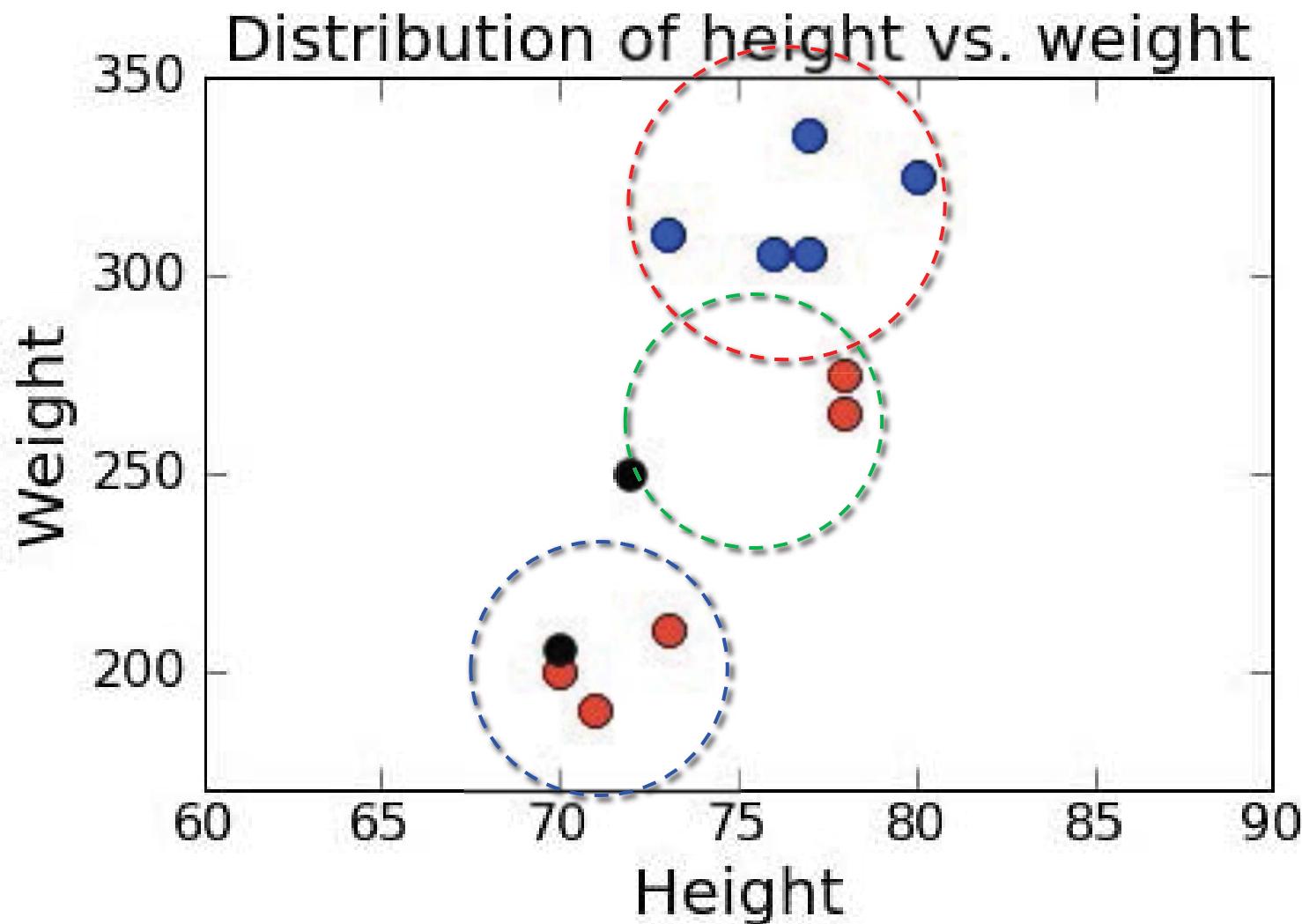
Clustering approaches

- Suppose we know that there are k different groups in our training data, but don't know labels
 - Pick k samples (at random?) as exemplars
 - Cluster remaining samples by minimizing distance between samples in same cluster (**objective function**) – put sample in group with closest exemplar
 - Find median example in each cluster as new exemplar
 - Repeat until no change
- Issues:
 - How do we decide on the best number of clusters?
 - How do we select the best features, the best distance metric?

Clustering using Unlabeled Data



Fitting Three Clusters Unsupervised



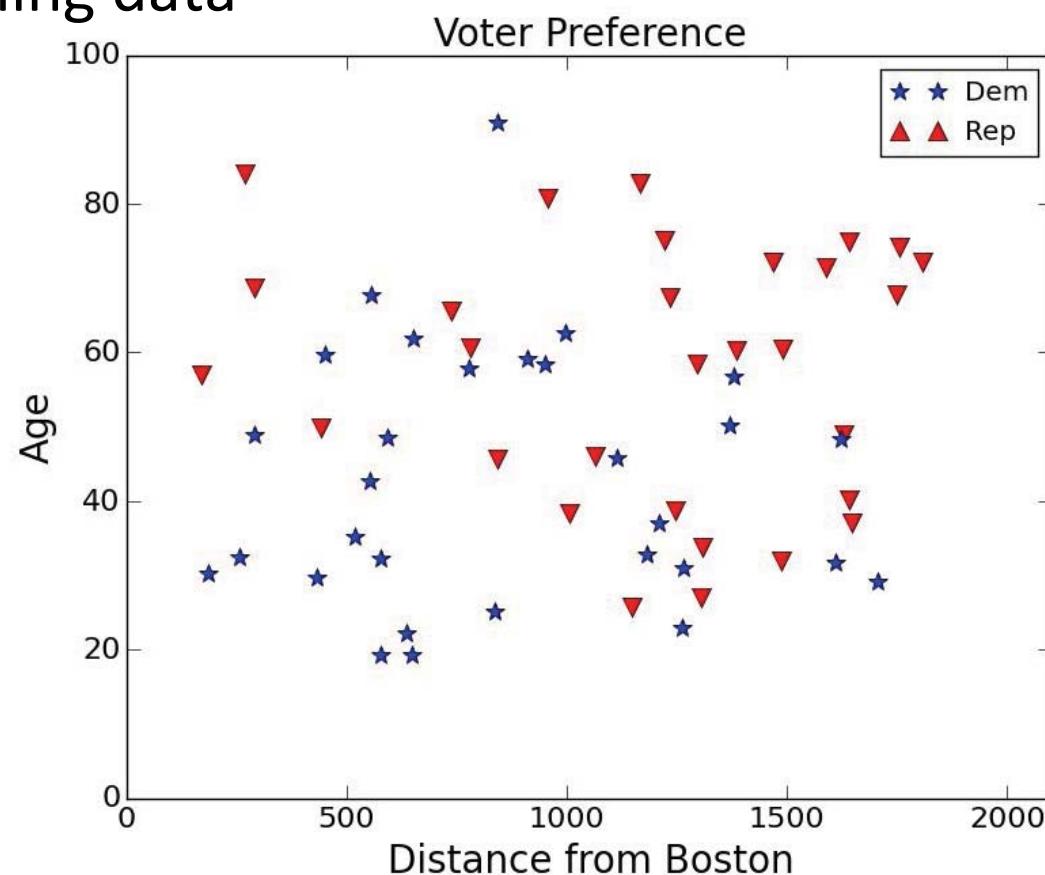
Classification approaches

- Want to find boundaries in feature space that separate different classes of labeled examples
 - Look for simple surface (e.g. best line or plane) that separates classes
 - Look for more complex surfaces (subject to constraints) that separate classes
 - Use voting schemes
 - Find k nearest training examples, use majority vote to select label
- Issues:
 - How do we avoid over-fitting to data?
 - How do we measure performance?
 - How do we select best features?

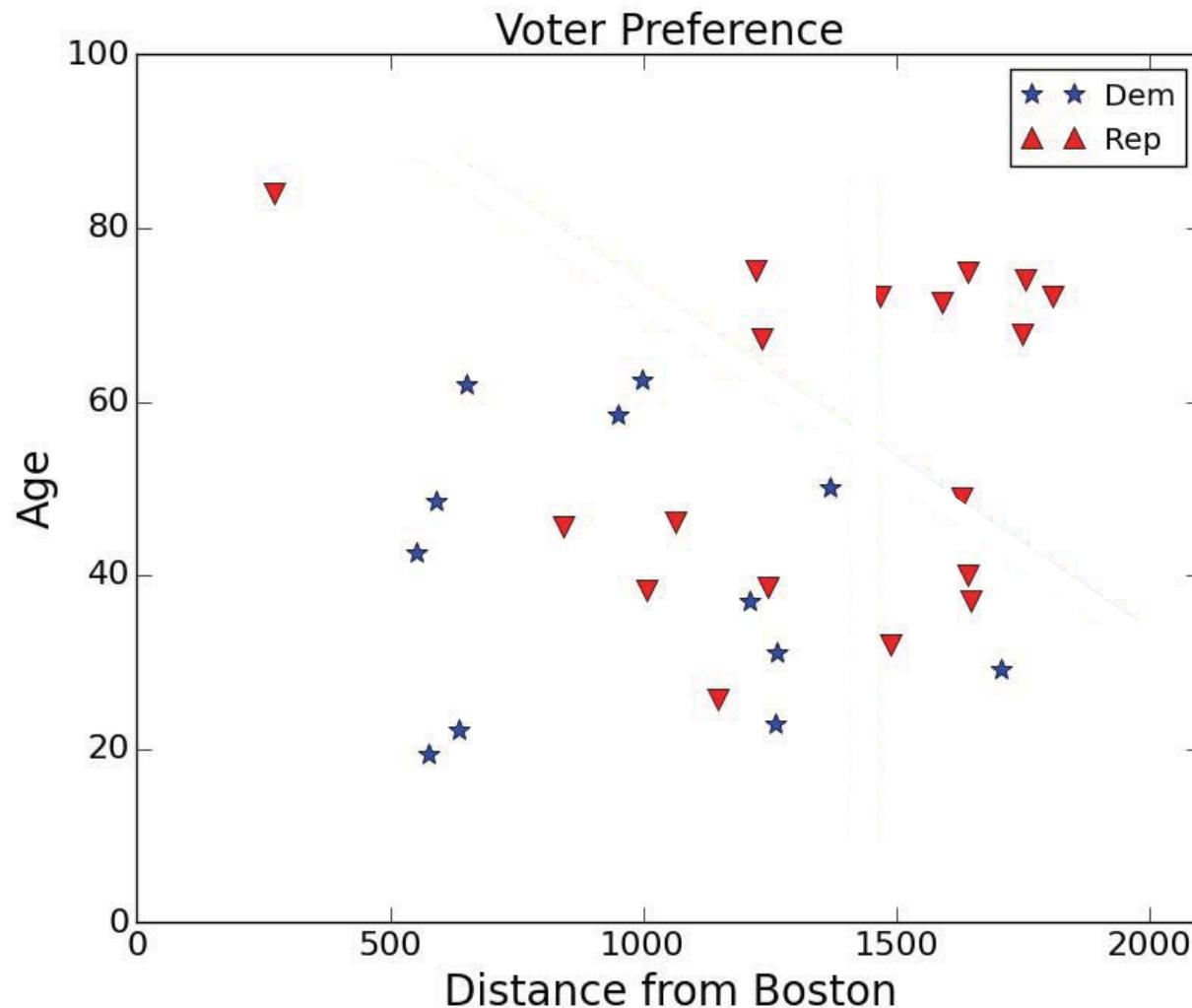
Classification

- Attempt to minimize error on training data
 - Similar to fitting a curve to data
- Evaluate on training data

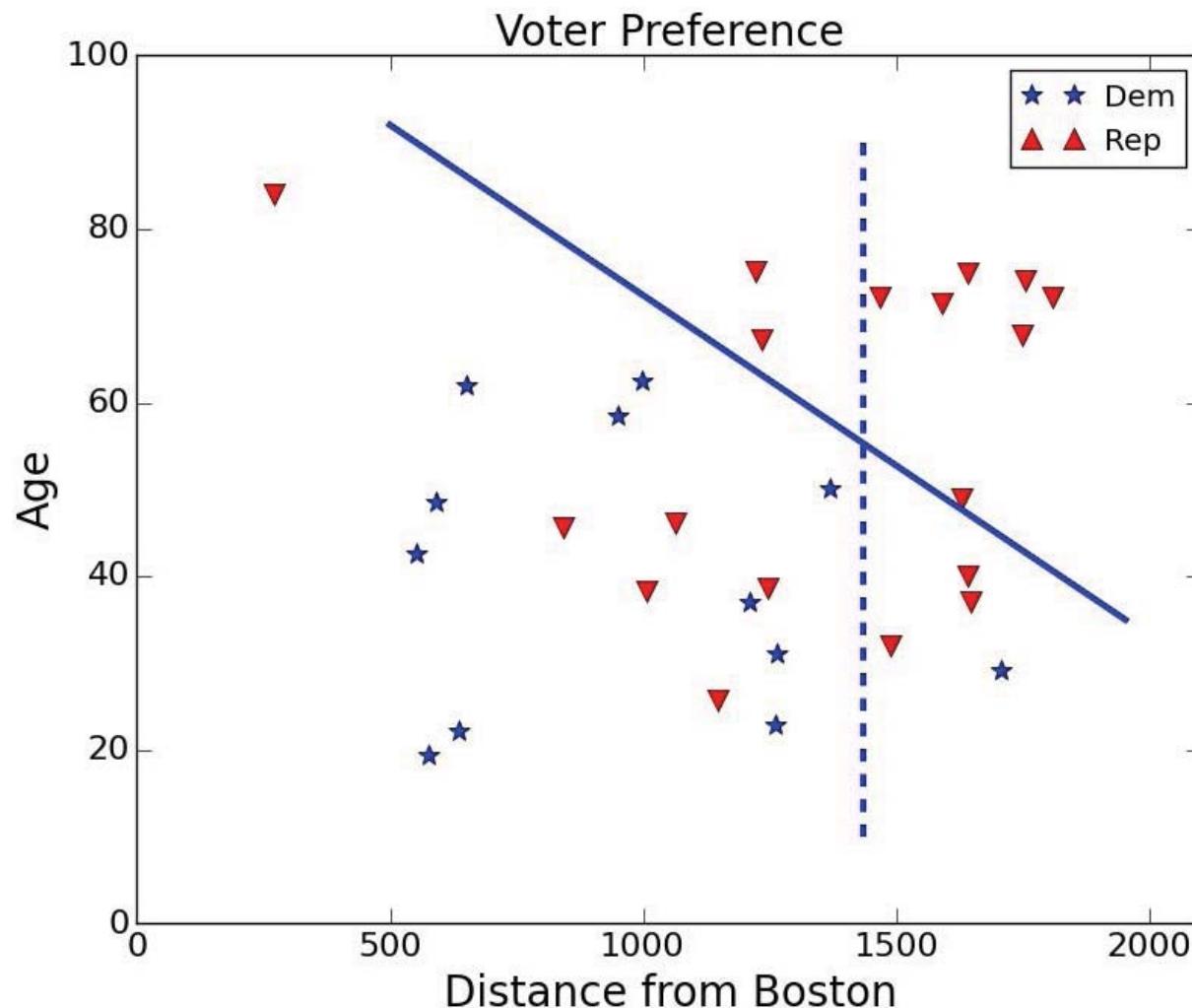
Voter preference,
by age and
distance from
Boston



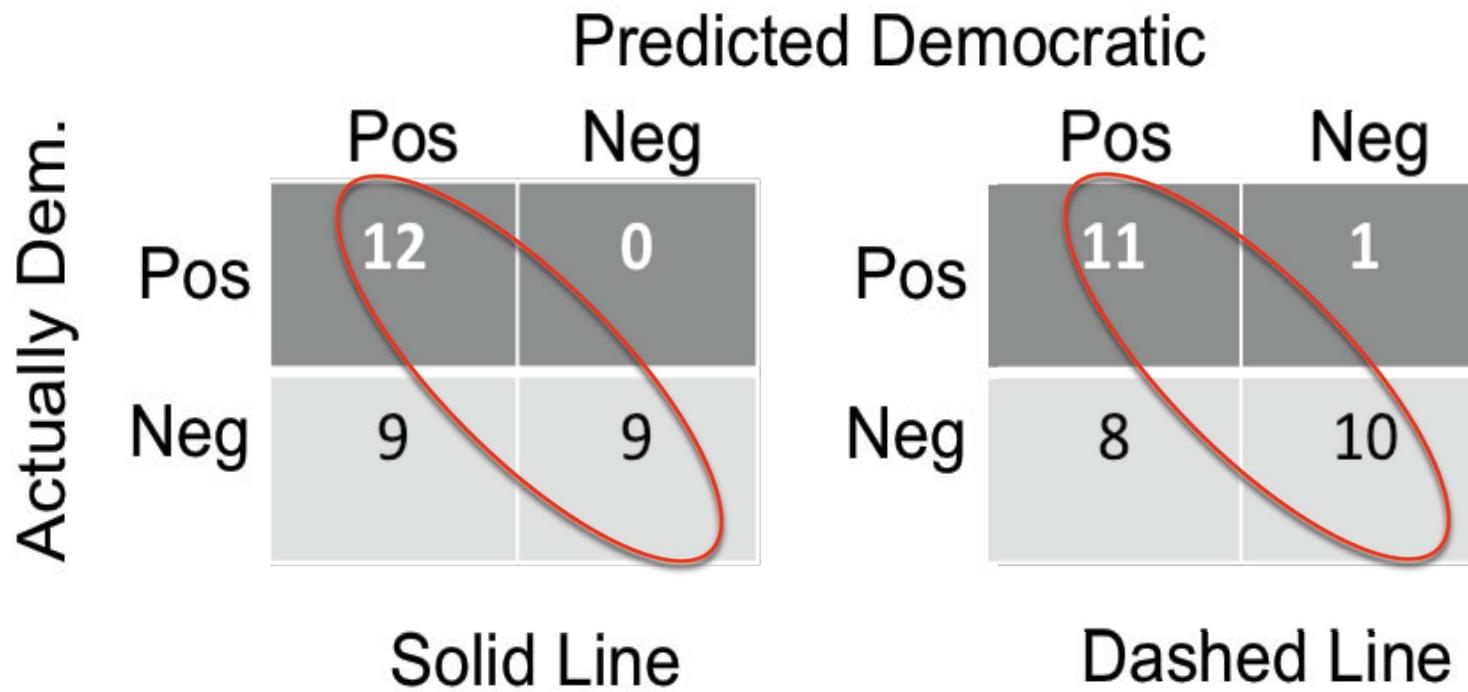
Randomly Divide Data into Training and Test Set



Two Possible Models for a Training Set



Confusion Matrices (Training Error)

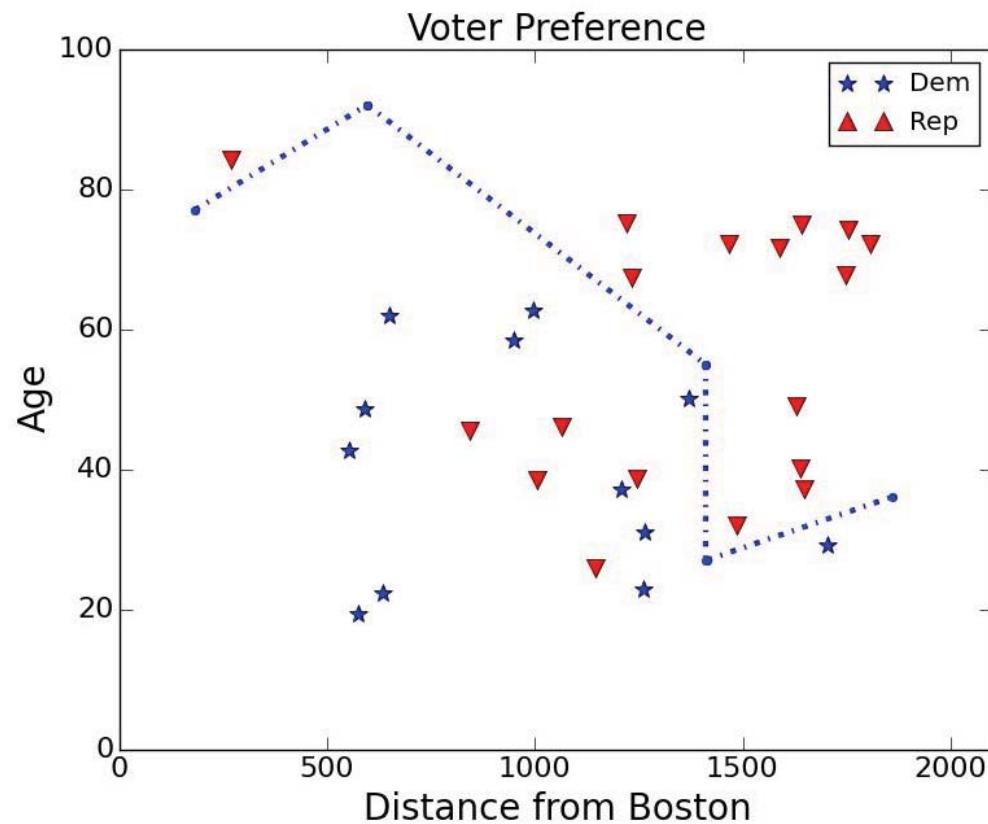


Training Accuracy of Models

$$accuracy = \frac{true\ positive + true\ negative}{true\ positive + true\ negative + false\ positive + false\ negative}$$

- 0.7 for both models
 - Which is better?
- Can we find a model with less training error?

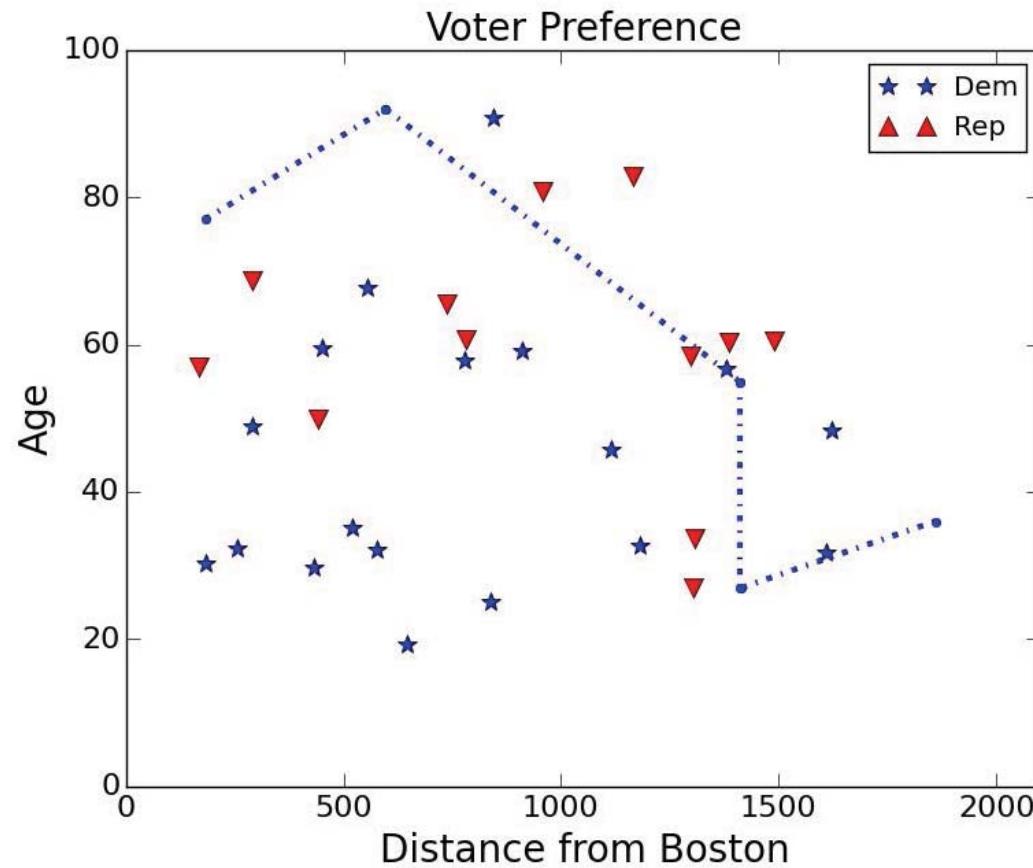
A More Complex Model



TP = 12, FP = 5, TN = 13, FN = 0

Accuracy = 25/30 = 0.833

Applying Model to Test Data



TP = 14, FP = 4, TN = 4, FN = 8

Accuracy = 18/30 = 0.6

Other Statistical Measures

$$\text{positive predictive value} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

- Solid line model: .57
- Dashed line model: .58
- Complex model, training: .71
- Complex model, testing: .78
- You will also see “sensitivity” versus “specificity”

$$\text{sensitivity} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

$$\text{specificity} = \frac{\text{true negative}}{\text{true negative} + \text{false positive}}$$

Percentage
correctly
found

Percentage
correctly
rejected

Summary

- Machine learning methods provide a way of building models of processes from data sets
 - Supervised learning uses labeled data, and creates classifiers that optimally separate data into known classes
 - Unsupervised learning tries to infer latent variables by clustering training examples into nearby groups
- Choice of features influences results
- Choice of distance measurement between examples influences results
- We will see some examples of clustering methods, such as k-means
- We will see some examples of classifiers, such as k nearest neighbor methods

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Lecture 12: Clustering

Reading

- Chapter 23

Machine Learning Paradigm

- Observe set of examples: **training data**
- Infer something about process that generated that data
- Use inference to make predictions about previously unseen data: **test data**
- Supervised: given a set of feature/label pairs, find a rule that predicts the label associated with a previously unseen input
- *Unsupervised*: given a set of feature vectors (without labels) group them into “natural clusters”

Clustering Is an Optimization Problem

$$\text{variability}(c) = \sum_{e \in c} \text{distance}(\text{mean}(c), e)^2$$

$$\text{dissimilarity}(C) = \sum_{c \in C} \text{variability}(c)$$

- Why not divide variability by size of cluster?
 - Big and bad worse than small and bad
- Is optimization problem finding a C that minimizes $\text{dissimilarity}(C)$?
 - No, otherwise could put each example in its own cluster
- Need a constraint, e.g.,
 - Minimum distance between clusters
 - Number of clusters

Two Popular Methods

- Hierarchical clustering
- K-means clustering

Hierarchical Clustering

1. Start by assigning each item to a cluster, so that if you have N items, you now have N clusters, each containing just one item.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster, so that now you have one fewer cluster.
3. Continue the process until all items are clustered into a single cluster of size N .

What does distance mean?

Linkage Metrics

- *Single-linkage*: consider the distance between one cluster and another cluster to be equal to the shortest distance from any member of one cluster to any member of the other cluster
- *Complete-linkage*: consider the distance between one cluster and another cluster to be equal to the greatest distance from any member of one cluster to any member of the other cluster
- *Average-linkage*: consider the distance between one cluster and another cluster to be equal to the average distance from any member of one cluster to any member of the other cluster

Example of Hierarchical Clustering

	BOS	NY	CHI	DEN	SF	SEA
BOS	0	206	963	1949	3095	2979
NY		0	802	1771	2934	2815
CHI			0	966	2142	2013
DEN				0	1235	1307
SF					0	808
SEA						0

{BOS} {NY}

{BOS, NY}

{BOS, NY, CHI}

{BOS, NY, CHI}

{BOS, NY, CHI, DEN}

{CHI}

{CHI}

{DEN}

{DEN}

{DEN}

{DEN}

{SF, SEA}

{SF}

{SF}

{SF}

{SF, SEA}

Single linkage

or

{BOS, NY, CHI}

{DEN, SF, SEA} Complete linkage

Clustering Algorithms

- Hierarchical clustering
 - Can select number of clusters using dendrogram
 - Deterministic
 - Flexible with respect to linkage criteria
 - Slow
 - Naïve algorithm n^3
 - n^2 algorithms exist for some linkage criteria
- K-means a much faster greedy algorithm
 - Most useful when you know how many clusters you want

K-means Algorithm

randomly chose k examples as initial centroids
while true:

 create k clusters by assigning each
 example to closest centroid

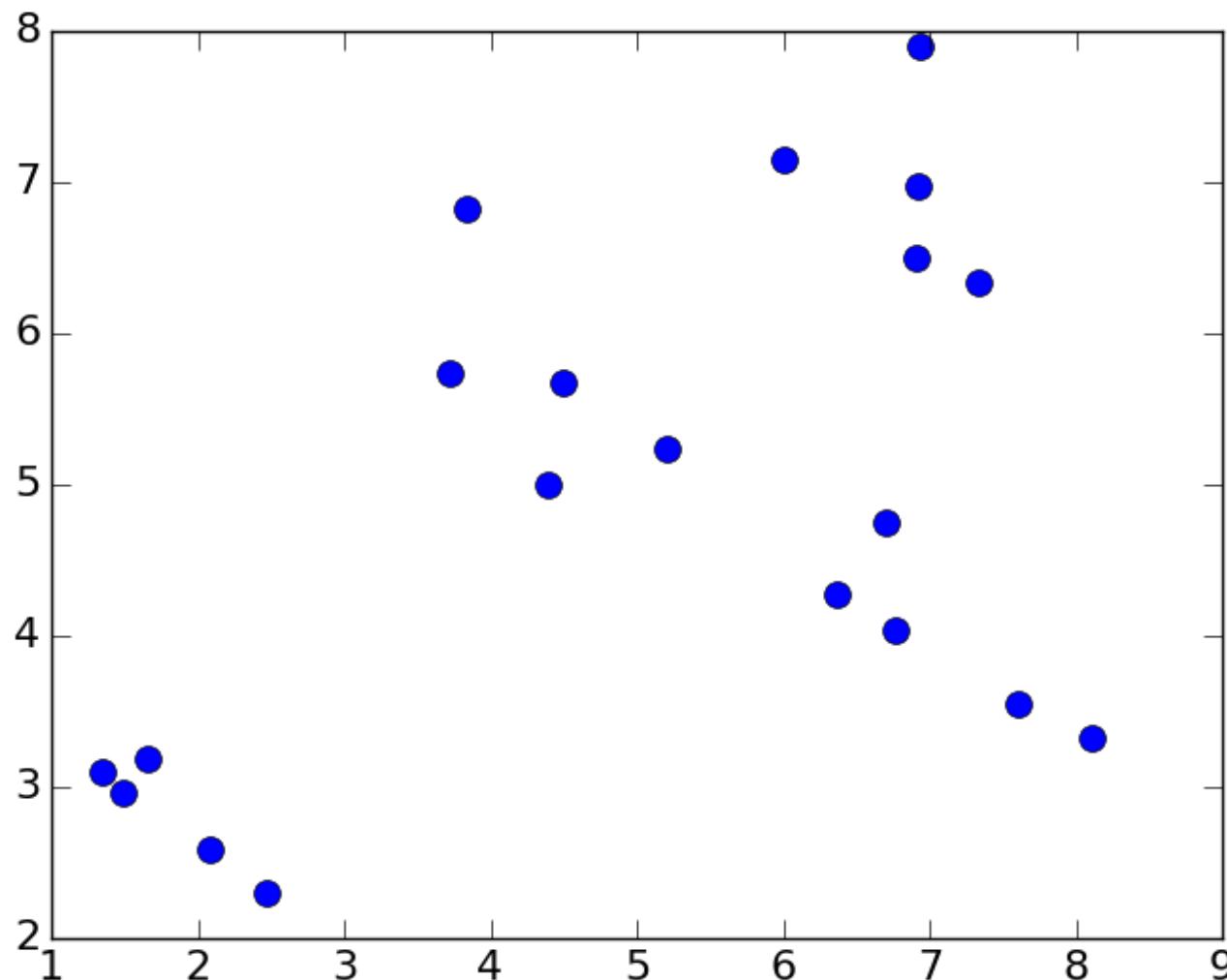
 compute k new centroids by averaging
 examples in each cluster

 if centroids don't change:
 break

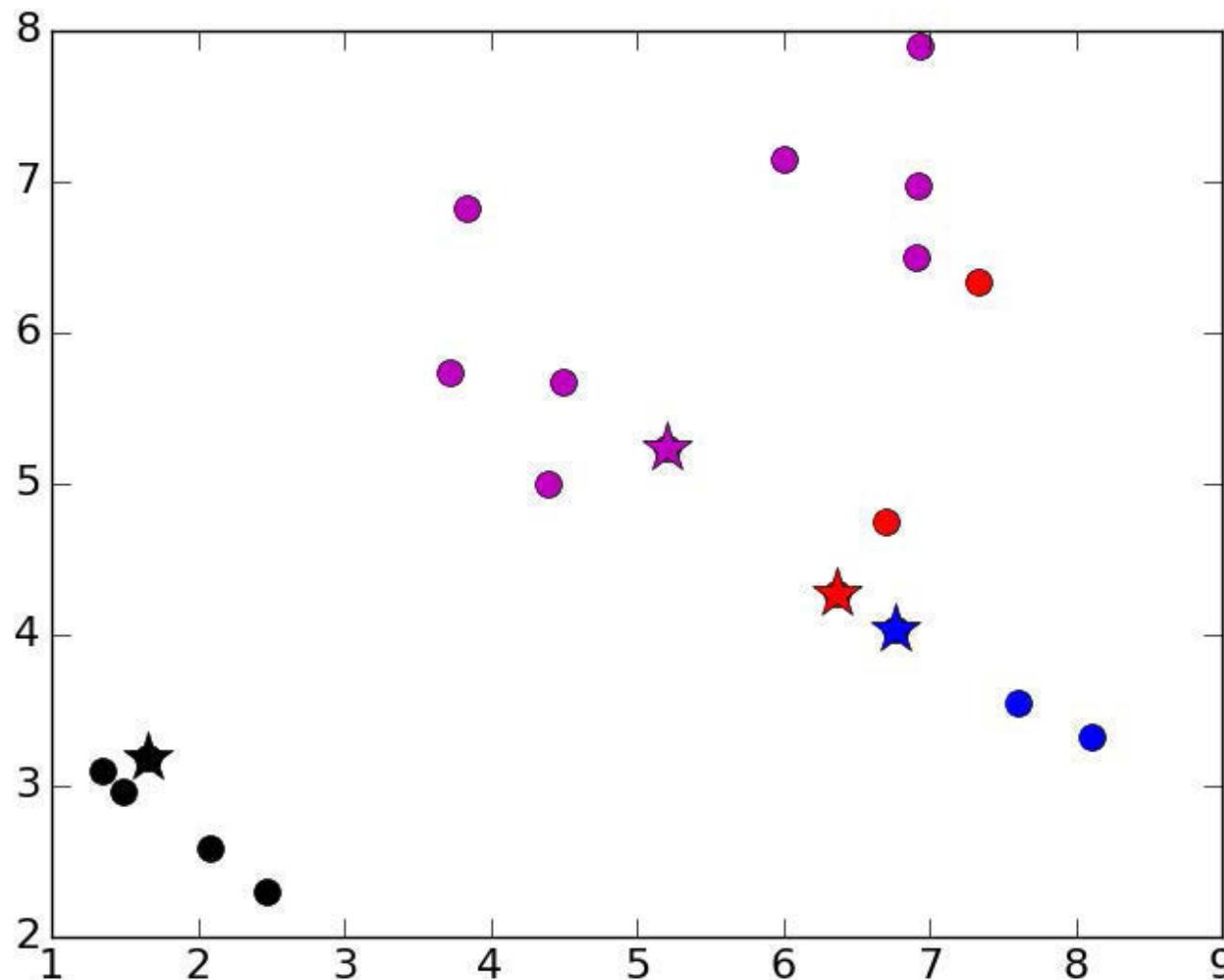
What is complexity of one iteration?

$k * n * d$, where n is number of points and d time required
to compute the distance between a pair of points

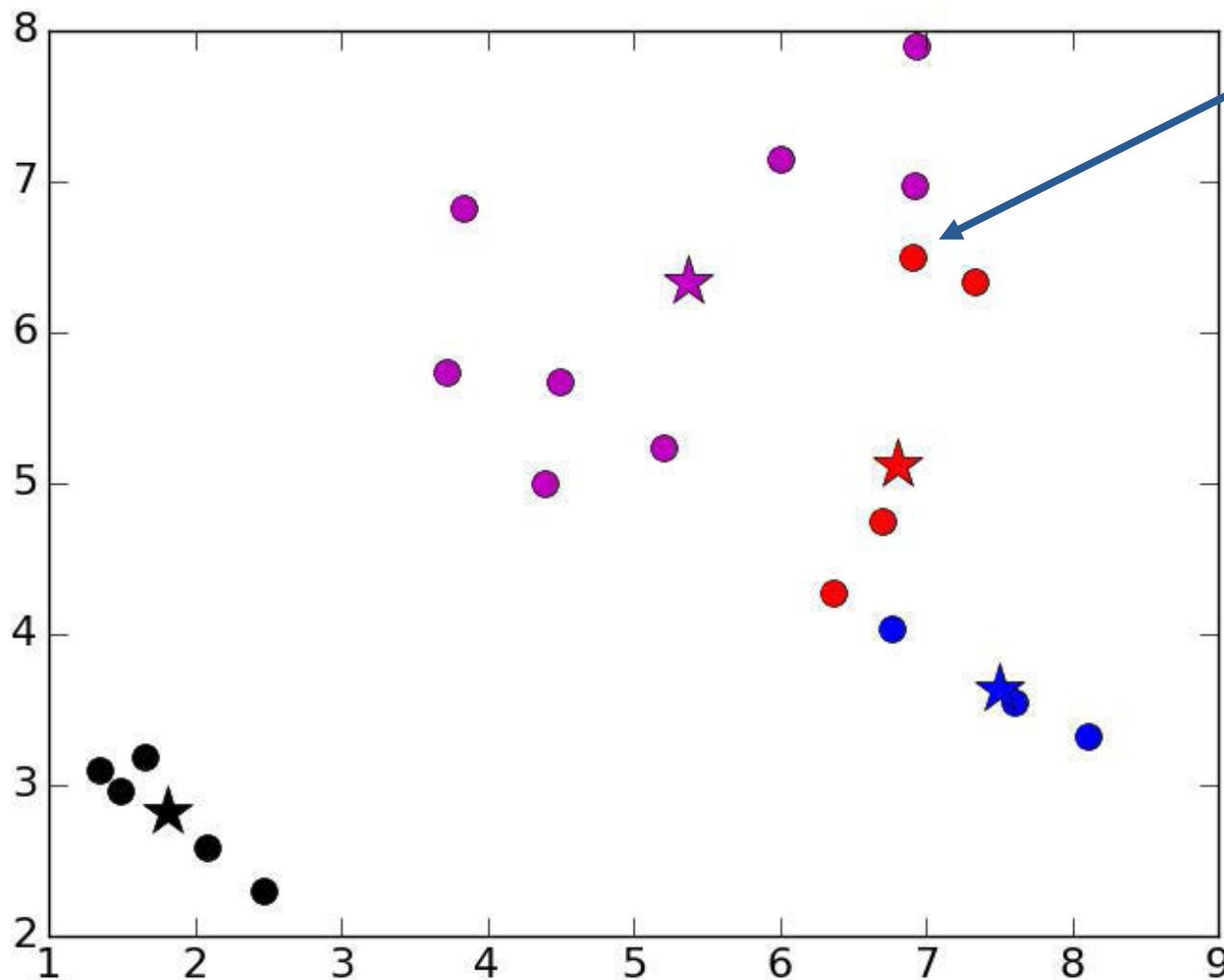
An Example



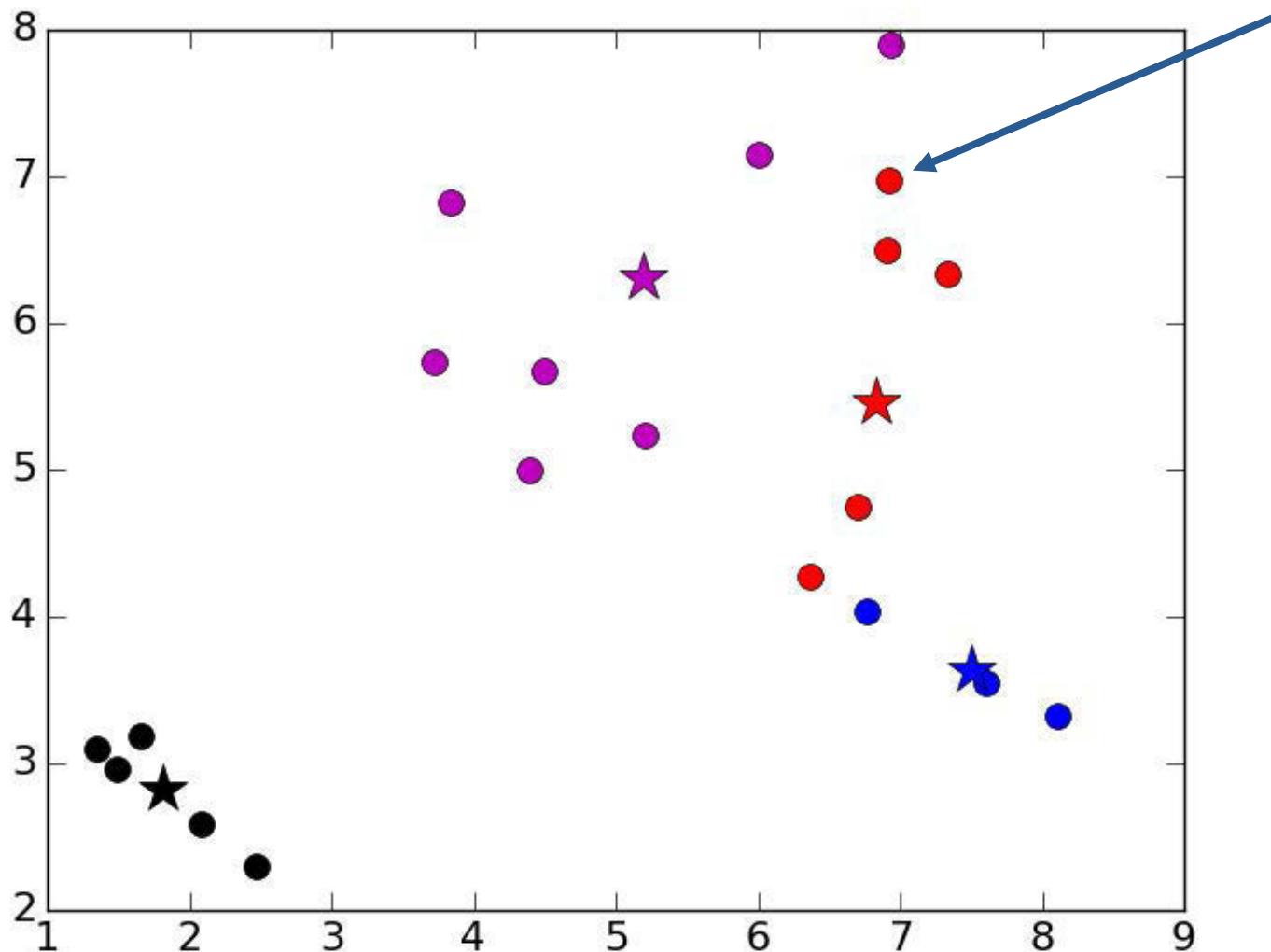
$K = 4$, Initial Centroids



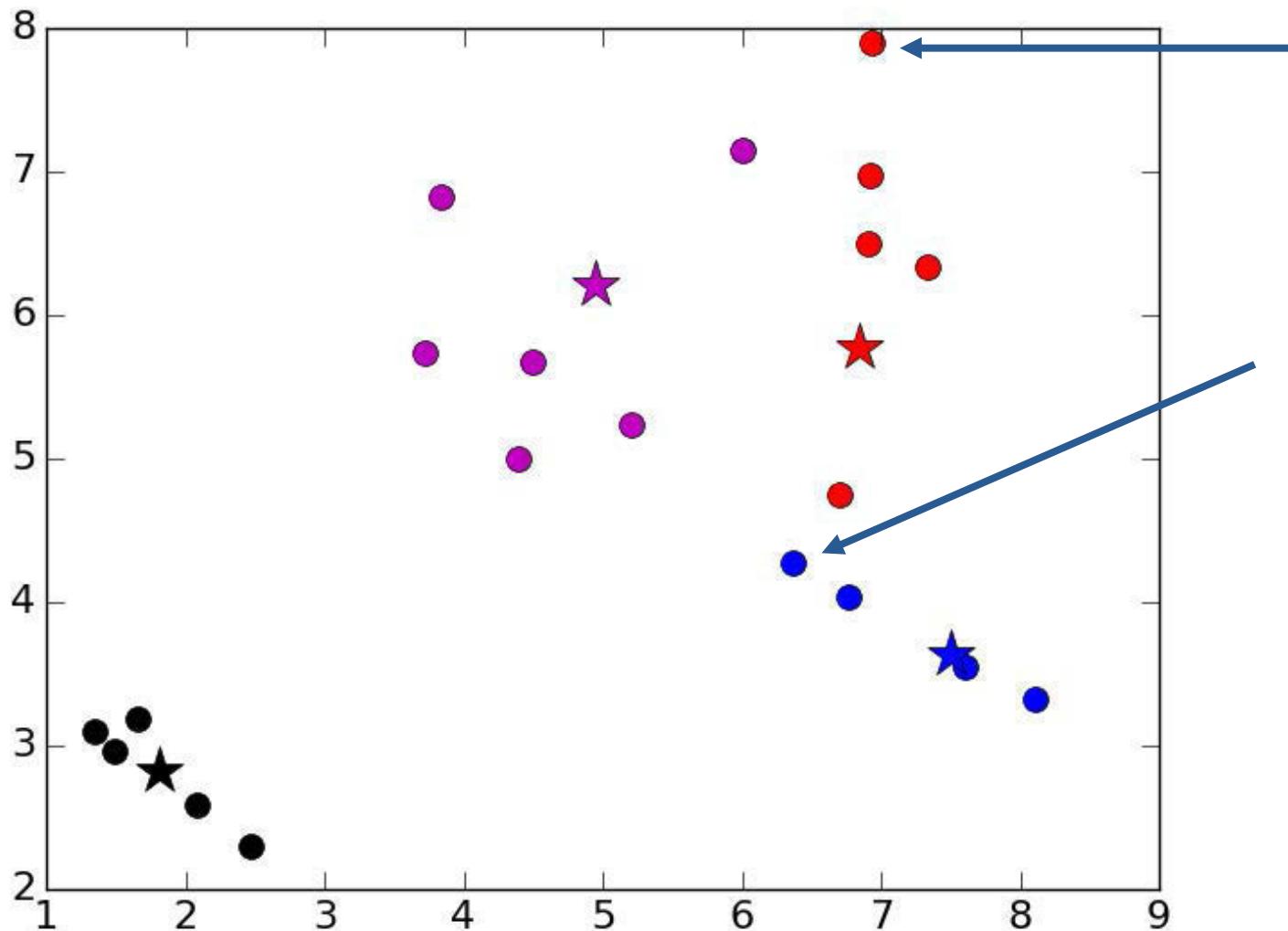
Iteration 1



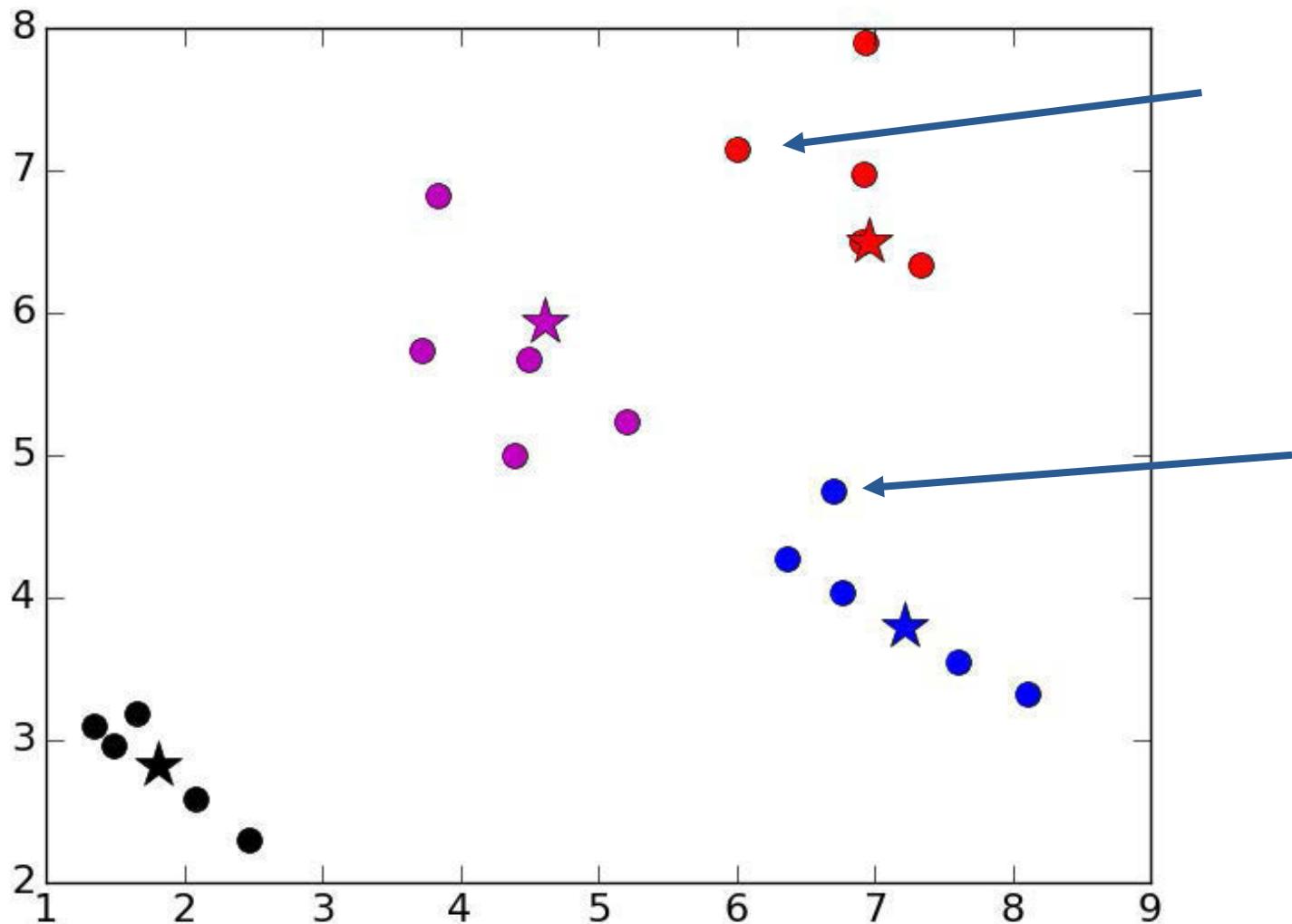
Iteration 2



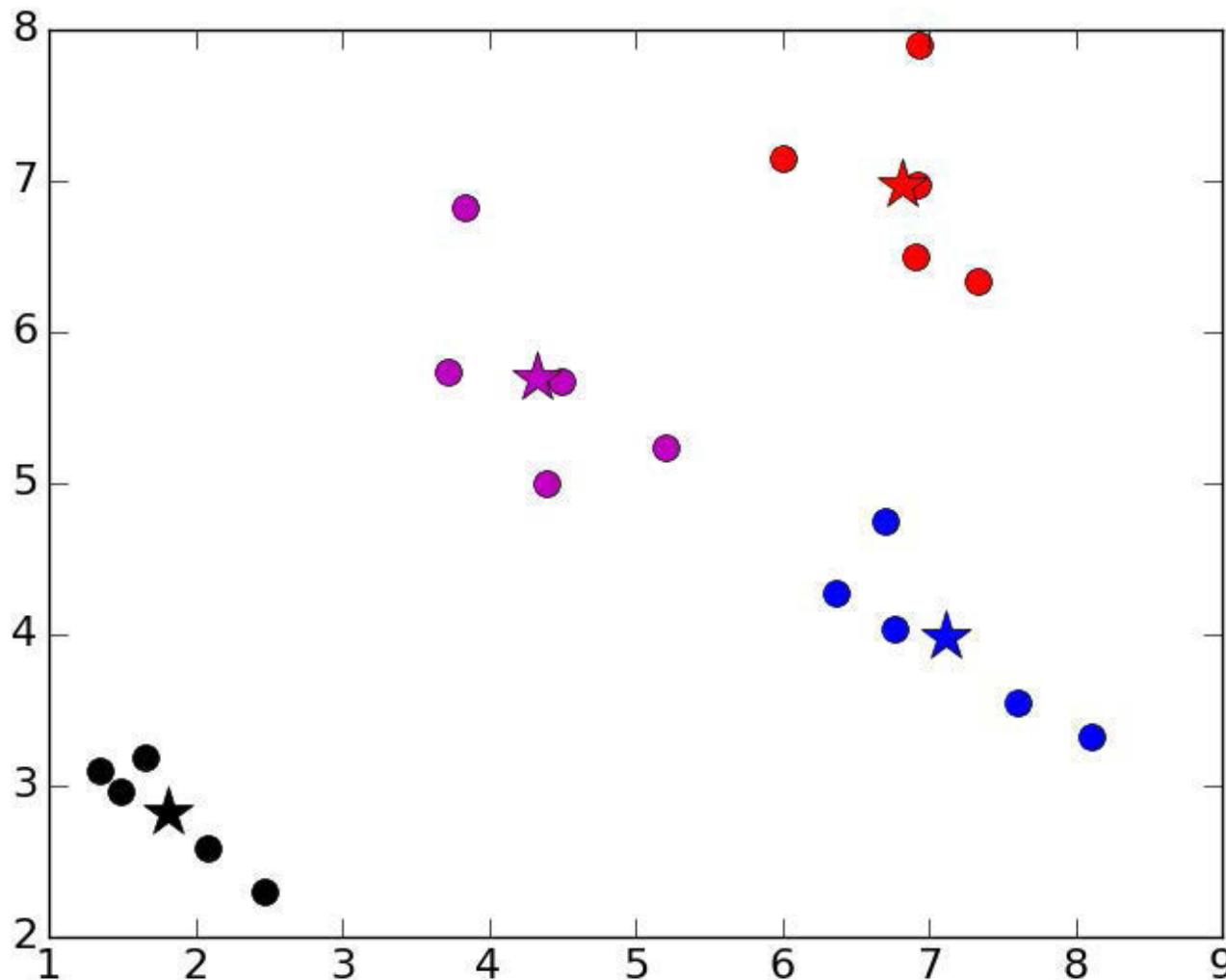
Iteration 3



Iteration 4

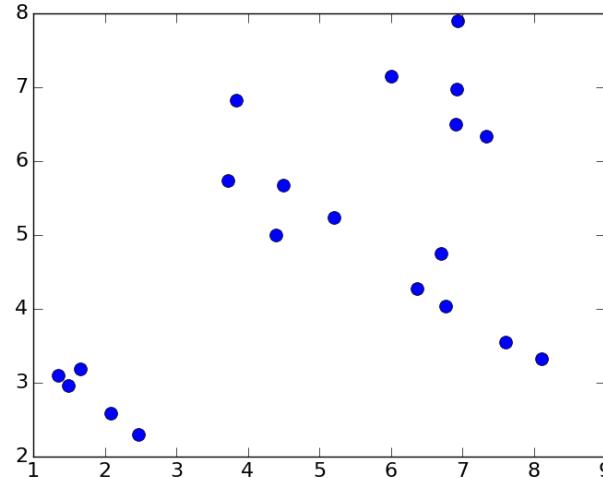


Iteration 5



Issues with k-means

- Choosing the “wrong” k can lead to strange results
 - Consider $k = 3$

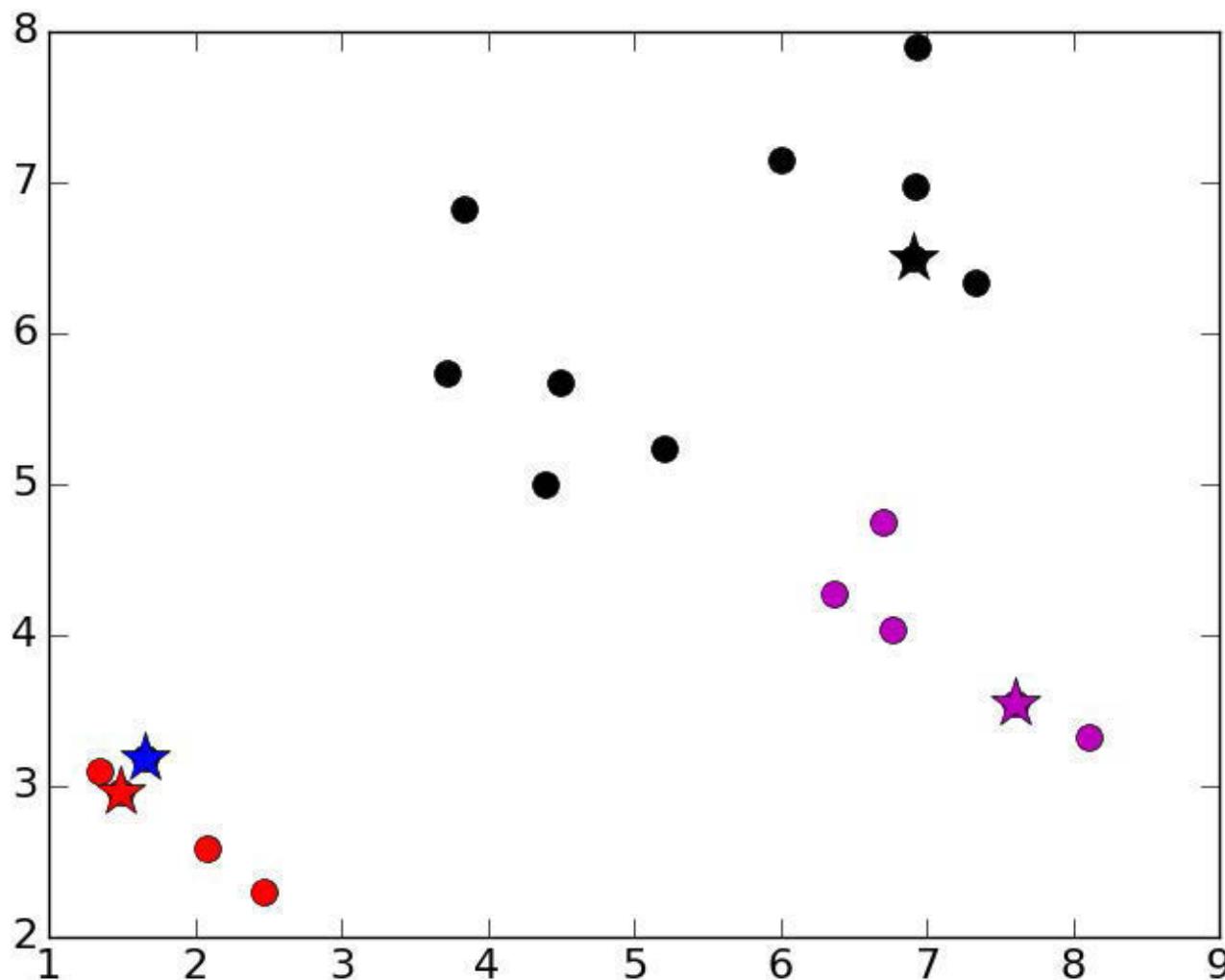


- Result can depend upon initial centroids
 - Number of iterations
 - Even final result
 - Greedy algorithm can find different local optimas

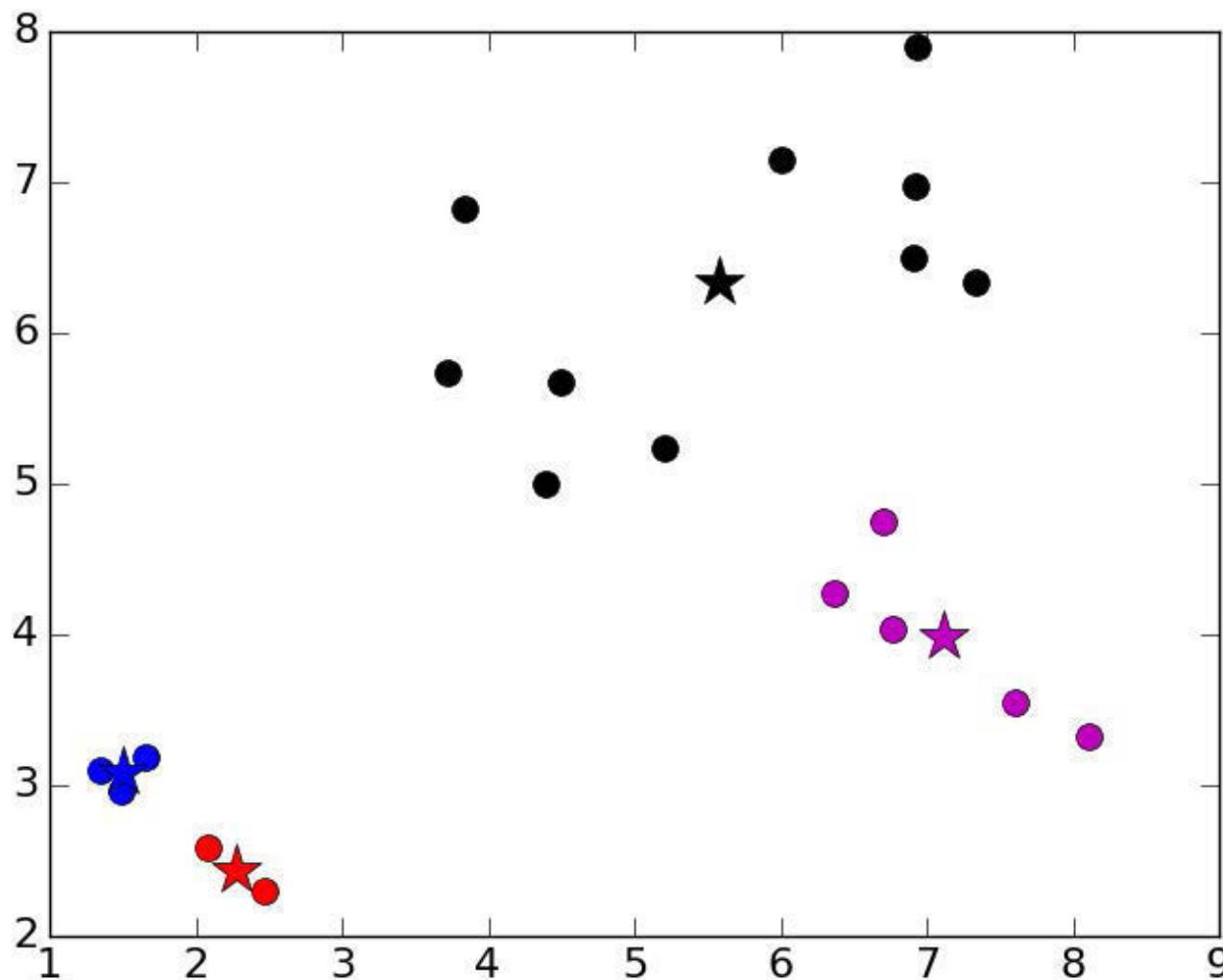
How to Choose K

- *A priori* knowledge about application domain
 - There are two kinds of people in the world: $k = 2$
 - There are five different types of bacteria: $k = 5$
- Search for a good k
 - Try different values of k and evaluate quality of results
 - Run hierarchical clustering on subset of data

Unlucky Initial Centroids



Converges On



Mitigating Dependence on Initial Centroids

Try multiple sets of randomly chosen initial centroids

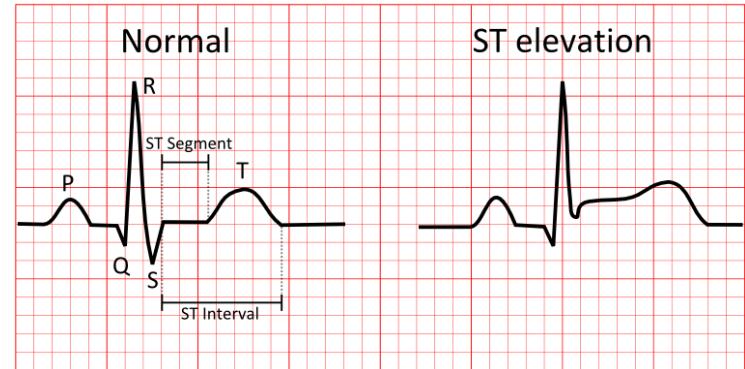
Select “best” result

```
best = kMeans(points)
for t in range(numTrials):
    C = kMeans(points)
    if dissimilarity(C) < dissimilarity(best):
        best = C
return best
```

An Example

- Many patients with 4 features each

- Heart rate in beats per minute
 - Number of past heart attacks
 - Age
 - ST elevation (binary)



- Outcome (death) based on features
 - Probabilistic, not deterministic
 - E.g., older people with multiple heart attacks at higher risk
- Cluster, and examine purity of clusters relative to outcomes

Data Sample

	HR	Att	STE	Age	Outcome
P000:	[89.	1.	0.	66.]	:1
P001:	[59.	0.	0.	72.]	:0
P002:	[73.	0.	0.	73.]	:0
P003:	[56.	1.	0.	65.]	:0
P004:	[75.	1.	1.	68.]	:1
P005:	[68.	1.	0.	56.]	:0
P006:	[73.	1.	0.	75.]	:1
P007:	[72.	0.	0.	65.]	:0
P008:	[73.	1.	0.	64.]	:1
P009:	[73.	0.	0.	58.]	:0
P010:	[100.	0.	0.	75.]	:0
P011:	[79.	0.	0.	31.]	:0
P012:	[81.	0.	0.	58.]	:0
P013:	[89.	1.	0.	50.]	:1
P014:	[81.	0.	0.	70.]	:0

Class Example

```
class Example(object):

    def __init__(self, name, features, label = None):
        #Assumes features is an array of floats
        self.name = name
        self.features = features
        self.label = label

    ...

    def distance(self, other):
        return minkowskiDist(self.features,
                             other.getFeatures(), 2)

    ...
```

Class Cluster

```
class Cluster(object):

    def __init__(self, examples):
        """Assumes examples a non-empty list of Examples"""
        ...

    def update(self, examples):
        """Assume examples is a non-empty list of Examples
           Replace examples; return amount centroid has
           changed"""
        ...

    def computeCentroid(self):
        vals = pylab.array([0.0]*self.examples[0].\
                           dimensionality())
        for e in self.examples: #compute mean
            vals += e.getFeatures()
        centroid = Example('centroid', vals/len(self.examples))
        return centroid
    ...
```

Class Cluster, cont.

```
def variability(self):
    totDist = 0
    for e in self.examples:
        totDist += (e.distance(self.centroid))**2
    return totDist

def members(self):
    for e in self.examples:
        yield e

...
```

Evaluating a Clustering

```
def dissimilarity(clusters):
    """Assumes clusters a list of clusters
       Returns a measure of the total dissimilarity of the
       clusters in the list"""
    totDist = 0
    for c in clusters:
        totDist += c.variability()
    return totDist
```

Patients

```
import cluster, pylab, numpy

class Patient(cluster.Example):
    pass

def scaleAttrs(vals):
    vals = pylab.array(vals)
    mean = sum(vals)/len(vals)
    sd = numpy.std(vals)
    vals = vals - mean
    return vals/sd

def getData(toScale = False):
    #read in data
    ...
    if toScale:
        hrList = scaleAttrs(hrList)
    ...
    #Build points
    ...
    return points
```

Z-Scaling

Mean = ?

Std = ?

kmeans

```
def kmeans(examples, k, verbose = False):
    #Get k randomly chosen initial centroids,
    #create cluster for each
    ...
    #Iterate until centroids do not change
    ...
    #Associate each example with closest centroid
    ...
    for c in newClusters: #Avoid having empty clusters
        if len(c) == 0:
            raise ValueError('Empty Cluster')

    #Update each cluster; check if a centroid has changed
    ...

def trykmeans(examples, numClusters, numTrials, verbose=False):
    """Calls kmeans numTrials times and returns the result with
       the lowest dissimilarity"""
    ...
```

Examining Results

```
def printClustering(clustering):
    """Assumes: clustering is a sequence of clusters
    Prints information about each cluster
    Returns list of fraction of pos cases in each cluster"""
    ...

def testClustering(patients, numClusters, seed = 0,
                   numTrials = 5):
    random.seed(seed)
    bestClustering = trykmeans(patients, numClusters,
                               numTrials)
    posFracs = printClustering(bestClustering)
    return posFracs

patients = getData()
for k in (2,):
    print('\n      Test k-means (k = ' + str(k) + ')')
    posFracs = testClustering(patients, k)
```

Result of Running It

Test k-means ($k = 2$)

Cluster of size 118 with fraction of positives = 0.3305

Cluster of size 132 with fraction of positives = 0.3333

Like it?

Try patients = getData(True)

Test k-means ($k = 2$)

Cluster of size 224 with fraction of positives = 0.2902

Cluster of size 26 with fraction of positives = 0.6923

Happy with sensitivity?

How Many Positives Are There?

```
numPos = 0
for p in patients:
    if p.getLabel() == 1:
        numPos += 1
print('Total number of positive patients =', numPos)
```

Total number of positive patients = 83

Test k-means ($k = 2$)

Cluster of size 224 with fraction of positives = 0.2902

Cluster of size 26 with fraction of positives = 0.6923

A Hypothesis

- Different subgroups of positive patients have different characteristics
- How might we test this?
- Try some other values of k

```
patients = getData()
for k in (2,4,6):
    print('\n      Test k-means (k = ' + str(k) + ')')
posFracs = testClustering(patients, k, 2)
```

Testing Multiple Values of k

Test k-means ($k = 2$)

Cluster of size 224 with fraction of positives = 0.2902

Cluster of size 26 with fraction of positives = 0.6923

Test k-means ($k = 4$)

Cluster of size 26 with fraction of positives = 0.6923

Cluster of size 86 with fraction of positives = 0.0814

Cluster of size 76 with fraction of positives = 0.7105

Cluster of size 62 with fraction of positives = 0.0645

Test k-means ($k = 6$)

Cluster of size 49 with fraction of positives = 0.0204

Cluster of size 26 with fraction of positives = 0.6923

Cluster of size 45 with fraction of positives = 0.0889

Cluster of size 54 with fraction of positives = 0.0926

Cluster of size 36 with fraction of positives = 0.7778

Cluster of size 40 with fraction of positives = 0.675

Pick a k

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Lecture 13: Classification

Announcements

- Reading
 - Chapter 24
 - Section 5.3.2 (list comprehension)
- Course evaluations
 - Online evaluation now through noon on Friday, December 16

Supervised Learning

- Regression

- Predict a real number associated with a feature vector
 - E.g., use linear regression to fit a curve to data

- *Classification*

- Predict a discrete value (label) associated with a feature vector

An Example (similar to earlier lecture)

Name	Features					Label
	Egg-laying	Scales	Poisonous	Cold-blooded	Number legs	Reptile
Cobra	1	1	1	1	0	1
Rattlesnake	1	1	1	1	0	1
Boa constrictor	0	1	0	1	0	1
Chicken	1	1	0	1	2	0
Guppy	0	1	0	0	0	0
Dart frog	1	0	1	0	4	0
Zebra	0	0	0	0	4	0
Python	1	1	0	1	0	1
Alligator	1	1	0	1	4	1

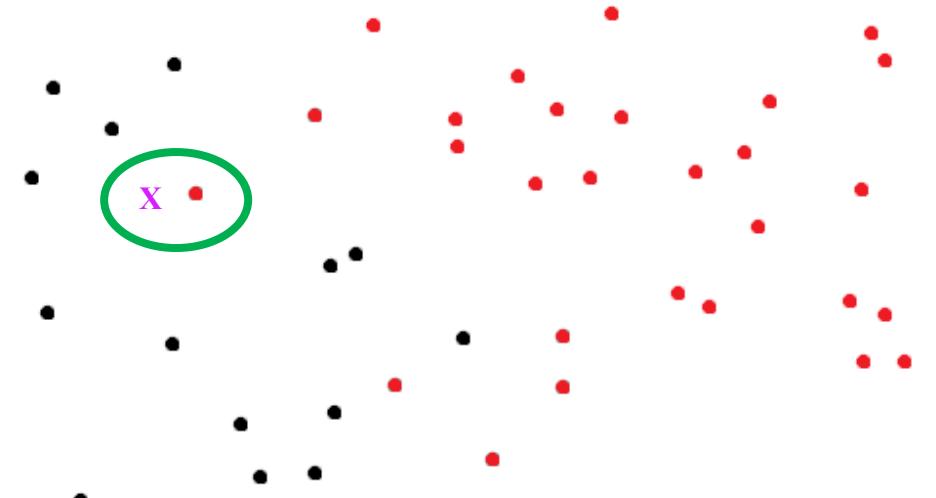
Distance Matrix

	cobra	rattlesnake	boa constrictor	chicken	guppy	dart frog	zebra	python	alligator
cobra	--	0.0	1.414	2.236	1.732	1.732	2.236	1.0	1.414
rattlesnake	0.0	--	1.414	2.236	1.732	1.732	2.236	1.0	1.414
boa constrictor	1.414	1.414	--	2.236	1.0	2.236	1.732	1.0	1.414
chicken	2.236	2.236	2.236	--	2.449	2.0	2.0	2.0	1.0
guppy	1.732	1.732	1.0	2.449	--	2.0	1.414	1.414	1.732
dart frog	1.732	1.732	2.236	2.0	2.0	--	1.414	2.0	1.732
zebra	2.236	2.236	1.732	2.0	1.414	1.414	--	2.0	1.732
python	1.0	1.0	1.0	2.0	1.414	2.0	2.0	--	1.0
alligator	1.414	1.414	1.414	1.0	1.732	1.732	1.732	1.0	--

Code for producing this table posted

Using Distance Matrix for Classification

- Simplest approach is probably **nearest neighbor**
- Remember training data
- When predicting the label of a new example
 - Find the nearest example in the training data
 - Predict the label associated with that example



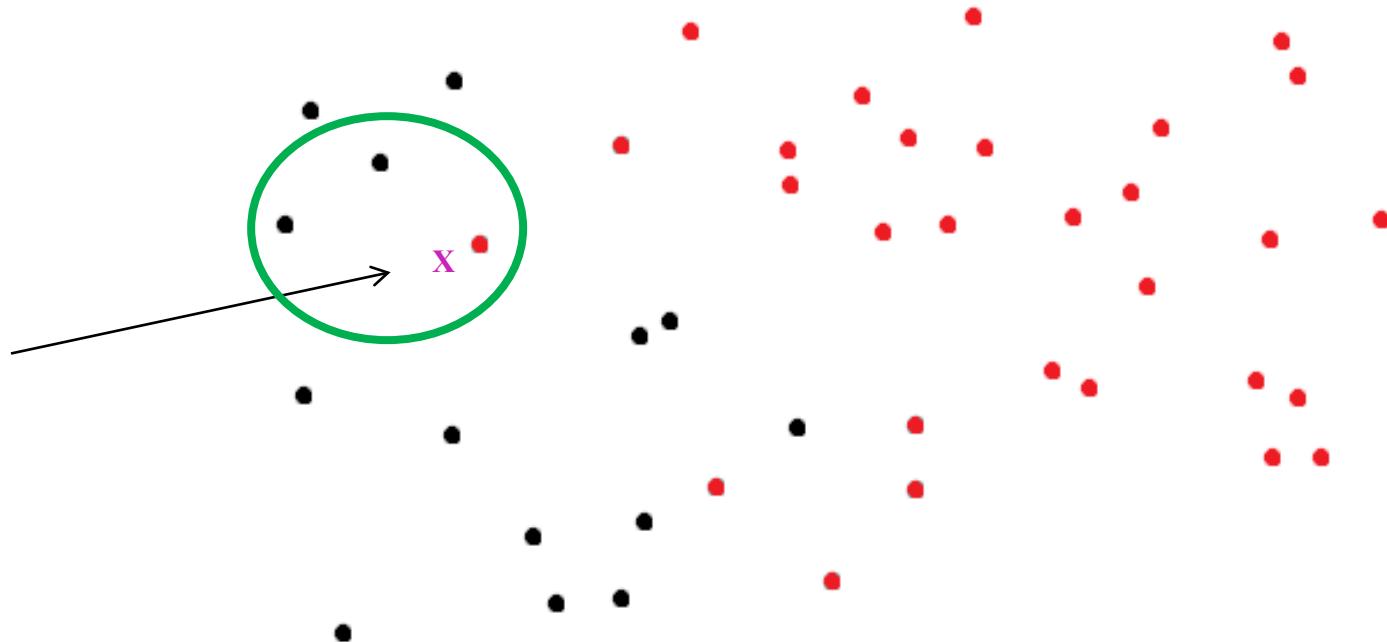
Distance Matrix

	cobra	rattlesnake	boa constrictor	chicken	guppy	dart frog	zebra	python	alligator	Label
cobra	-	0.0	1.414	2.236	1.732	1.732	2.236	1.0	1.414	R
rattlesnake	0.0	-	1.414	2.236	1.732	1.732	2.236	1.0	1.414	R
boa constrictor	1.414	1.414	-	2.236	1.0	2.236	1.732	1.0	1.414	R
chicken	2.236	2.236	2.236	--	2.449	2.0	2.0	2.0	1.0	~R
guppy	1.732	1.732	1.0	2.449	--	2.0	1.414	1.414	1.732	~R
dart frog	1.732	1.732	2.236	2.0	2.0	--	1.414	2.0	1.732	~R
zebra	2.236	2.236	1.732	2.0	1.414	1.414				
python	1.0	1.0	1.0	2.0	1.414	2.0				
alligator	1.414	1.414	1.414	1.0	1.732	1.732				

An Example

0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9

K-nearest Neighbors



An Example

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

Advantages and Disadvantages of KNN

■ Advantages

- Learning fast, no explicit training
- No theory required
- Easy to explain method and results

■ Disadvantages

- Memory intensive and predictions can take a long time
 - Are better algorithms than brute force
- No model to shed light on process that generated data

The Titanic Disaster

- RMS Titanic sank in the North Atlantic the morning of 15 April 1912, after colliding with an iceberg. Of the 1,300 passengers aboard, 812 died. (703 of 918 crew members died.)
- Database of 1046 passengers
 - Cabin class
 - 1st, 2nd, 3rd
 - Age
 - Gender

Is Accuracy Enough

- If we predict “died”, accuracy will be >62% or passenger and >76% for crew members
- Consider a disease that occurs in 0.1% of population
 - Predicting disease-free has an accuracy of 0.999

Other Metrics

$$sensitivity = \frac{true\ positive}{true\ positive + false\ negative}$$

$$specificity = \frac{true\ negative}{true\ negative + false\ positive}$$

$$positive\ predictive\ value = \frac{true\ positive}{true\ positive + false\ positive}$$

$$negative\ predictive\ value = \frac{true\ negative}{true\ negative + false\ negative}$$

sensitivity = recall

specificity = precision

Testing Methodology Matters

- Leave-one-out
- Repeated random subsampling

Leave-one-out

```
def leaveOneOut(examples, method, toPrint = True):
    truePos, falsePos, trueNeg, falseNeg = 0, 0, 0, 0
    for i in range(len(examples)):
        testCase = examples[i]
        trainingData = examples[0:i] + examples[i+1:]
        results = method(trainingData, [testCase])
        truePos += results[0]
        falsePos += results[1]
        trueNeg += results[2]
        falseNeg += results[3]
    if toPrint:
        getStats(truePos, falsePos, trueNeg, falseNeg)
    return truePos, falsePos, trueNeg, falseNeg
```

Repeated Random Subsampling

```
def split80_20(examples):
    sampleIndices = random.sample(range(len(examples)),
                                   len(examples)//5)
    trainingSet, testSet = [], []
    for i in range(len(examples)):
        if i in sampleIndices:
            testSet.append(examples[i])
        else:
            trainingSet.append(examples[i])
    return trainingSet, testSet
```

Repeated Random Subsampling

```
def randomSplits(examples, method, numSplits, toPrint = True):  
    truePos, falsePos, trueNeg, falseNeg = 0, 0, 0, 0  
    random.seed(0)  
    for t in range(numSplits):  
        trainingSet, testSet = split80_20(examples)  
        results = method(trainingSet, testSet)  
        truePos += results[0]  
        falsePos += results[1]  
        trueNeg += results[2]  
        falseNeg += results[3]  
    getStats(truePos/numSplits, falsePos/numSplits,  
            trueNeg/numSplits, falseNeg/numSplits, toPrint)  
    return truePos/numSplits, falsePos/numSplits,\n           trueNeg/numSplits, falseNeg/numSplits
```

Let's Try KNN

```
def KNearestClassify(training, testSet, label, k):
    """Assumes training & testSet lists of examples, k an int
    Predicts whether each example in testSet has label
    Returns number of true positives, false positives,
    true negatives, and false negatives"""

knn = lambda training, testSet:\n    KNearestClassify(training, testSet,\n                      'Survived', 3)

numSplits = 10
print('Average of', numSplits,
      '80/20 splits using KNN (k=3)')
truePos, falsePos, trueNeg, falseNeg =\
    randomSplits(examples, knn, numSplits)

print('Average of LOO testing using KNN (k=3)')
truePos, falsePos, trueNeg, falseNeg =\
    leaveOneOut(examples, knn)
```

Results

Average of 10 80/20 splits using KNN (k=3)

Accuracy = 0.766

Sensitivity = 0.67

Specificity = 0.836

Pos. Pred. Val. = 0.747

Average of LOO testing using KNN (k=3)

Accuracy = 0.769

Sensitivity = 0.663

Specificity = 0.842

Pos. Pred. Val. = 0.743

Considerably better than 62%

Not much difference between experiments

Logistic Regression

- Analogous to linear regression
- Designed explicitly for predicting **probability** of an event
 - Dependent variable can only take on a finite set of values
 - Usually 0 or 1
- Finds **weights** for each feature
 - Positive implies variable positively correlated with outcome
 - Negative implies variable negatively correlated with outcome
 - Absolute magnitude related to strength of the correlation
- Optimization problem a bit complex, key is use of a log function—won't make you look at it

Class LogisticRegression

```
import sklearn.linear_model
```

`fit(sequence of feature vectors, sequence of labels)`

 Returns object of type LogisticRegression

`coef_`

 Returns weights of features

`predict_proba(feature vector)`

 Returns probabilities of labels

Building a Model

```
def buildModel(examples, toPrint = True):
    featureVecs, labels = [], []
    for e in examples:
        featureVecs.append(e.getFeatures())
        labels.append(e.getLabel())
    LogisticRegression = sklearn.linear_model.LogisticRegression
    model = LogisticRegression().fit(featureVecs, labels)
    if toPrint:
        ...
return model
```

Applying Model

```
def applyModel(model, testSet, label, prob = 0.5):
    testFeatureVecs = [e.getFeatures() for e in testSet]
    probs = model.predict_proba(testFeatureVecs)
    truePos, falsePos, trueNeg, falseNeg = 0, 0, 0, 0
    for i in range(len(probs)):
        if probs[i][1] > prob:
            if testSet[i].getLabel() == label:
                truePos += 1
            else:
                falsePos += 1
        else:
            if testSet[i].getLabel() != label:
                trueNeg += 1
            else:
                falseNeg += 1
    return truePos, falsePos, trueNeg, falseNeg
```

List Comprehension

`expr for id in L`

Creates a list by evaluating `expr` `len(L)` times with
`id` in `expr` replaced by each element of `L`

```
L = [x*x for x in range(10)]
print(L)
L = [x*x for x in range(10) if x%2 == 0]
print(L)
```

Applying Model

```
def applyModel(model, testSet, label, prob = 0.5):
    testFeatureVecs = [e.getFeatures() for e in testSet]
    probs = model.predict_proba(testFeatureVecs)
    truePos, falsePos, trueNeg, falseNeg = 0, 0, 0, 0
    for i in range(len(probs)):
        if probs[i][1] > prob:
            if testSet[i].getLabel() == label:
                truePos += 1
            else:
                falsePos += 1
        else:
            if testSet[i].getLabel() != label:
                trueNeg += 1
            else:
                falseNeg += 1
    return truePos, falsePos, trueNeg, falseNeg
```

Putting It Together

```
def lr(trainingData, testData, prob = 0.5):
    model = buildModel(trainingData, False)
    results = applyModel(model, testData, 'Survived', prob)
    return results

numSplits = 10
print('Average of', numSplits, '80/20 splits LR')
truePos, falsePos, trueNeg, falseNeg =\
    divide80_20(examples, lr, numSplits)

print('Average of L00 testing using LR')
truePos, falsePos, trueNeg, falseNeg =\
    leaveOneOut(examples, lr)
```

Results

Average of 10 80/20 splits LR

Accuracy = 0.804

Sensitivity = 0.719

Specificity = 0.859

Pos. Pred. Val. = 0.767

Average of LOO testing using LR

Accuracy = 0.786

Sensitivity = 0.705

Specificity = 0.842

Pos. Pred. Val. = 0.754

Compare to KNN Results

Average of 10 80/20 splits using KNN (k=3)

Accuracy = 0.744

Sensitivity = 0.629

Specificity = 0.829

Pos. Pred. Val. = 0.728

Average of LOO testing using KNN (k=3)

Accuracy = 0.769

Sensitivity = 0.663

Specificity = 0.842

Pos. Pred. Val. = 0.743

Average of 10 80/20 splits LR

Accuracy = 0.804

Sensitivity = 0.719

Specificity = 0.859

Pos. Pred. Val. = 0.767

Average of LOO testing using LR

Accuracy = 0.786

Sensitivity = 0.705

Specificity = 0.842

Pos. Pred. Val. = 0.754

Performance not much difference

Logistic regression slightly better

Also provides insight about variables

Looking at Feature Weights

```
def buildModel(examples, toPrint = True):  
    ...  
    if toPrint:  
        print('model.classes_ =', model.classes_)  
        for i in range(len(model.coef_)):  
            print('For label', model.classes_[1])  
            for j in range(len(model.coef_[0])):  
                print('    ', Passenger.featureNames[j], '=',  
                      model.coef_[0][j])  
  
    return model  
  
buildModel(examples, True)
```

Be wary of reading too
much into the weights

Features are often
correlated

model.classes_ = ['Died' 'Survived']

For label Survived

C1 = 1.66761946545

C2 = 0.460354552452

C3 = -0.50338282535

age = -0.0314481062387

male gender = -2.39514860929

Changing the Cutoff

```
random.seed(0)
trainingSet, testSet = split80_20(examples)
model = buildModel(trainingSet, False)
print('Try p = 0.1')
truePos, falsePos, trueNeg, falseNeg = \
    applyModel(model, testSet, 'Survived', 0.1)
getStats(truePos, falsePos, trueNeg, falseNeg)
print('Try p = 0.9')
truePos, falsePos, trueNeg, falseNeg = \
    applyModel(model, testSet, 'Survived', 0.9)
getStats(truePos, falsePos, trueNeg, falseNeg)
```

Try p = 0.1

Accuracy = 0.493

Sensitivity = 0.976

Specificity = 0.161

Pos. Pred. Val. = 0.444

Try p = 0.9

Accuracy = 0.656

Sensitivity = 0.176

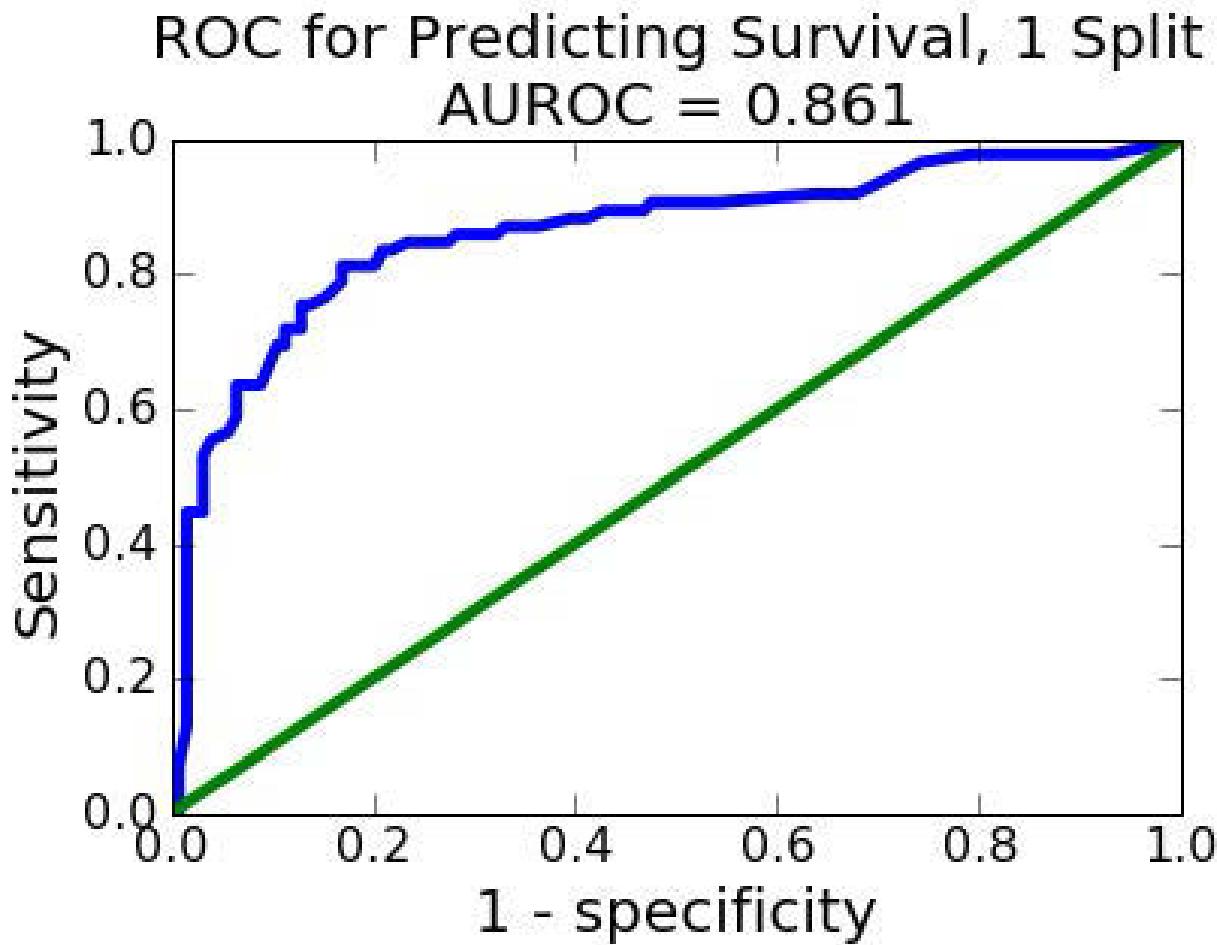
Specificity = 0.984

Pos. Pred. Val. = 0.882

ROC (Receiver Operating Characteristic)

```
def buildROC(trainingSet, testSet, title, plot = True):
    model = buildModel(trainingSet, True)
    xVals, yVals = [], []
    p = 0.0
    while p <= 1.0:
        truePos, falsePos, trueNeg, falseNeg = \
            applyModel(model, testSet,
                       'Survived', p)
        xVals.append(1.0 - specificity(trueNeg, falsePos))
        yVals.append(sensitivity(truePos, falseNeg))
        p += 0.01
    auroc = sklearn.metrics.auc(xVals, yVals, True)
    if plot:
        ...
return auroc
```

Output



MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Lecture 14: Classification, Statistical Sins

Announcements

- Reading
 - Chapter 21
- Course evaluations
 - Online evaluation now through noon on Friday, December 16
- Will be making study code for final exam available later today

Compare to KNN Results (from Monday)

Average of 10 80/20 splits using KNN (k=3)

Accuracy = 0.744

Sensitivity = 0.629

Specificity = 0.829

Pos. Pred. Val. = 0.728

Average of LOO testing using KNN (k=3)

Accuracy = 0.769

Sensitivity = 0.663

Specificity = 0.842

Pos. Pred. Val. = 0.743

Average of 10 80/20 splits LR

Accuracy = 0.804

Sensitivity = 0.719

Specificity = 0.859

Pos. Pred. Val. = 0.767

Average of LOO testing using LR

Accuracy = 0.786

Sensitivity = 0.705

Specificity = 0.842

Pos. Pred. Val. = 0.754

Performance not much difference

Logistic regression slightly better

Logistic regression provides insight about variables

Looking at Feature Weights

```
model.classes_ = ['Died' 'Survived']
```

For label Survived

C1 = 1.66761946545

C2 = 0.460354552452

C3 = -0.50338282535

age = -0.0314481062387

male gender = -2.39514860929

Be wary of reading too
much into the weights

Features are often
correlated

L1 regression tends to drive one variable to zero

L2 (default) regression spreads weights across variables

Correlated Features, an Example

- $c_1 + c_2 + c_3 = 1$
 - I.e., values are not independent
 - Is being in 1st class good, or being in the other classes bad?
- Suppose we eliminate c_1 ?

```
def __init__(self, pClass, age, gender, survived, name):  
    self.name = name  
    if pClass == 2:  
        self.featureVec = [1, 0, age, gender]  
    elif pClass == 3:  
        self.featureVec = [0, 1, age, gender]  
    else:  
        self.featureVec = [0, 0, age, gender]  
    self.label = survived  
    self.cabinClass = pClass
```

Comparative Results

Original Features

Average of 20 80/20 splits LR

Accuracy = 0.778

Sensitivity = 0.687

Specificity = 0.842

Pos. Pred. Val. = 0.755

model.classes_ = ['Died' 'Survived']

For label Survived

C1 = 1.68864047459

C2 = 0.390605976351

C3 = -0.46270349333

age = -0.0307090135358

male gender = -2.41191131088

Modified Features

Average of 20 80/20 splits LR

Accuracy = 0.779

Sensitivity = 0.674

Specificity = 0.853

Pos. Pred. Val. = 0.765

model.classes_ = ['Died' 'Survived']

For label Survived

C2 = -1.08356816806

C3 = -1.92251427055

age = -0.026056041377

male gender = -2.36239279331

Changing the Cutoff

```
random.seed(0)
trainingSet, testSet = split80_20(examples)
model = buildModel(trainingSet, False)
print('Try p = 0.1')
truePos, falsePos, trueNeg, falseNeg = \
    applyModel(model, testSet, 'Survived', 0.1)
getStats(truePos, falsePos, trueNeg, falseNeg)
print('Try p = 0.9')
truePos, falsePos, trueNeg, falseNeg = \
    applyModel(model, testSet, 'Survived', 0.9)
getStats(truePos, falsePos, trueNeg, falseNeg)
```

Try p = 0.1

Accuracy = 0.493

Sensitivity = 0.976

Specificity = 0.161

Pos. Pred. Val. = 0.444

Try p = 0.9

Accuracy = 0.656

Sensitivity = 0.176

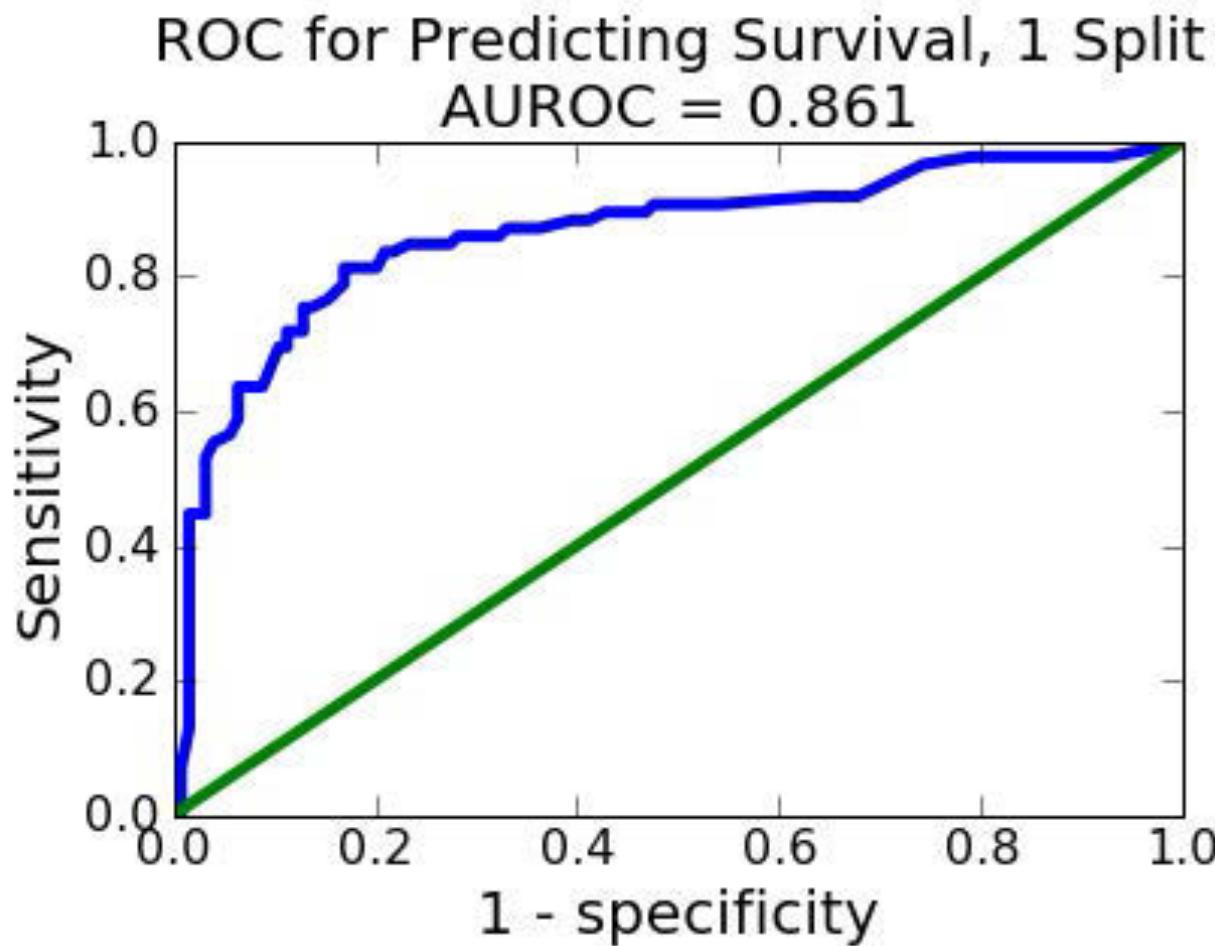
Specificity = 0.984

Pos. Pred. Val. = 0.882

ROC (Receiver Operating Characteristic)

```
def buildROC(trainingSet, testSet, title, plot = True):
    model = buildModel(trainingSet, True)
    xVals, yVals = [], []
    p = 0.0
    while p <= 1.0:
        truePos, falsePos, trueNeg, falseNeg = \
            applyModel(model, testSet,
                       'Survived', p)
        xVals.append(1.0 - specificity(trueNeg, falsePos))
        yVals.append(sensitivity(truePos, falseNeg))
        p += 0.01
    auroc = sklearn.metrics.auc(xVals, yVals, True)
    if plot:
        ...
return auroc
```

Output



There are Three Kinds of Lies

LIES

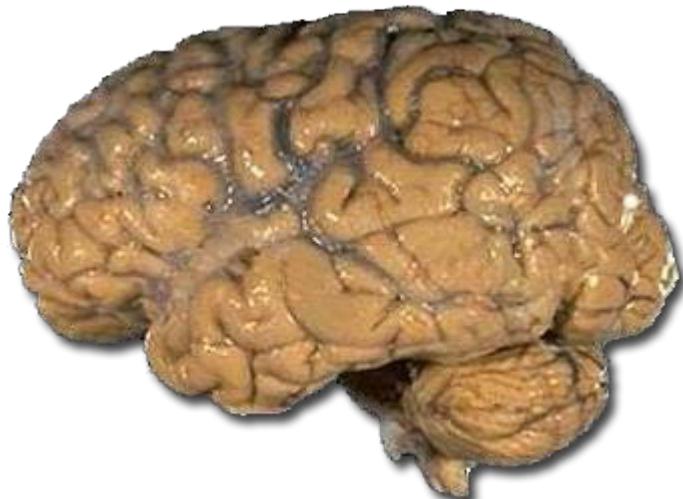
DAMNED LIES

and

STATISTICS

Humans and Statistics

Human Mind



Statistics

$$i = \text{len}(\text{observed}) - 1$$
$$\sum_{i=0}^{i=\text{len}(\text{observed})-1} (\text{observed}[i] - \text{predicted}[i])^2$$
$$1 - \left(\frac{n-1}{n} * \frac{n-2}{n} * \dots * \frac{n-(K-1)}{n} \right)$$

Image of brain © source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.

Humans and Statistics

“If you can't prove what you want to prove, demonstrate something else and pretend they are the same thing. In the daze that follows the collision of statistics with the human mind, hardly anyone will notice the difference.” – *Darrell Huff*

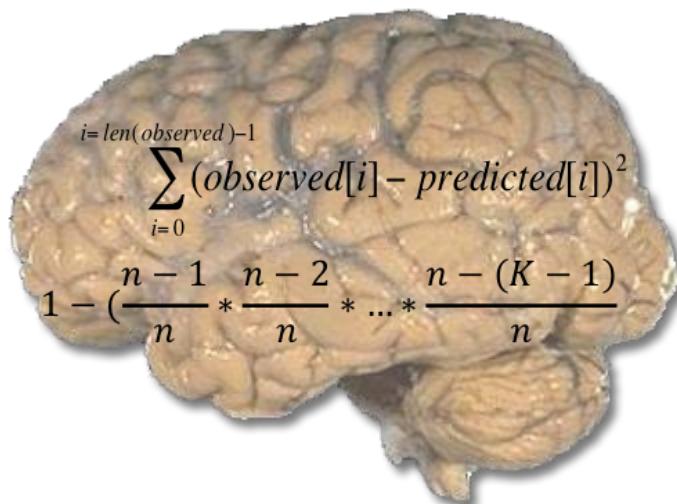


Image of brain © source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.

Anscombe's Quartet

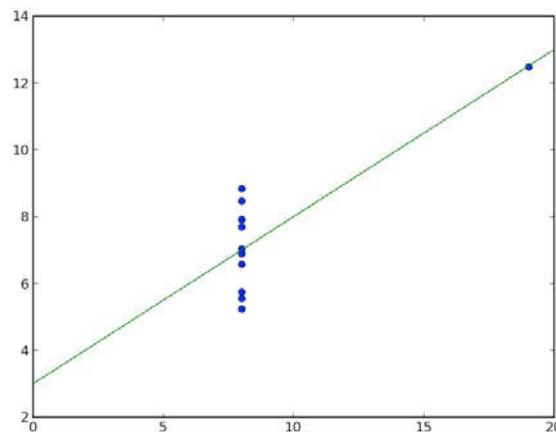
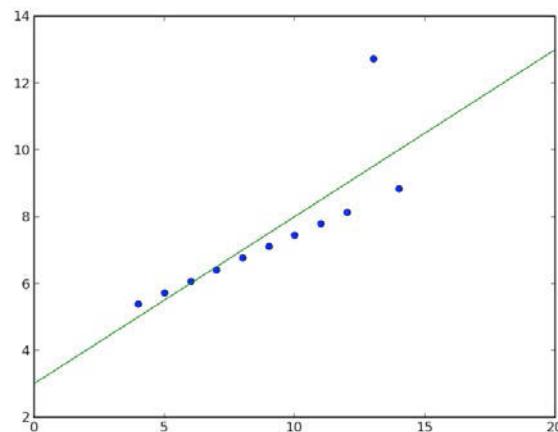
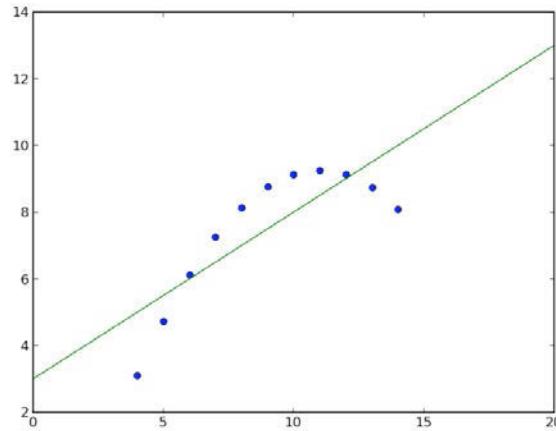
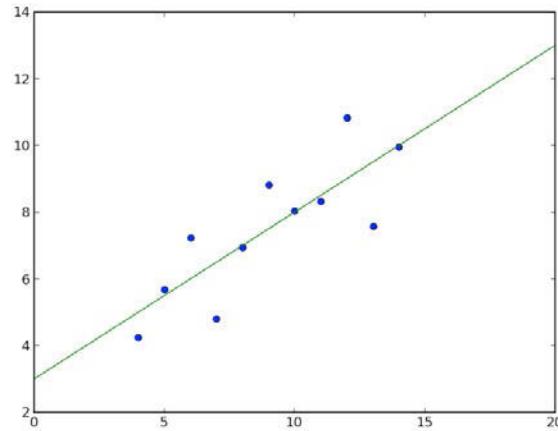
- Four groups each containing 11 x, y pairs

x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

Summary Statistics

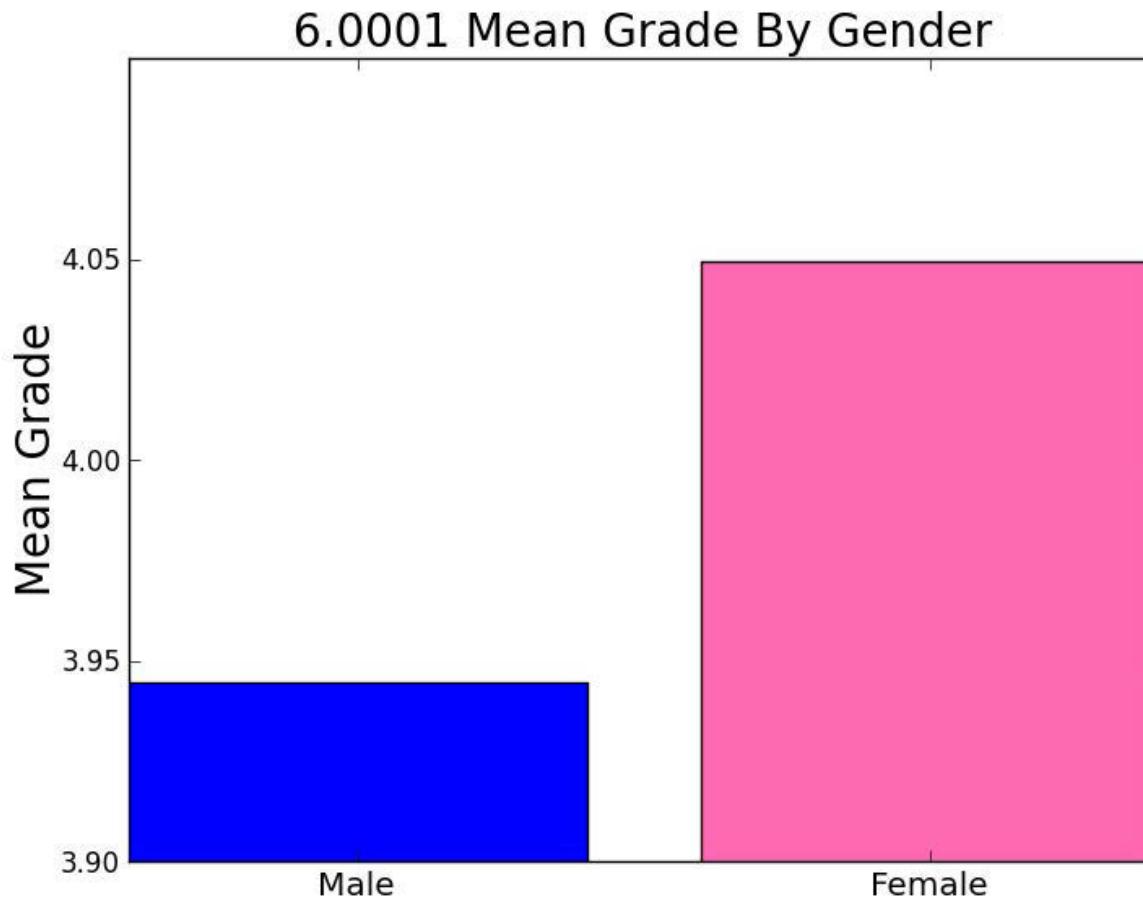
- Summary statistics for groups identical
 - Mean $x = 9.0$
 - Mean $y = 7.5$
 - Variance of $x = 10.0$
 - Variance of $y = 3.75$
 - Linear regression model: $y = 0.5x + 3$
- Are four data sets really similar?

Let's Plot the Data

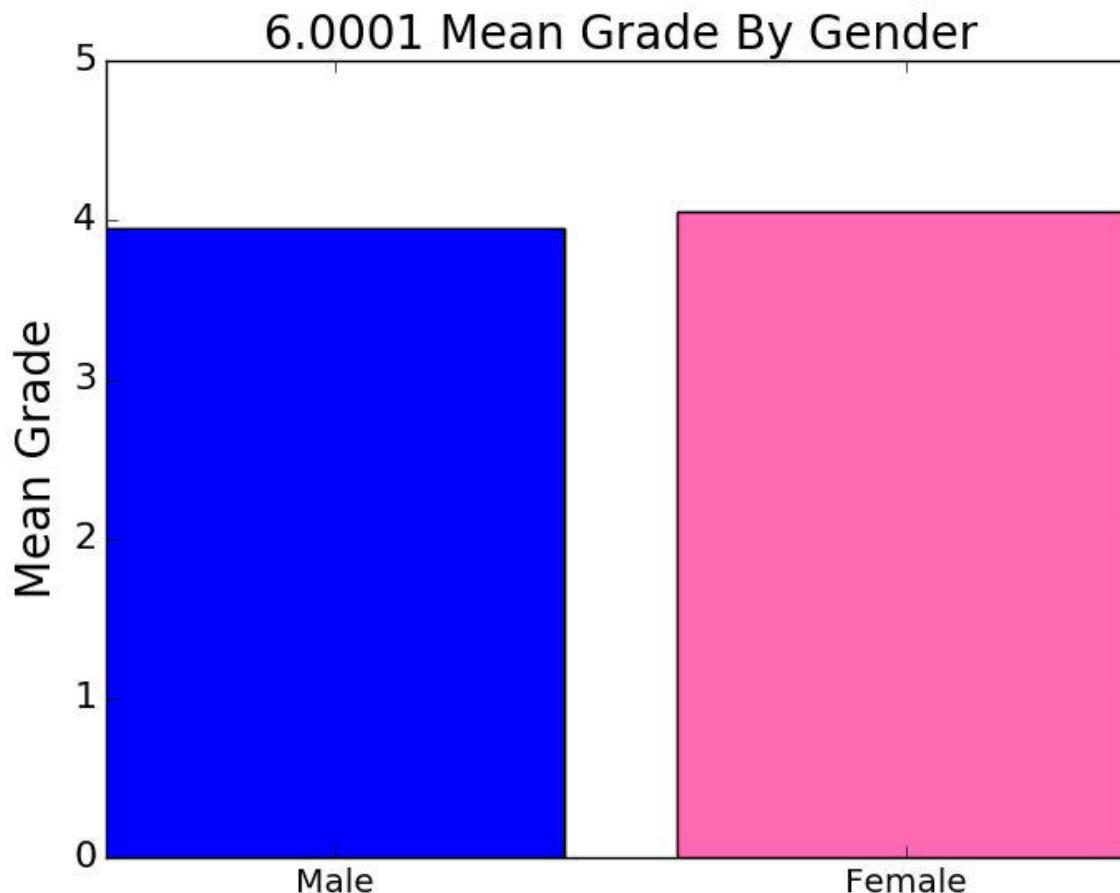


Moral: Statistics about the data is not the same as the data
Moral: Use visualization tools to look at the data itself

Lying with Pictures



Telling the Truth with Pictures



Moral: Look carefully at the axes labels and scales

Lying with Pictures



Moral: Ask whether the things being compared are actually comparable

Screenshot of Fox News © 20th / 21st Century Fox. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <https://ocw.mit.edu/help/faq-fair-use/>.

Garbage In, Garbage Out

“On two occasions I have been asked [by members of Parliament], ‘Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?’ I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.” – Charles Babbage (1791-1871)

Calhoun's Response to Errors in Data

“there were so many errors they balanced one another, and led to the same conclusion as if they were all correct.”

Was it the case that the measurement errors are unbiased and independent of each other, and therefore almost identically distributed on either side of the mean?

No, later analysis showed that the errors were not random but systematic.

“it was the census that was insane and not the colored people.”—
James Freeman Clarke

Moral: Analysis of bad data can lead to dangerous conclusions.

Sampling

- All statistical techniques are based upon the assumption that by sampling a subset of a population we can infer things about the population as a whole
- As we have seen, *if random sampling is used*, one can make meaningful mathematical statements about the expected relation of the sample to the entire population
- Easy to get random samples in simulations
- Not so easy in the field, where some examples are more convenient to acquire than others

Non-representative Sampling

- “Convenience sampling” not usually random, e.g.,
 - Survivor bias, e.g., course evaluations at end of course or grading final exam in 6.0002 on a strict curve
 - Non-response bias, e.g., opinion polls conducted by mail or online
- When samples not random and independent, we can still do things like computer means and standard deviations, but **we should not draw conclusions from them** using things like the empirical rule and central limit theorem.
- Moral: Understand how data was collected, and whether assumptions used in the analysis are satisfied. If not, be wary.

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.

Lecture 15: Statistical Sins and Wrapup

Announcements

- Course evaluations
 - Online evaluation now through noon on Friday, December 16
- Final exam on Monday!

Global Warming, Fact or Fiction

Average global temperature, 1880 to 2014

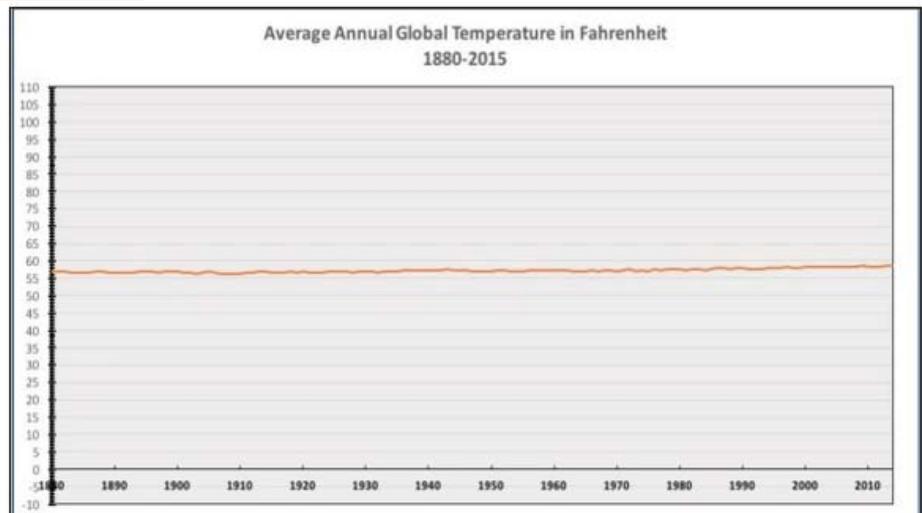
60° fahrenheit



ATLAS | Data: NASA

On Monday I said, beware of charts where the y-axis doesn't start at zero

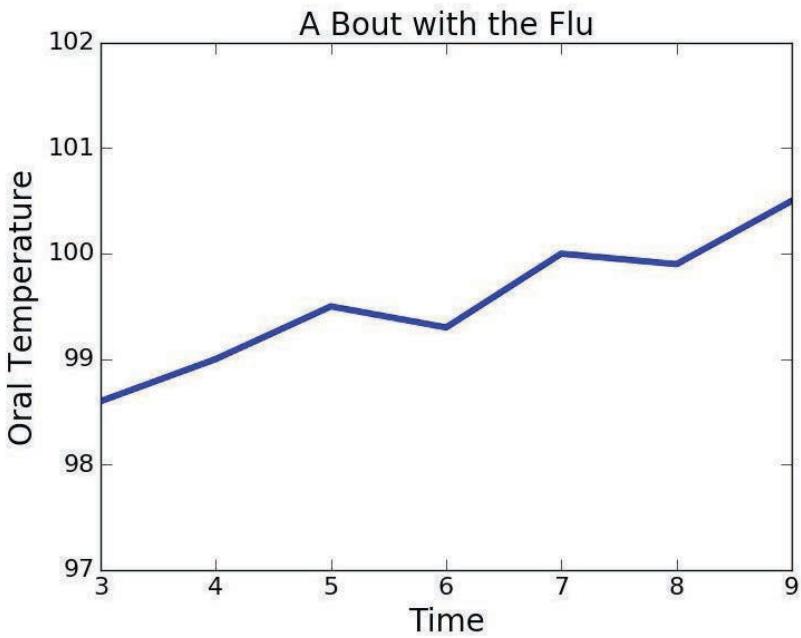
Which conveys a more accurate impression?



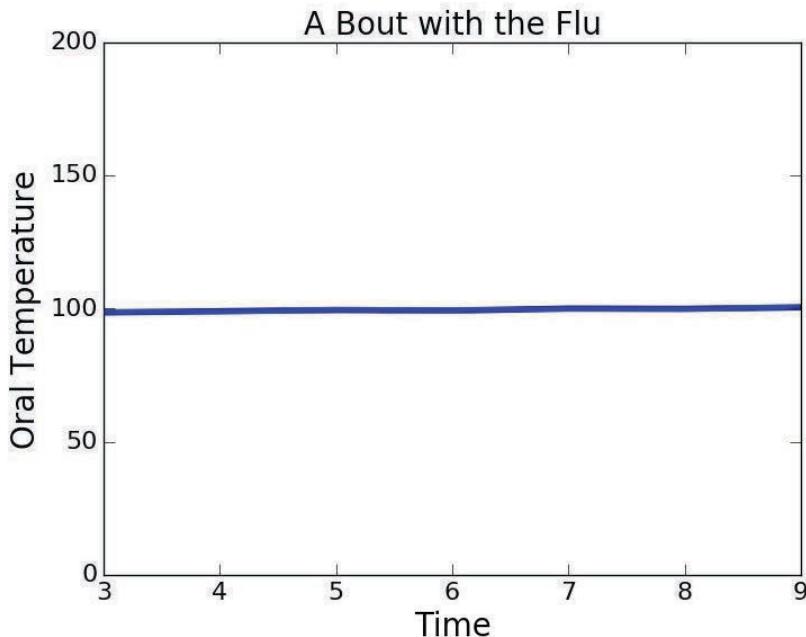
National Review, December 2015

Graph © National Review. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>

Fever and the Flu



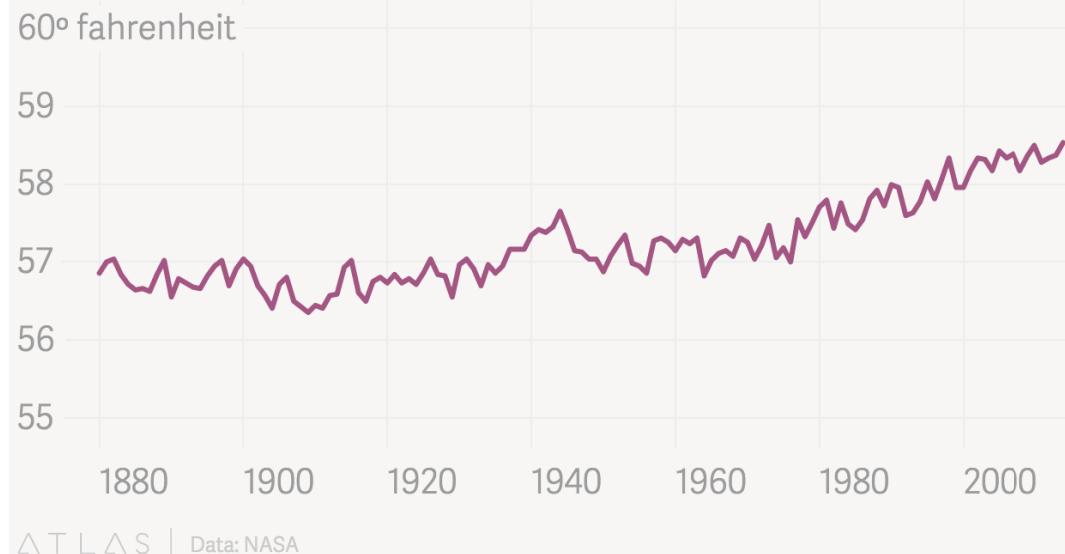
Moral: Truncate the y-axis to eliminate preposterous values.



The Myth of Global Warming



Average global temperature, 1880 to 2014

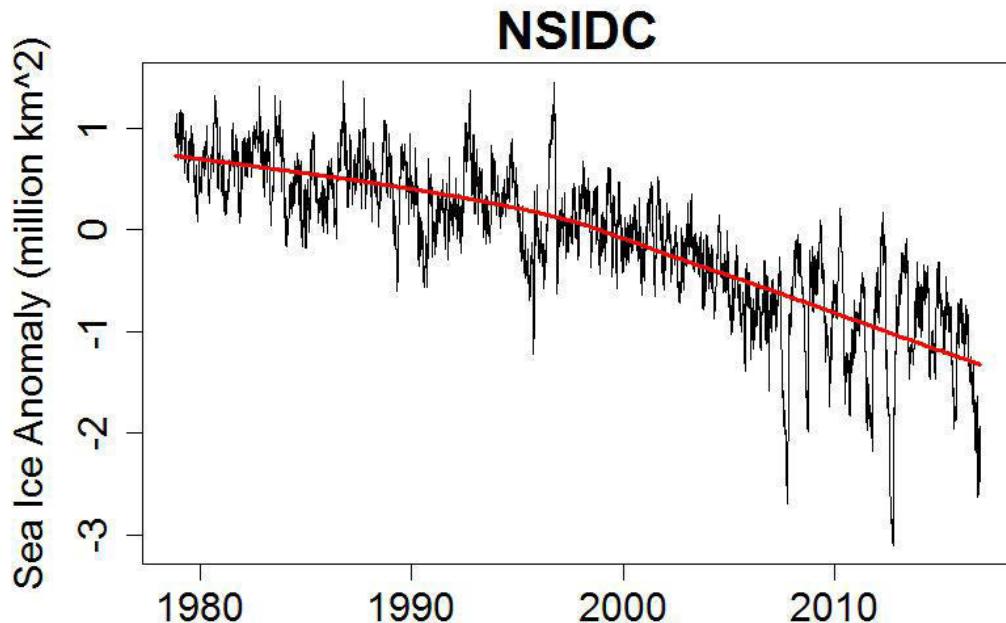


Moral: Don't
confuse
fluctuations with
trends.

Chose an interval
consistent with
phenomenon
being considered.

But At Least the Arctic Ice Isn't Melting

“Yesterday, April 14th, the Arctic had more sea ice than it had on April 14, 1989 – 14.511 million square kilometres vs 14.510 million square kilometres, according to the National Snow and Ice Data Center of the United States, an official source.” *Lawrence Solomon, Financial Post, April 15, 2013*



Cherry picking image © source unknown. All rights reserved.
This content is excluded from our Creative Commons license.
See <https://ocw.mit.edu/help/faq-fair-use/>.

A Comforting Statistic

- 99.8% of the firearms in the U.S. will not be used to commit a violent crime in any given year
- How many privately owned firearms in U.S.?
- ~300,000,000
- $300,000,000 * 0.002 = 600,000$

A Not So Comforting Statistic

“Mexican health officials suspect that the swine flu outbreak has caused more than 159 deaths and roughly 2,500 illnesses.” CNN, April 29, 2009

How many deaths per year from seasonal flu in U.S.?

About 36,000

Relative to What?

- Skipping lectures increases your probability of failing 6.0002 by 50%
- From 0.5 to 0.75
- From 0.005 to 0.0075
- Moral: Beware of percentage change when you don't know the denominator

Cancer Clusters

- A **cancer cluster** is defined by the CDC as “a greater-than-expected number of cancer cases that occurs within a group of people in a geographic area over a period of time”
- About 1000 “cancer clusters” per year are reported to health authorities in the U.S.
- Vast majority are deemed not significant

A Hypothetical Example

- Massachusetts is about 10,000 square miles
- About 36,000 new cancer cases per year
- Attorney partitioned state into 1000 regions of 10 squares miles each, and looked at distribution of cases
 - Expected number of cases per region: 36
- Discovered that region 111 had 143 new cancer cases over a 3 year period!
 - More than 32% greater than expected
- How worried should residents be?

How Likely Is it Just Bad Luck?

```
numCasesPerYear = 36000
numYears = 3
stateSize = 10000
communitySize = 10
numCommunities = stateSize//communitySize

numTrials = 100
numGreater = 0
for t in range(numTrials):
    locs = [0]*numCommunities
    for i in range(numYears*numCasesPerYear):
        locs[random.choice(range(numCommunities))] += 1
    if locs[111] >= 143:
        numGreater += 1
prob = round(numGreater/numTrials, 4)
print('Est. probability of region 111 having\
at least 143 cases =', prob)
```

How Likely Is it Just Bad Luck?

```
anyRegion = 0
for trial in range(numTrials):
    locs = [0]*numCommunities
    for i in range(numYears*numCasesPerYear):
        locs[random.choice(range(numCommunities))] += 1
    if max(locs) >= 143:
        anyRegion += 1
print(anyRegion)
aProb = round(anyRegion/numTrials, 4)
print('Est. probability of some region having\
at least 143 cases = ', aProb)
```

A variant of cherry picking called
multiple hypothesis testing

The Bottom Line

- When drawing inferences from data, skepticism is merited.
- But remember, skepticism and denial are different.
- “Doubt, indulged and cherished, is in danger of becoming denial; but if honest, and bent on thorough investigation, it may soon lead to full establishment of the truth.” – Ambrose Bierce

6.0002 Major Topics

- Optimization problems
- Stochastic thinking
- Modeling aspects of the world
- Becoming a better programmer
 - Exposure to a few extra features of Python and some useful libraries
 - Practice, practice, practice

Optimization Problems

- Many problems can be formulated in terms of
 - Objective function
 - Set of constraints
- Greedy algorithms often useful
 - But may not find optimal solution
- Many optimization problems inherently exponential
 - But dynamic programming often works
 - And memoization a generally useful technique
- Examples: knapsack problems, graph problems, curve fitting, clustering

Stochastic Thinking

- The world is (predictably) non-deterministic
- Thinking in terms of probabilities is often useful
- Randomness is a powerful tool for building computations that model the world
- Random computations useful even when for problems that do not involve randomness
 - E.g., integration

Modeling the World

- Models always inaccurate
 - Provide abstractions of reality
- Deterministic models, e.g., graph theoretic
- Statistical models
 - Simulation models: Monte Carlo simulation
 - Models based on sampling
 - Characterizing accuracy is critical
 - Central limit theorem
 - Empirical rule
 - Machine learning
 - Unsupervised and supervised
- Presentation of data
 - Plotting
 - Good and bad practices

What's Next for You?

- Many of you have worked very hard
 - Rest of the staff and I appreciate it
- Only you know your return on investment
 - Take a look at early problem sets
 - Think about what you'd be willing tackle now
- Remember that you can write programs to get answers
- There are other CS courses you are prepared to take
 - 6.009, 6.005, 6.006, 6.034
- Find an interesting UROP
- Minor in CS
- Major in CS

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.