Lecture 10: Validation, Intro to ML and Clustering

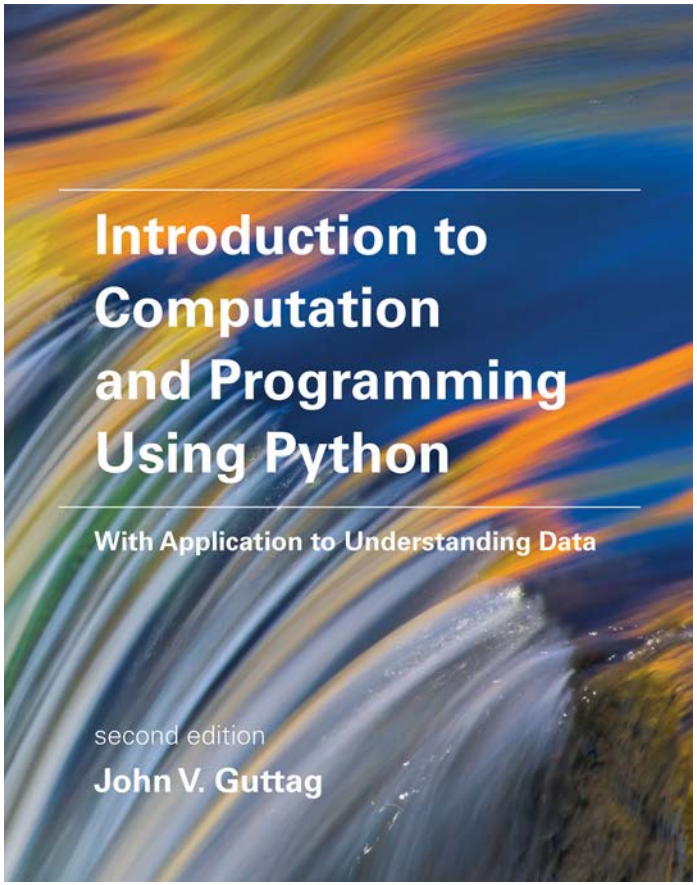
(download slides and .py files from Stellar to follow along)

John Guttag

MIT Department of Electrical Engineering and Computer Science

Relevant Reading

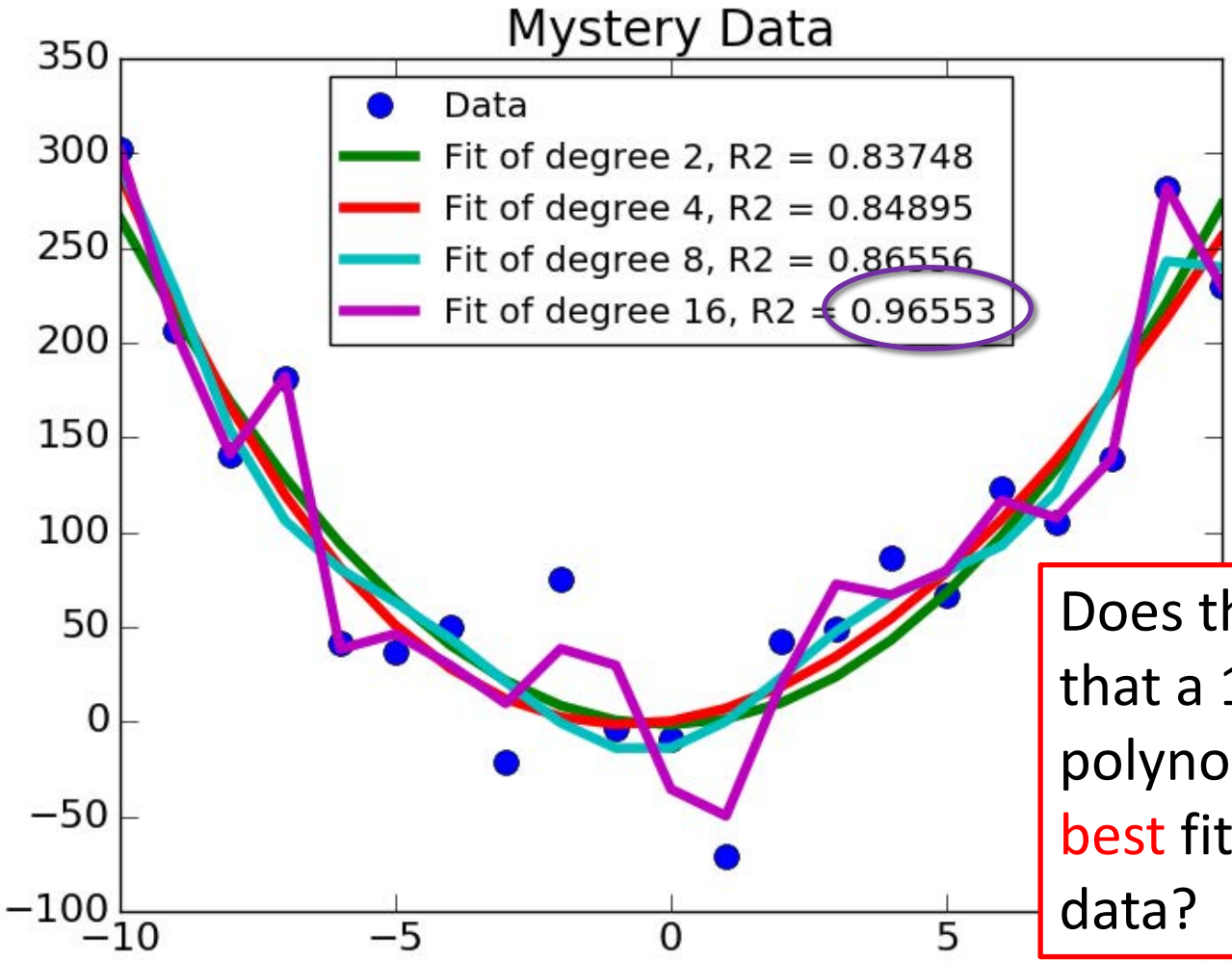
- Today
  - Chapters 23-23
- Wednesday
  - Chapter 24



Two Topics Today

- Finish up curve fitting
  - curveFitting.py
- Start machine learning
  - Lect10.py

Four Curves Fit to the Same Data



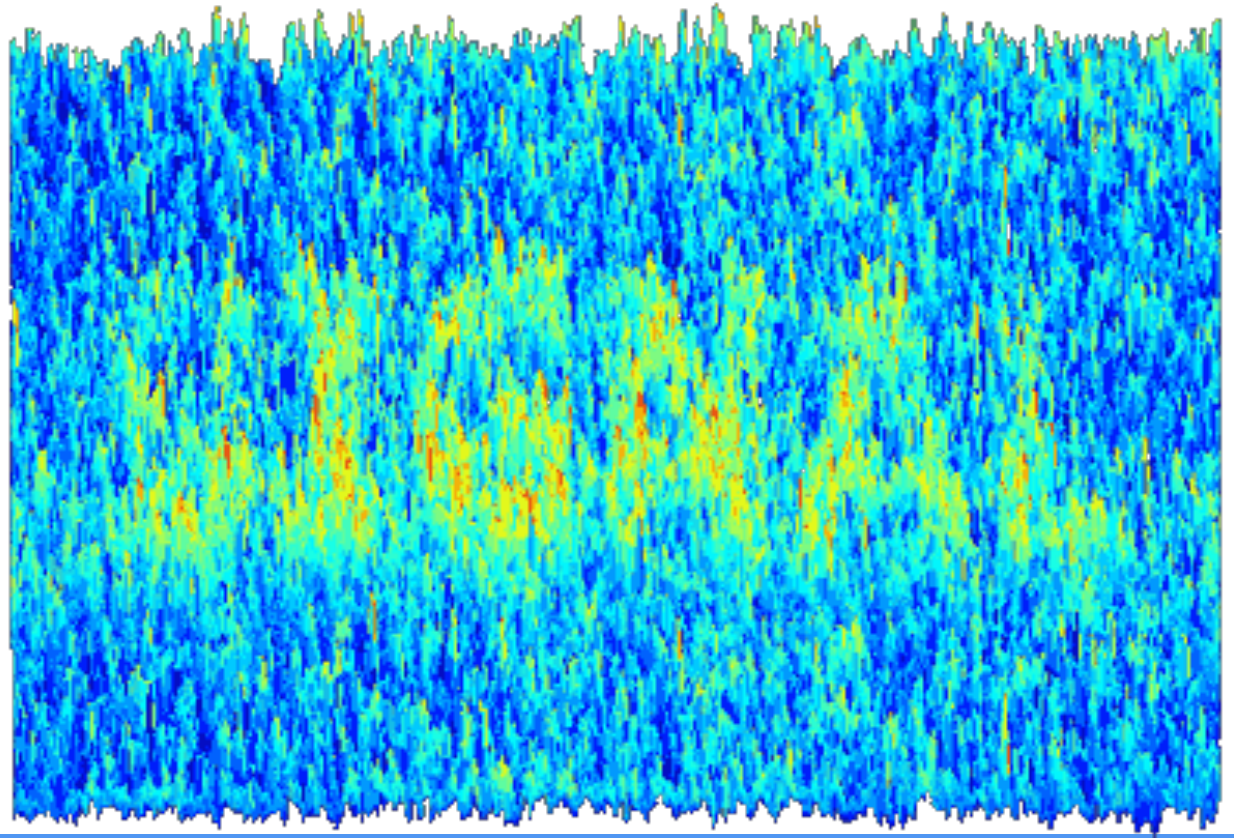
Does this mean that a 16th order polynomial is the best fit for the data?

Does Tightest = Best?

- 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?**
- 1) Help us understand process that generated the data
  - E.g., the properties of a particular linear spring
- 2) 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
- A good model helps us do both of these things

The Key Question

- To what extent is the model shaped by the underlying process we are trying to understand?
- When the model is complex, it runs the risk of fitting the noise



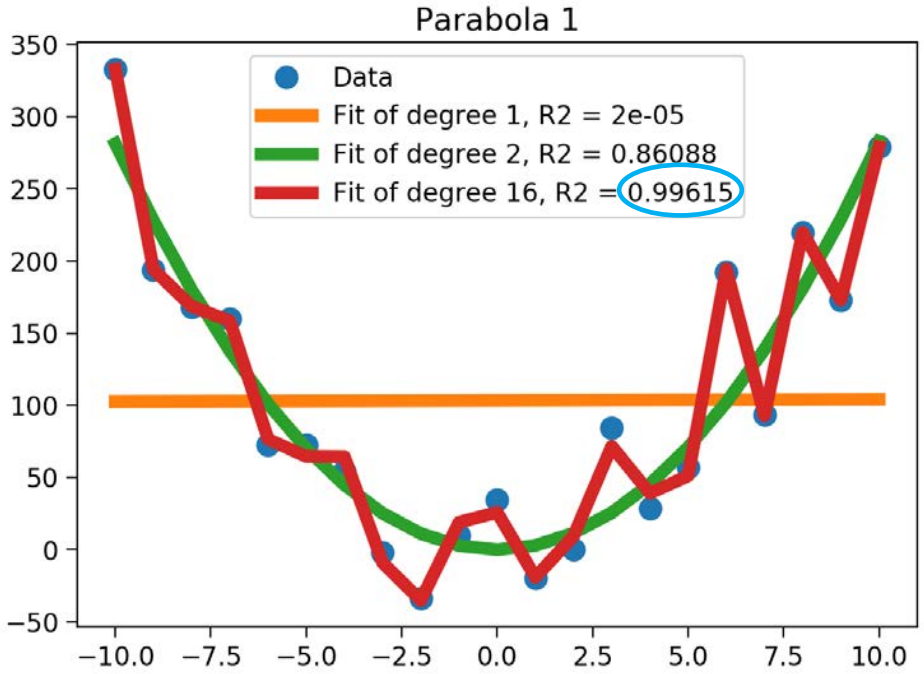
Training versus Testing

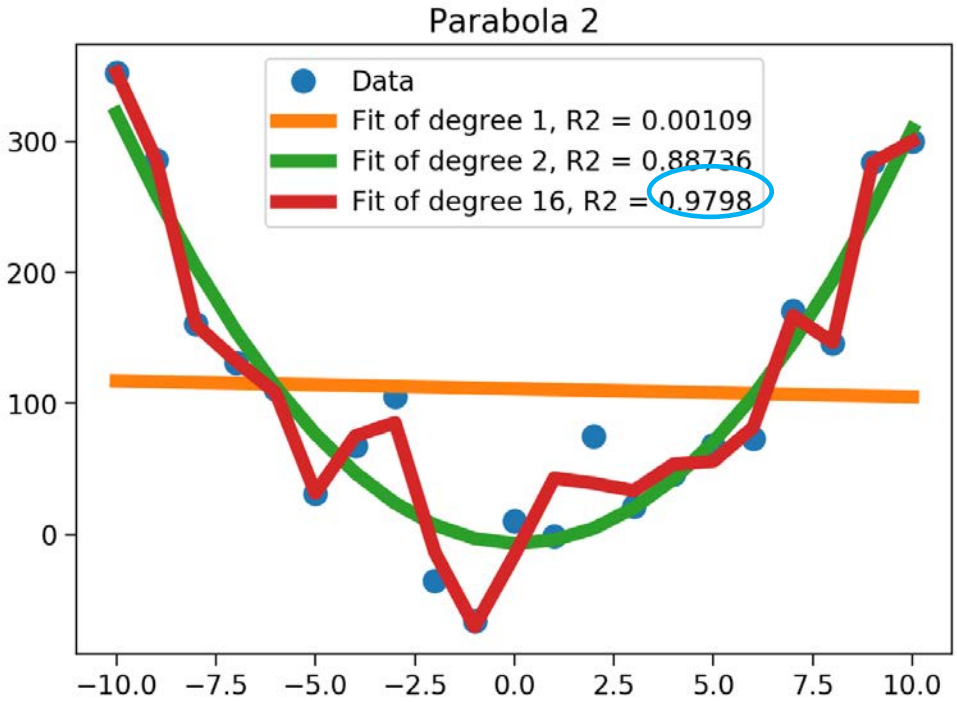
- One way to separate out impact of noise on model is to take advantage of fact that each time we sample a system
  - Signal will be roughly the same
  - Noise will be different
- Use set of data as a "training" set to fit a model
- Use a second set of data as a "test" set, and see how well the model from the training set accounts for the test set

Generate 2 Data Sets from Same Distribution

```
xVals = range(-10, 11, 1)
a, b, c = 3, 0, 0
genNoisyParabolicData(a, b, c, xVals, 'parabola1.txt')
genNoisyParabolicData(a, b, c, xVals, 'parabola2.txt')

degrees = (1, 2, 16)
xVals1, yVals1 = getData('parabola1.txt')
models1 = genFits(xVals1, yVals1, degrees)
testFits(models1, degrees, xVals1, yVals1, 'Parabola 1')

plt.figure()
xVals2, yVals2 = getData('parabola2.txt')
models2 = genFits(xVals2, yVals2, degrees)
testFits(models2, degrees, xVals2, yVals2, 'Parabola 2')
```

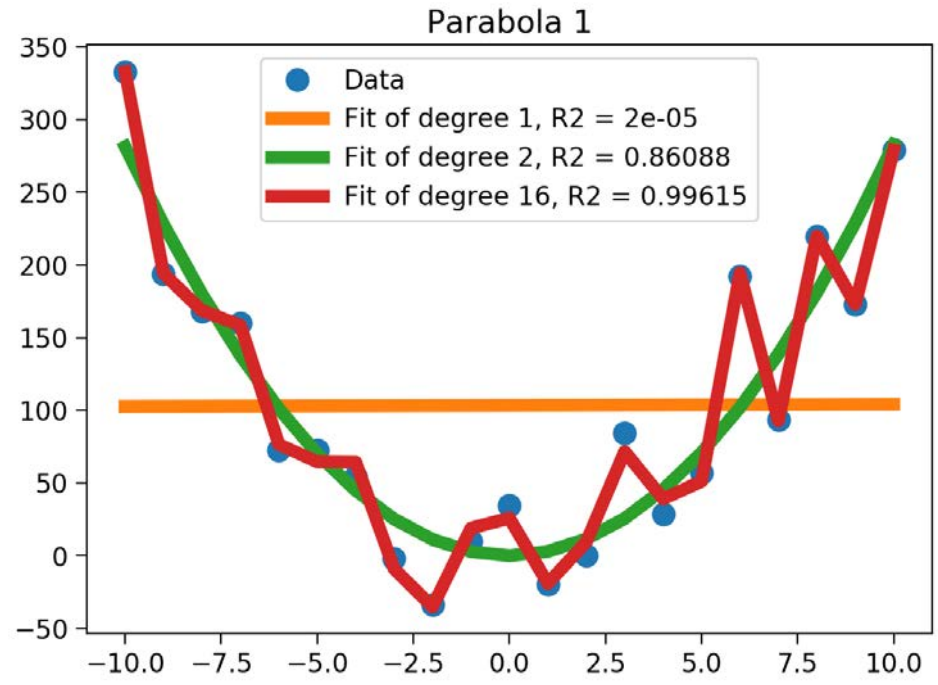
Look at Fits to Training Data

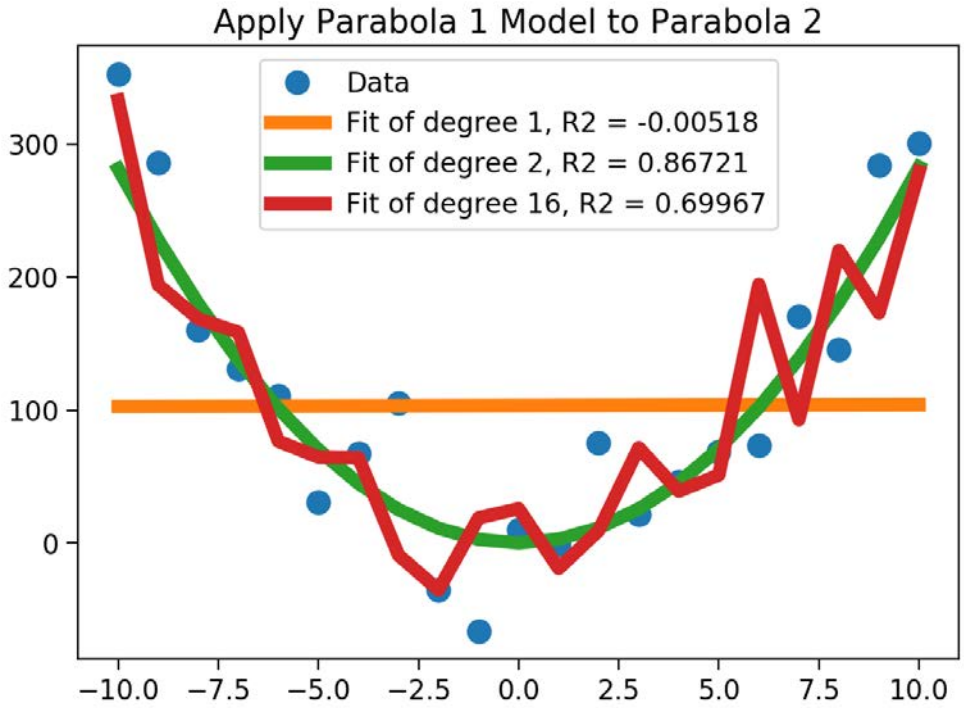




Training and Testing Errors

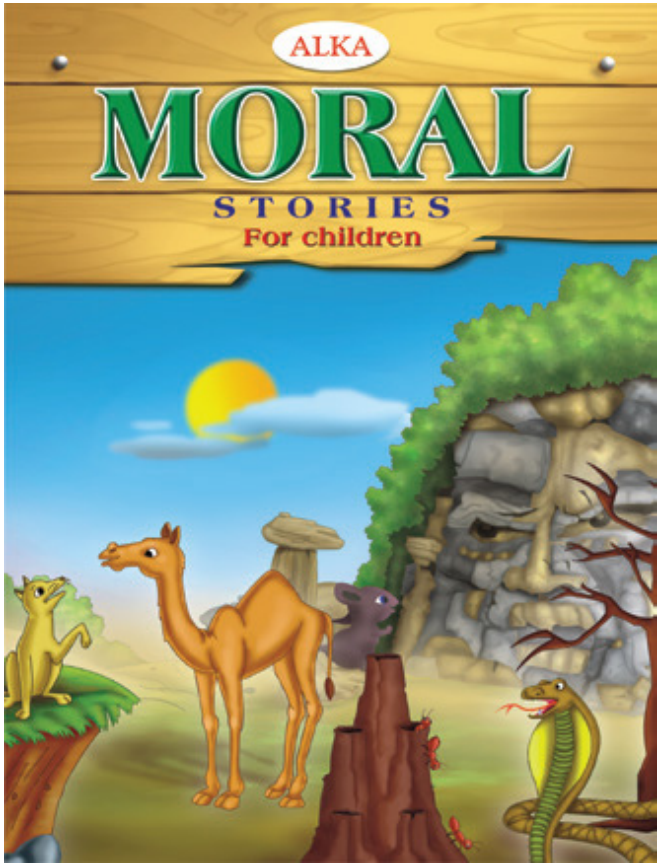
```
testFits(models1, degrees, xVals1, yVals1, 'Parabola 1')

testFits(models1, degrees, xVals2, yVals2,
         'Apply Parabola 1 Model to Parabola 2')
```





The Moral of the Story

- 16-degree polynomial is an example of overfitting to the data
- If we only look at how well model fits training data, we may not detect that model is too complex
- Need to validate: Train on one data set, then test on a different set



Alas, We Can't Simply Generate Data

```
def splitData(xVals, yVals, fracTraining):
    trainingSize = int(len(xVals)*fracTraining)
    trainingI = random.sample(range(len(xVals)), trainingSize)
    trainingI.sort()
    trainingX, trainingY, testX, testY = [], [], [], []
    for i in range(len(xVals)):
        if i in trainingI:
            trainingX.append(xVals[i])
            trainingY.append(yVals[i])
        else:
            testX.append(xVals[i])
            testY.append(yVals[i])
    return (trainingX, trainingY), (testX, testY)
```

Alas, We Can't Simply Generate Data

- Use **cross validation**
  - Split data into training and validation set

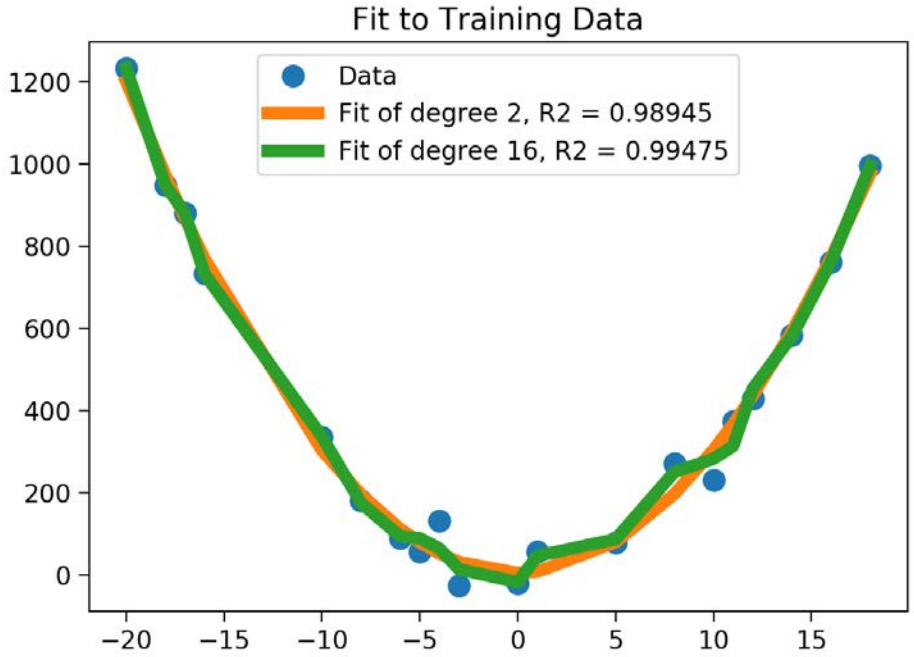
```
def splitData(xVals, yVals, fracTraining):
    trainingSize = int(len(xVals)*fracTraining)
    trainingI = random.sample(range(len(xVals)), trainingSize)
    trainingI.sort()
    trainingX, trainingY, testX, testY = [], [], [], []
    for i in range(len(xVals)):
        if i in trainingI:
            trainingX.append(xVals[i])
            trainingY.append(yVals[i])
        else:
            testX.append(xVals[i])
            testY.append(yVals[i])
    return (trainingX, trainingY), (testX, testY)
```

Validating a Model

```
def fitAndValidate(xVals, yVals, degrees):
    training, test = splitData(xVals, yVals, .5)
    models = []
    for d in degrees:
        models.append(np.polyfit(training[0], training[1], d))
    for m in models:
        print([round(c, 2) for c in m])
    testFits(models, degrees, training[0], training[1],
             'Fit to Training Data')
    plt.figure()
    testFits(models, degrees, test[0], test[1],
             'Applied to Test Data')

xVals = range(-20, 20, 1)
a, b, c = 3, 0, 0
genNoisyParabolicData(a, b, c, xVals, 'parabola.txt')
xVals, yVals = getData('parabola.txt')

fitAndValidate(xVals, yVals, (2, 16))
```

Training and Test Splits

Validating a Model



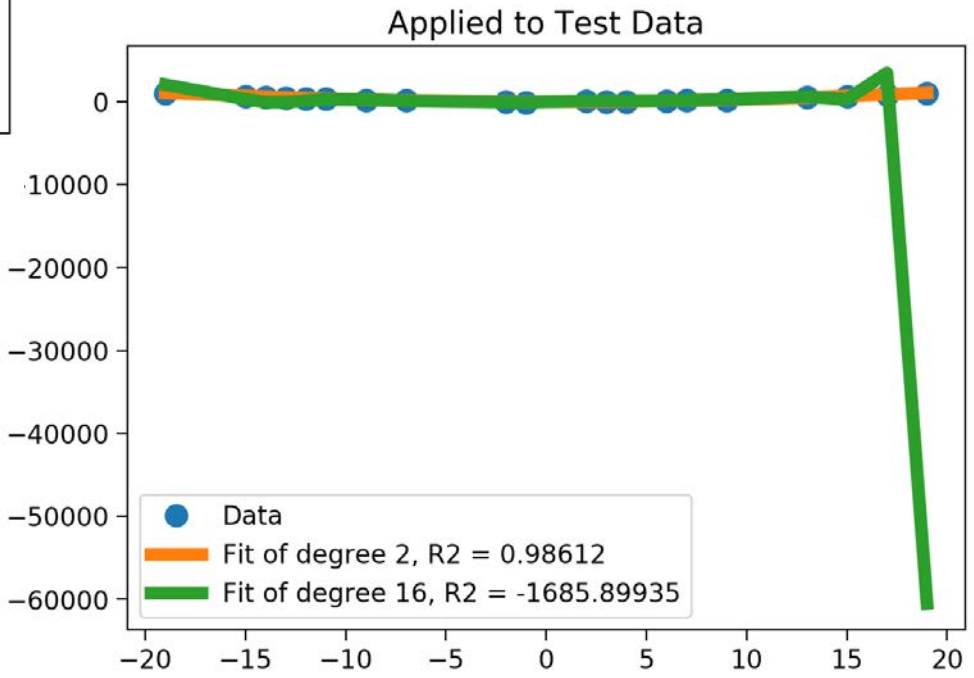
degree = 2 model (rounded)

$y = 2.99x^2 + 0.07x + 4.11$

Degree = 16 model (rounded)

$y = -0.03x^6 + 0.23x^5 - 0.95x^4 - 5.82x^3 + 15.94x^2 + 54.55x - 17.95$

Poll



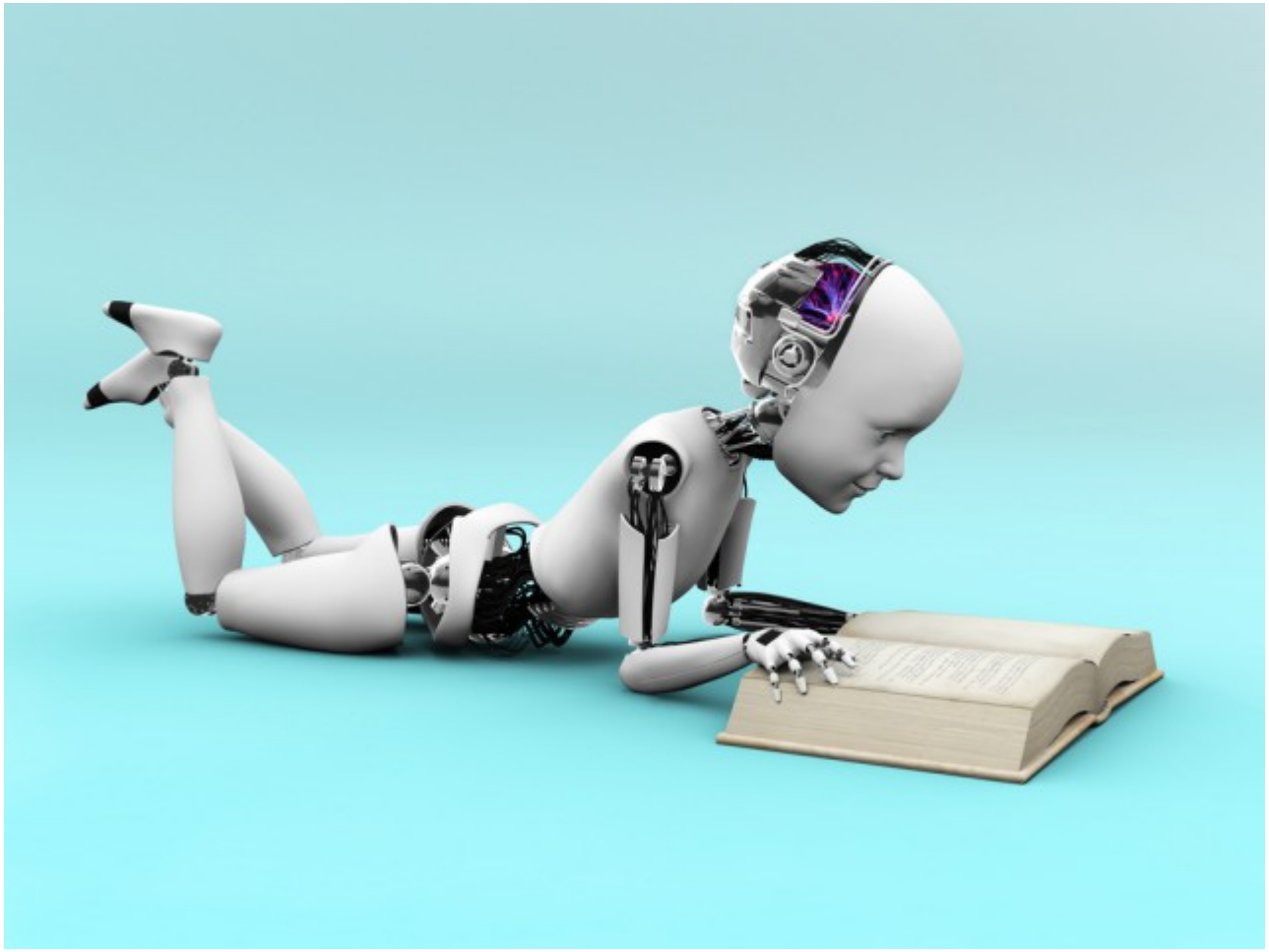
We Could Have Gotten Lucky

- Training and test data just happened to have similar noise
- **n-fold cross validation**
  - Split data into training and test sets n times
  - Look at behavior across all splits

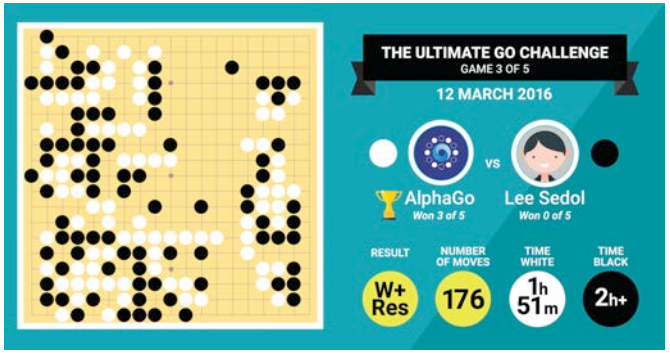
The Take Home Message

- 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
- **Applies to all data-driven models**
  - Not just linear regression

Onto Machine Learning

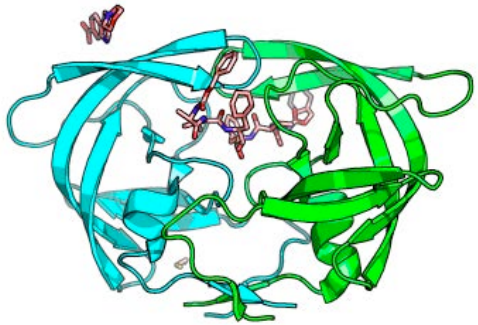


Machine Learning Is Everywhere

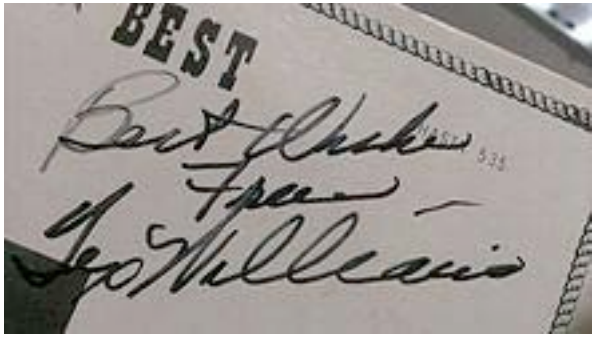


Games

Recommendation systems

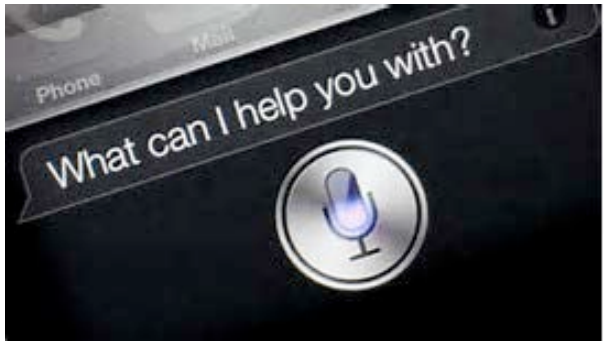


Drug discovery

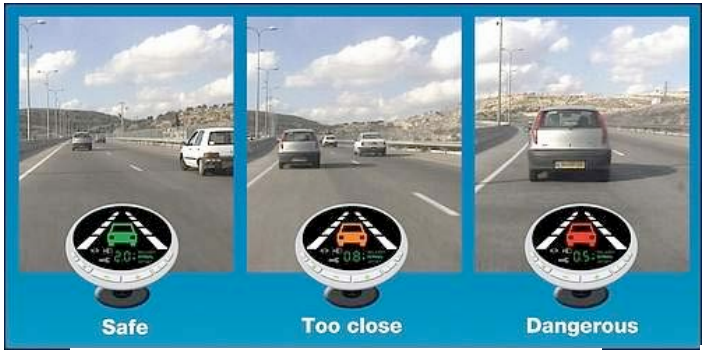


Character recognition

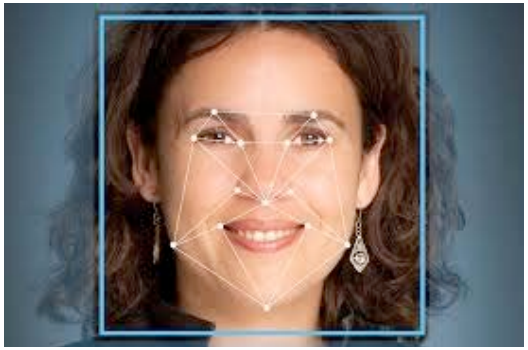
Hedge fund stock predictions



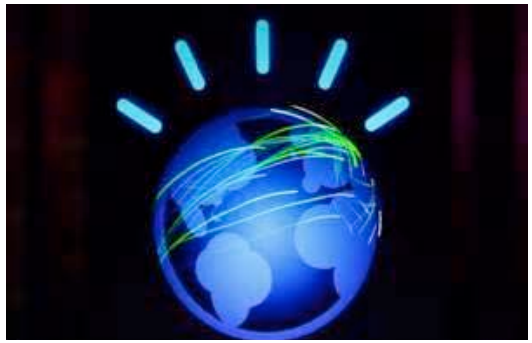
Voice assistants



Assisted driving



Face detection/recognition



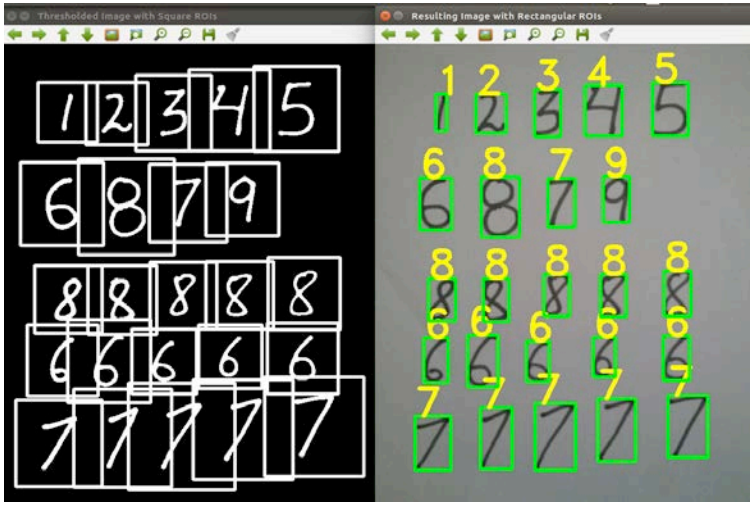
Cancer diagnosis

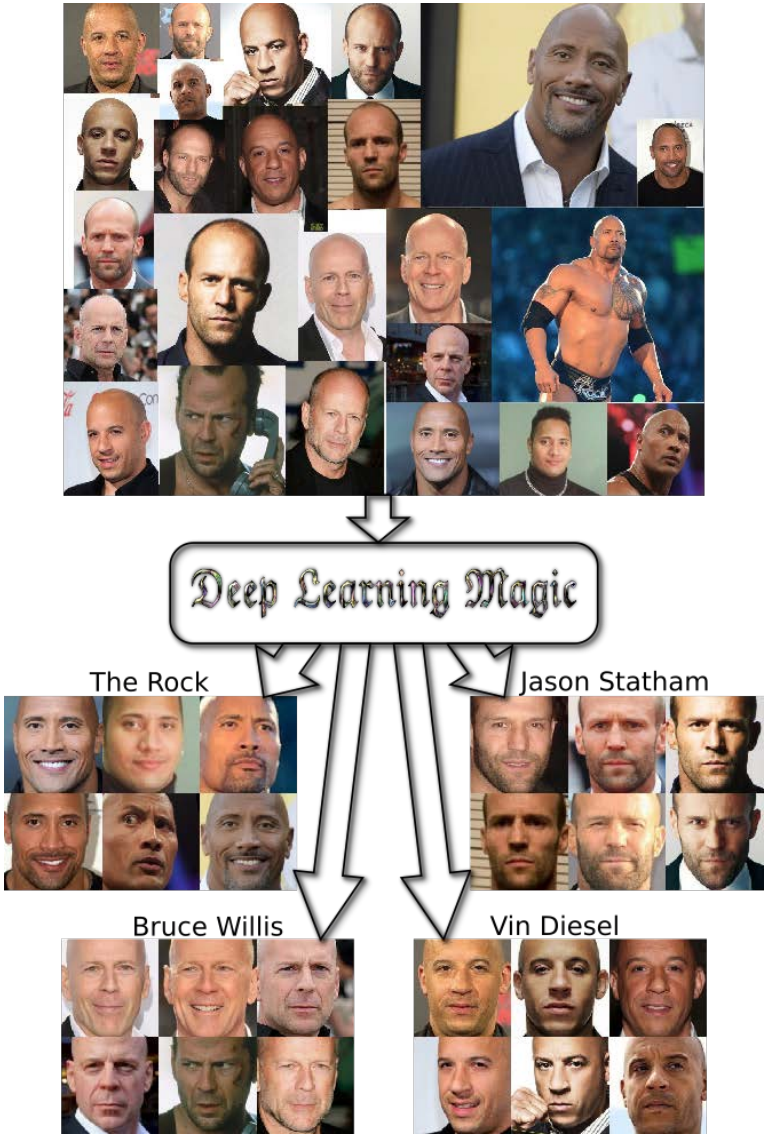
Success stories: Speech & Language

- Many applications already available
  - Apple Siri
  - Amazon Echo
  - Baidu Deep Voice
  - Google Translate
  - Zoom captioning

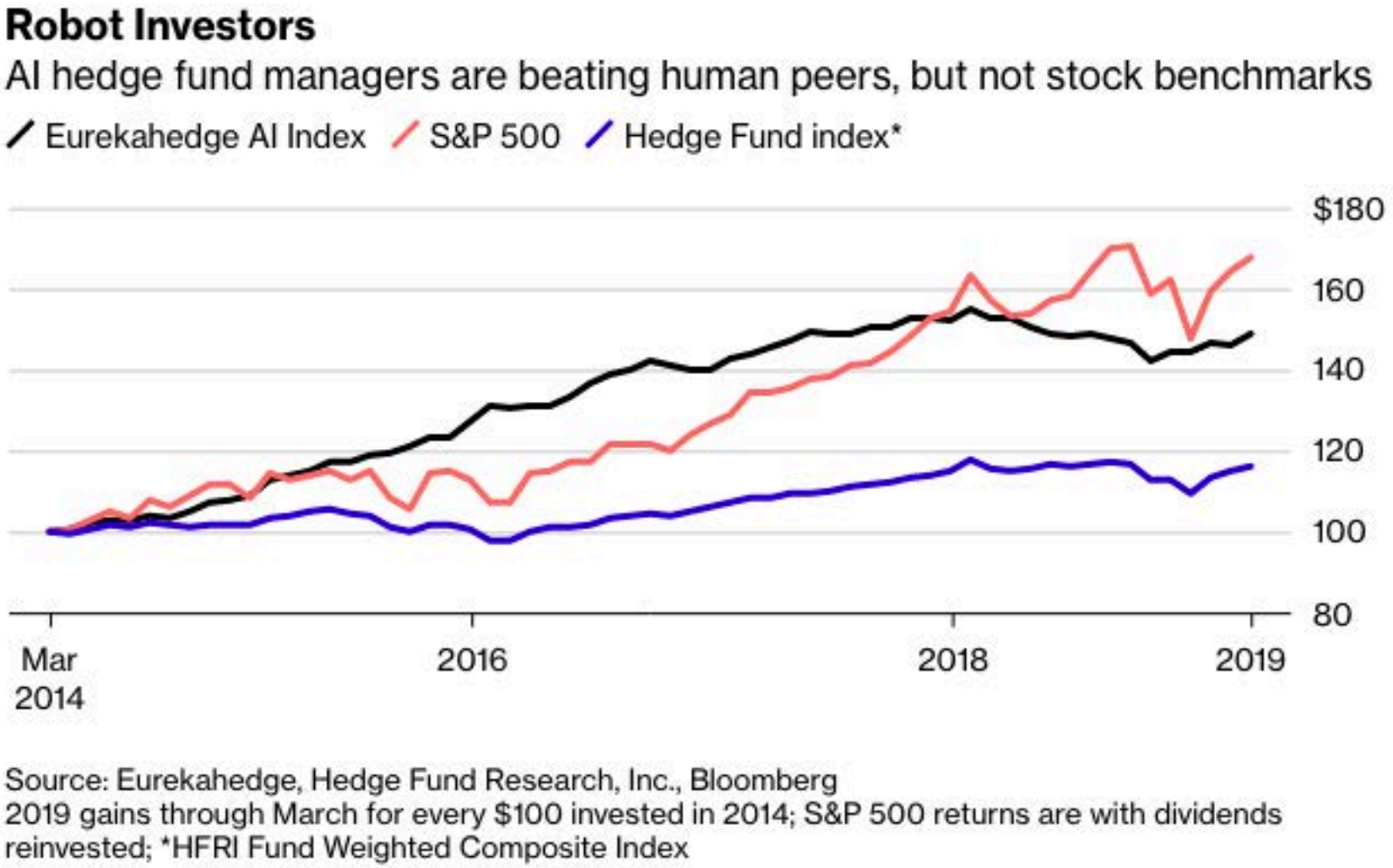
Success stories: Vision

- Face recognition
- Postal service uses handwriting recognition





Finance


Robot Investors
AI hedge fund managers are beating human peers, but not stock benchmarks
Eurekahedge AI Index
S&P 500
Hedge Fund index*
$180
160
140
120
100
80
Mar 2014
2016
2018
2019
Source: Eurekahedge, Hedge Fund Research, Inc., Bloomberg
2019 gains through March for every $100 invested in 2014; S&P 500 returns are with dividends reinvested; *HFRI Fund Weighted Composite Index

Success stories: Game players

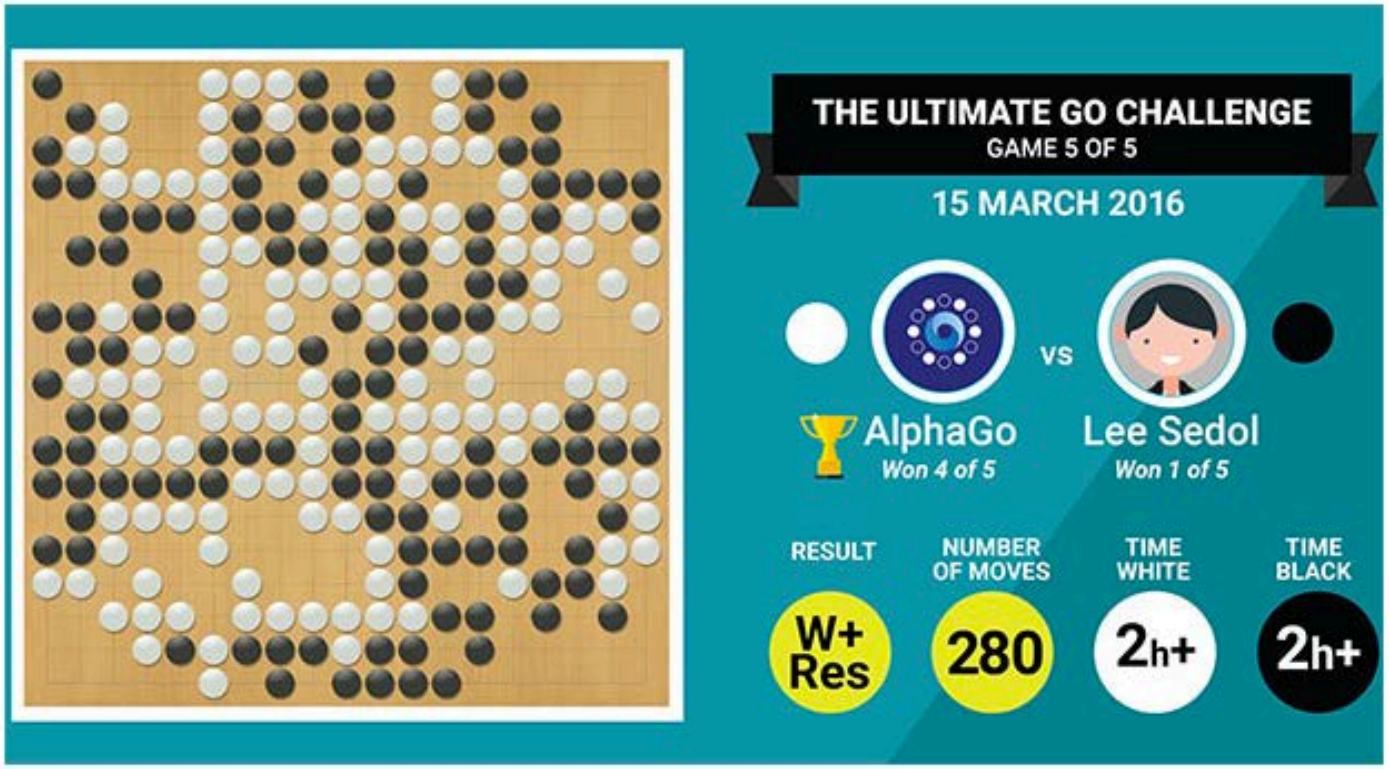


Image: Google

AlphaGo
Early 2016

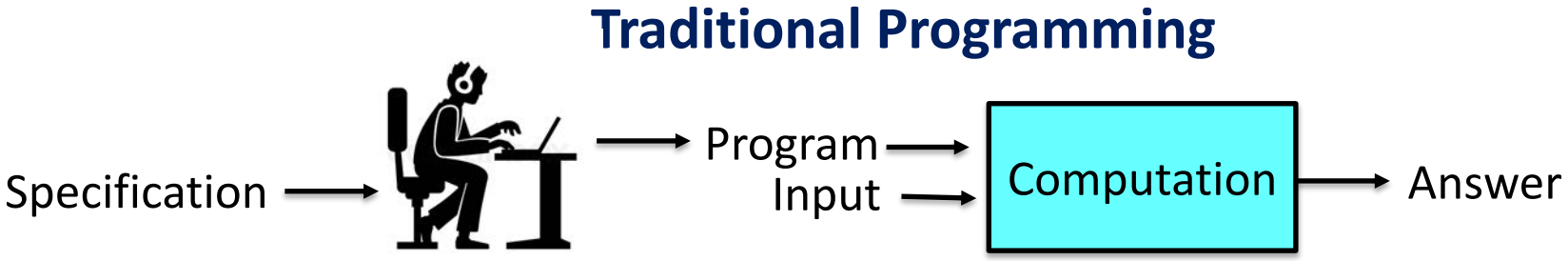
What Is Machine Learning?

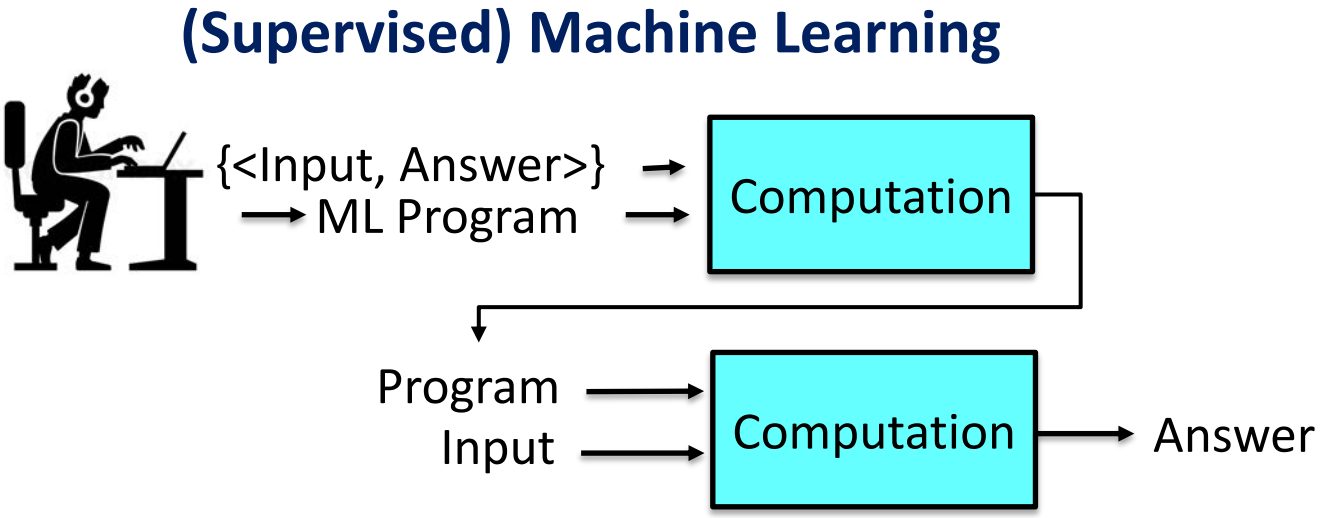
- All useful programs "learn" something
- In the first lecture of 6.0001 we looked at an algorithm for finding (learning?) square roots
- We recently looked at using linear regression to find (learn) a model of a collection of points
- We could argue that root finding and curve fitting algorithms "learn" models to fit to data sets
- But each algorithm is designed to meet a specific goal, and somehow machine learning should be broader than that

What Is Machine Learning?

- 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
- *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."* Tom Mitchell – CMU (1997)

What Is Machine Learning?





What Is Machine Learning?

https://xkcd.com/1838/

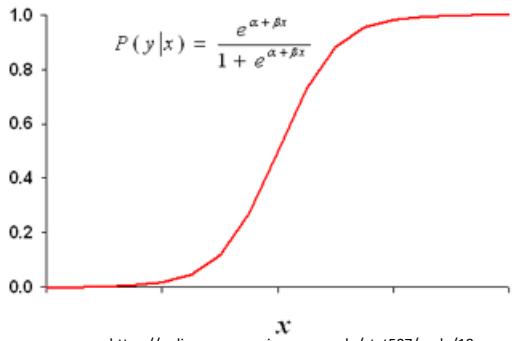
How Are Things Learned?

- Memorization
  - Accumulation of individual facts
  - Limited by
    - Time to observe facts
    - Memory to store facts
- Generalization
  - Deduce new facts from old facts
  - Limited by accuracy of deduction process
    - Essentially a predictive activity
    - Assumes that the past predicts the future
- Extend deduction to programs that can infer useful information from **implicit** patterns in data

Basic Paradigm

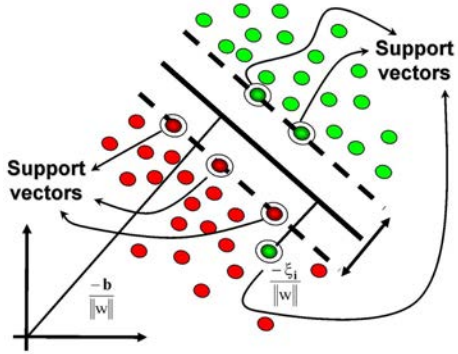
- Observe set of examples: training data
- Infer something about process that generated that data – learn a model that predicts data
  - Regression: prediction is continuous
    - E.g., predict what a student's GPA will
  - Classification: prediction is categorical
    - E.g., predict whether a student will major in CS
- Use inference to make predictions about previously unseen data: test data

All ML Methods Solve Optimization Problems



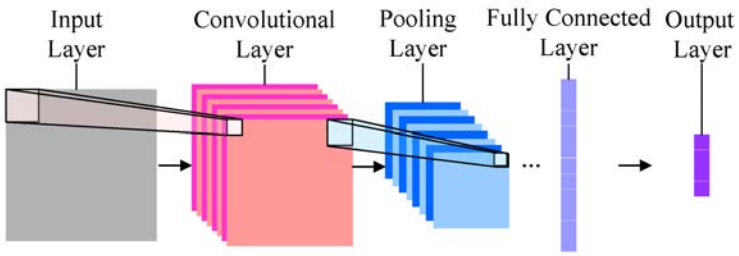
https://onlinecourses.science.psu.edu/stat507/node/18

Logistic Regression
1958 (Cox)



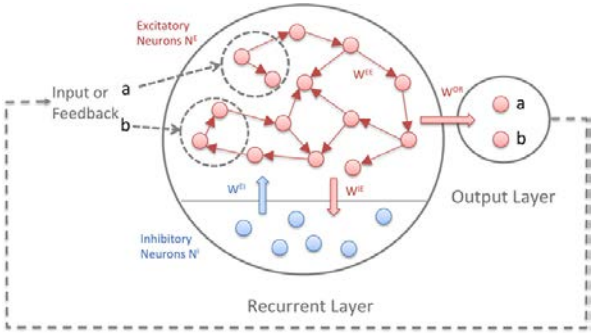
Source: https://medium.com/@haydar_ai/learning-data-science-day-11-support-vector-machine-8ef06da91bfc

Support Vector Machines
1963/1992 (Vapnik et al.)



Source: http://www.mdpi.com

Convolutional Neural Networks



Source: www.frontiersin.org/articles/10.3389/fncom.2015.00036/full

Recurrent Neural Networks

Neural Networks
1957/1986/1998/2006/2012

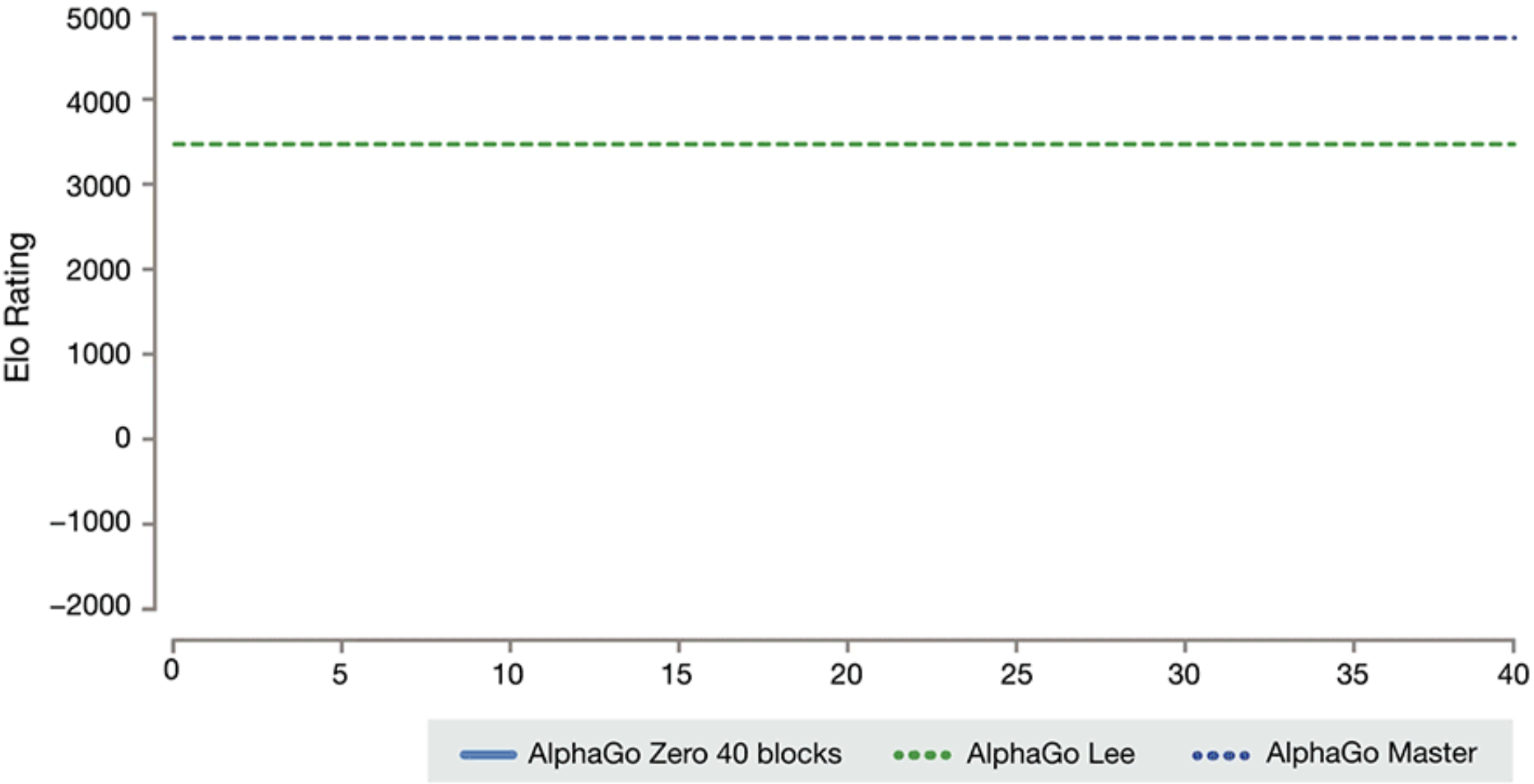
Unsupervised and Supervised Learning

- Unsupervised learning
  - Given set of unlabeled examples
  - Convert each into a vector of features
  - Cluster based on similarity of feature vectors (e.g., k means)
- Supervised learning
  - Given set of labeled examples
  - Convert each into a vector of features
  - Use algorithm to learn coefficient of variables to optimize tradeoff of selectivity and specificity

Returning to DeepMind's AlphaGo

- AlphaGo used a Monte Carlo tree search algorithm to find moves, based on knowledge learned using an artificial neural network (ANN) trained against humans and itself
  - Uses reinforcement learning on an ANN to refine model
- AlphaGo Zero had no human input
  - Had rules for generating legal moves
  - Learned by playing itself

The Training of AlphaGo Zero

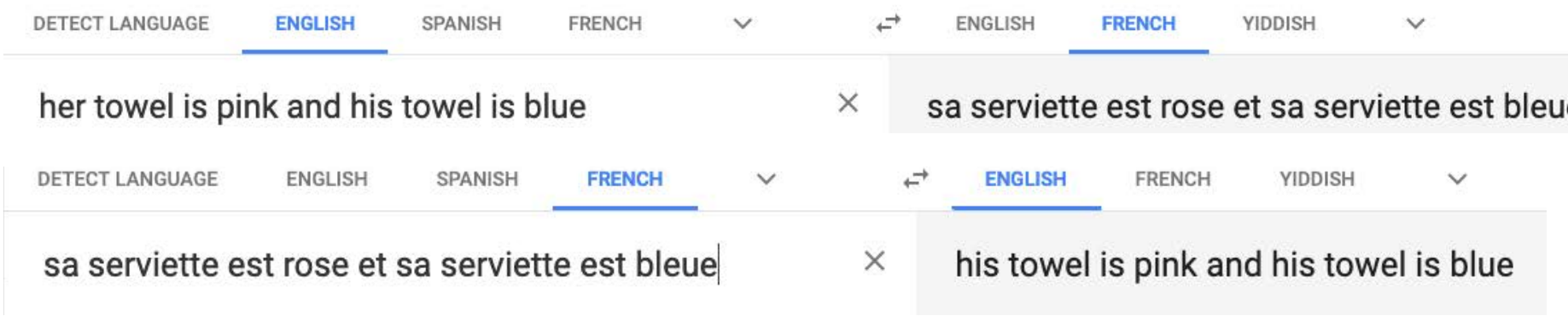


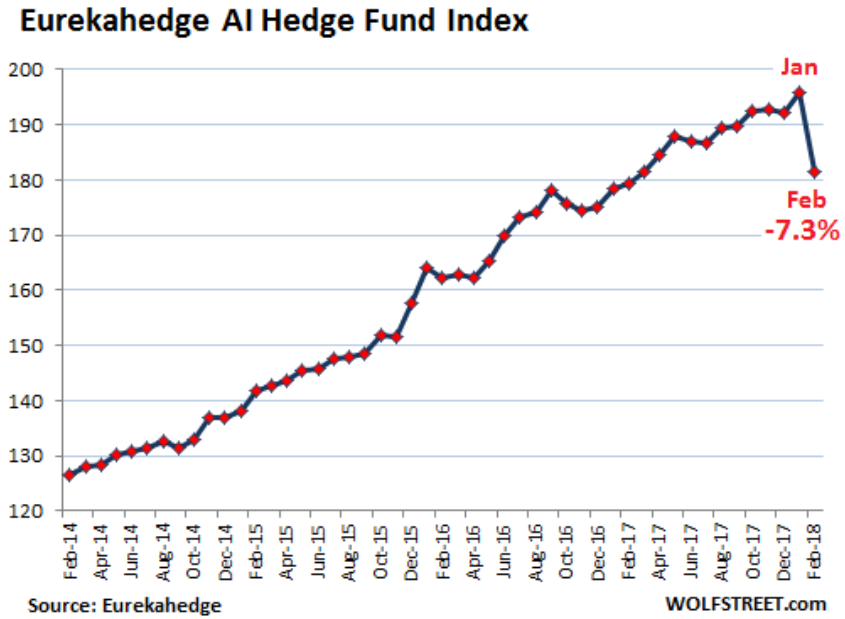
All ML Methods Require:

- 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

At the End of the Day, ML Produces a Model

- All models are wrong

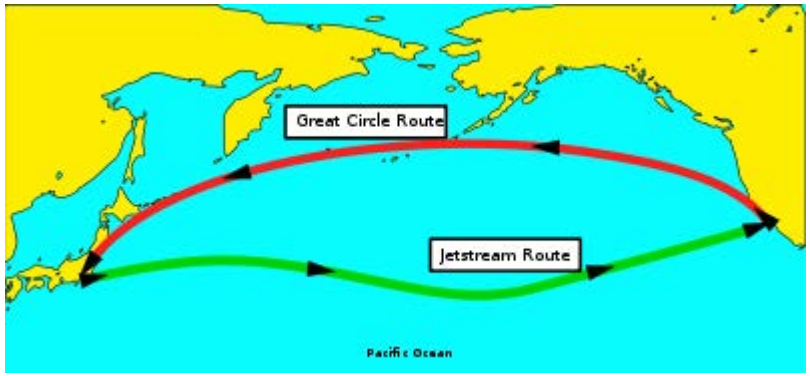




Poll

Setting Up the Learning Framework

- How are we going to represent our training data?
  - What features are important?
  - How are they represented? (Typically we want features that can be mapped to numerical values, so we can measure distances between examples)
    - Binary
    - Integers
    - Floats
- How do we measure distances between feature vectors representing instances?



Feature Representation

- Features never fully describe the situation
- 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), mathematical sophistication
    - Others might cause me to overfit, e.g., birth month, eye color
- Want to maximize ratio of useful input to irrelevant input in choice of features
  - Signal-to-Noise Ratio (SNR)

How Do We Learn to Assign Labels to Examples?

- Have sets of examples represented as points in a feature space
- Intuition – examples with same label are close to one another in feature space
  - Do similar examples form one cluster in feature space, or several?
  - Which features are most important in grouping examples?
  - What does "close" mean?
- Goal is to find way to group similar objects
  - Use **distance** between examples to determine important features and to identify new instances by type

Feature Engineering

- Deciding which features to include and which are merely adding noise to classifier

You've seen this! – variant of overfitting

- 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

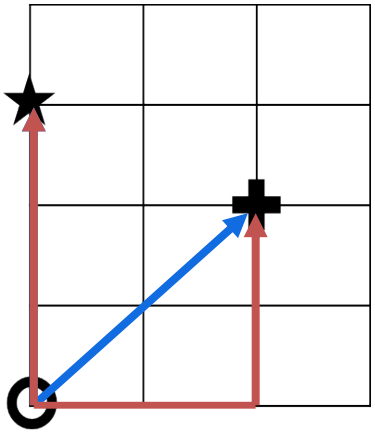
Measuring Distance: Minkowski Metric

- **Poll**

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

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

Poll

Other Distance Metrics

- Minkowski distance is commonly used because it naturally supports gradient descent optimization methods
- But there are other distance metrics that are sometimes more appropriate
- One common metric:
  - Earth mover's distance

Earth Mover's Distance

- Given two distributions (or histograms), what is the minimum amount of matter (dirt) that has to be moved (cost is amount to move times distance moved) to make the distributions match

Break

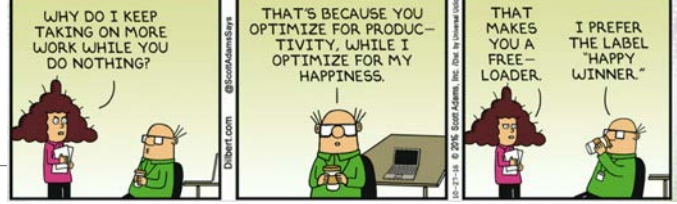
Clustering

- Cluster: a collection of examples that share a set of similar properties
  - Ideally properties are numeric or can be converted to a numeric scale, so that clusters defined by sets of "nearby" examples
- Goal is to use clusters to predict outcomes or labels for new examples
  - Categorize new examples by assigning to "closest" cluster
  - Categorize new examples by deriving an algorithm deduced from shared properties of examples within cluster

Key Challenges with Clustering

- Assuming that examples with shared outcome/label all lie "near" one another in space of feature measurements
  - What if clusters overlap in feature space? Or in some dimensions of feature space?
  - What if some features are irrelevant, but confuse clustering? How can we identify the key features?
  - What if the scales of different dimensions of feature space (different measurements) are very different?
- We need:
  - Way of deciding how "close" examples are
  - Way of using "closeness" to group examples into clusters

Clustering Is Optimization



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

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

c is a cluster

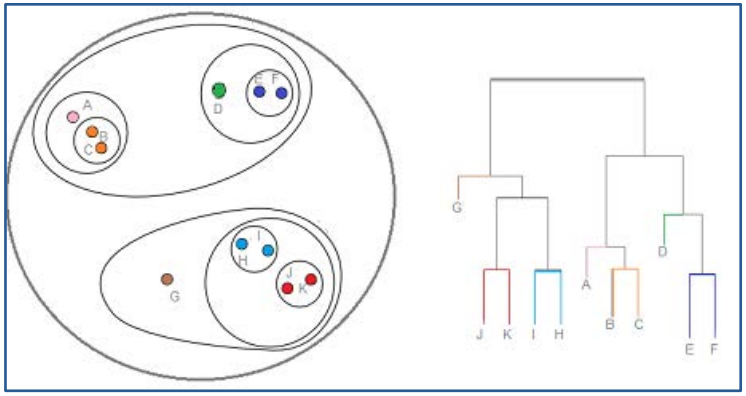
C is a collection of clusters

Want to minimize dissimilarity

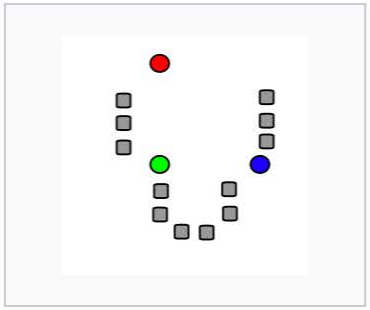
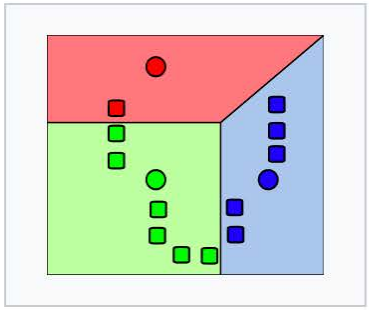
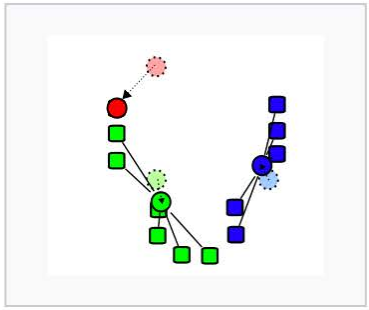
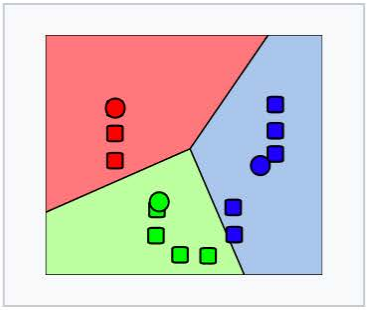
- Why not divide variability by size of cluster?
  - Big and bad is worse than small and bad
- Is optimization problem simply finding a C that minimizes *dissimilarity(C)*?
  - No, otherwise could put each example in its own cluster
- Need a constraint, for example:
  - Minimum distance between clusters
  - Number of clusters (maximum overall, or specific number)

Two Popular Methods for Clustering

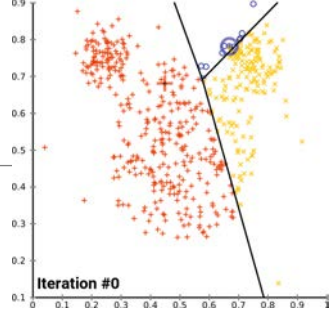
- Hierarchical clustering



- K-means clustering

K-means Algorithm

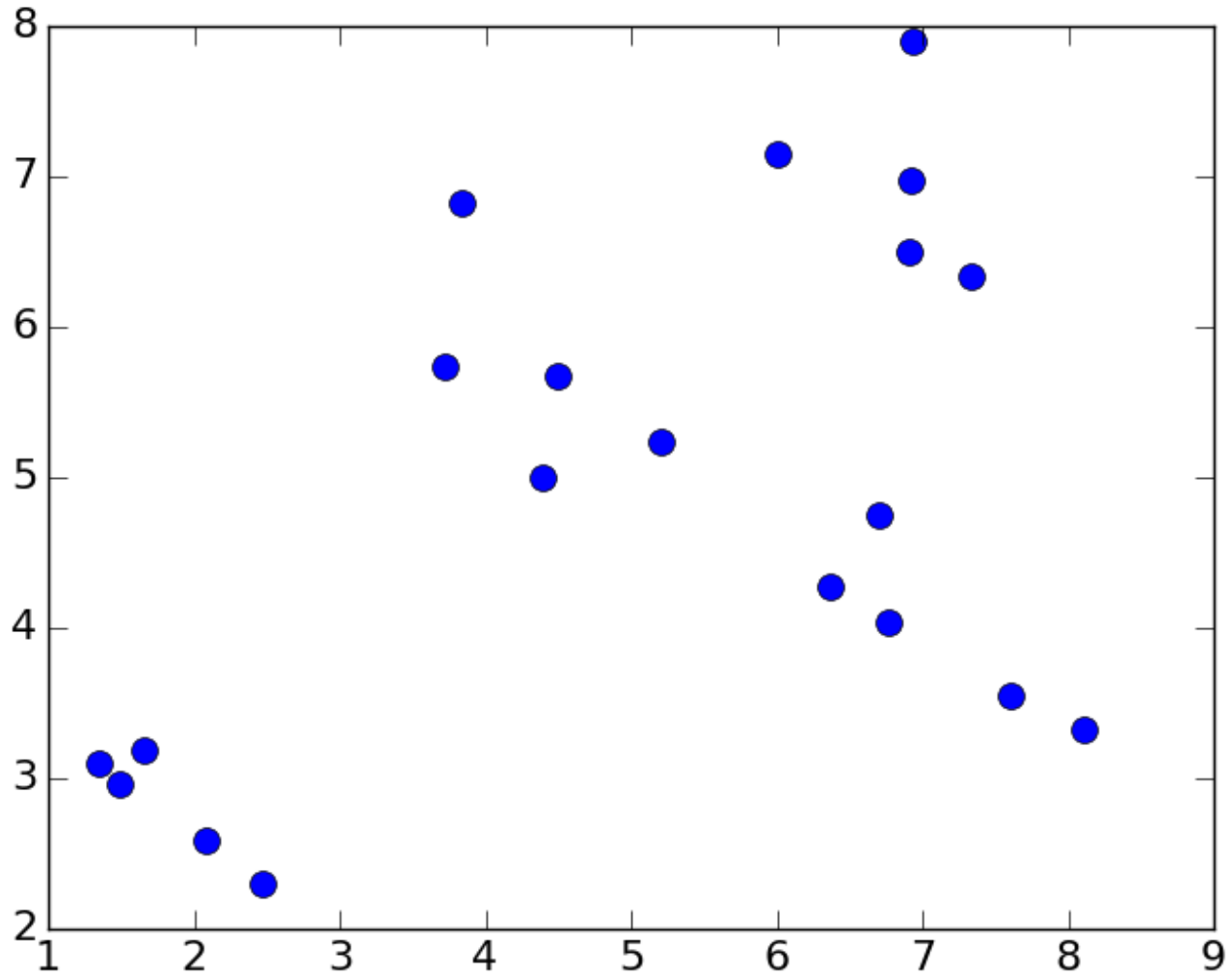



```
randomly chose k examples as initial centroids
while true:
    create k clusters by assigning each
        example to closest centroid
    compute k new centroids by averaging
        examples in each cluster
    if centroids don't change:
        break
```

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

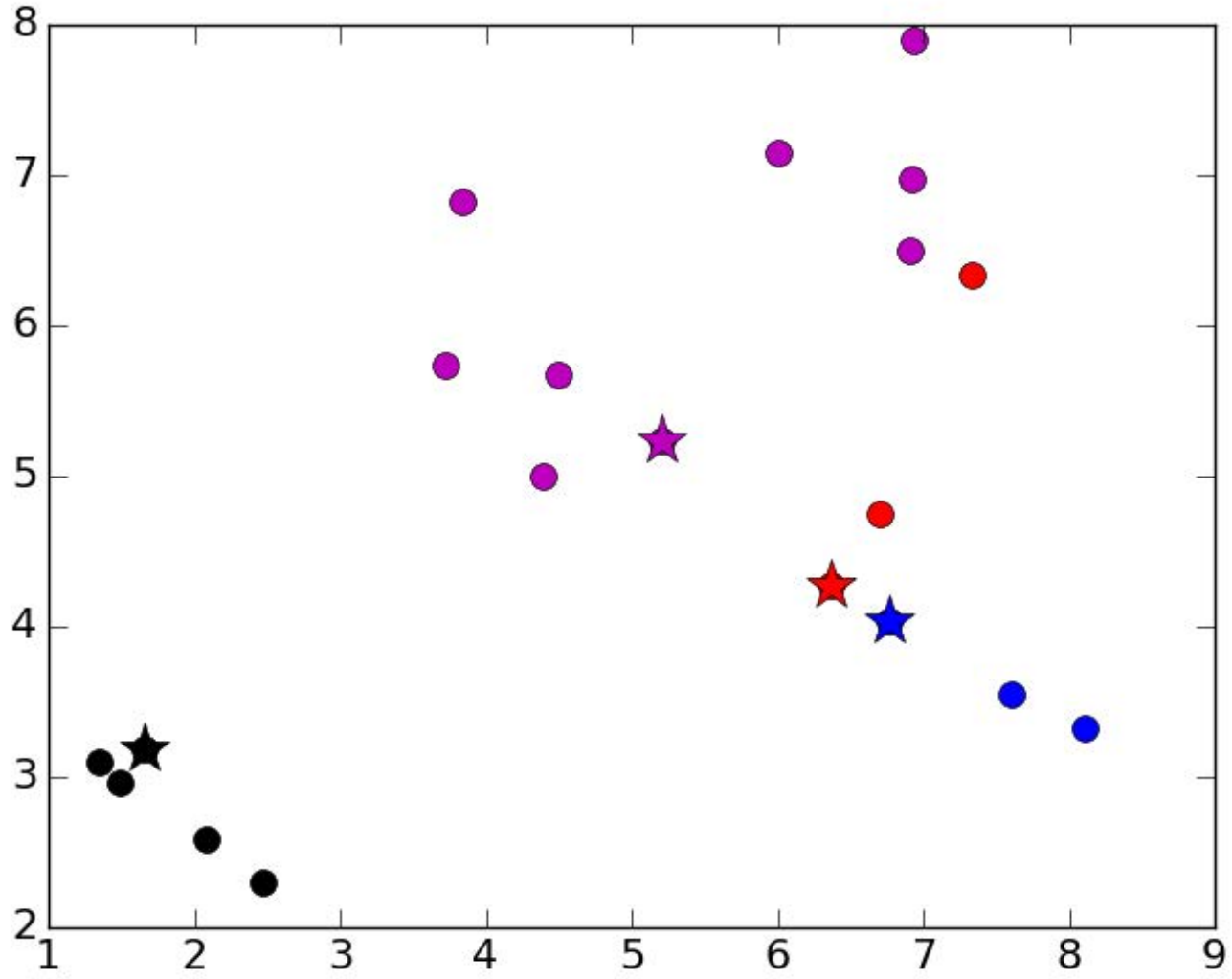
An Example



K = 4, Initial Centroids

Choose centroids at random;

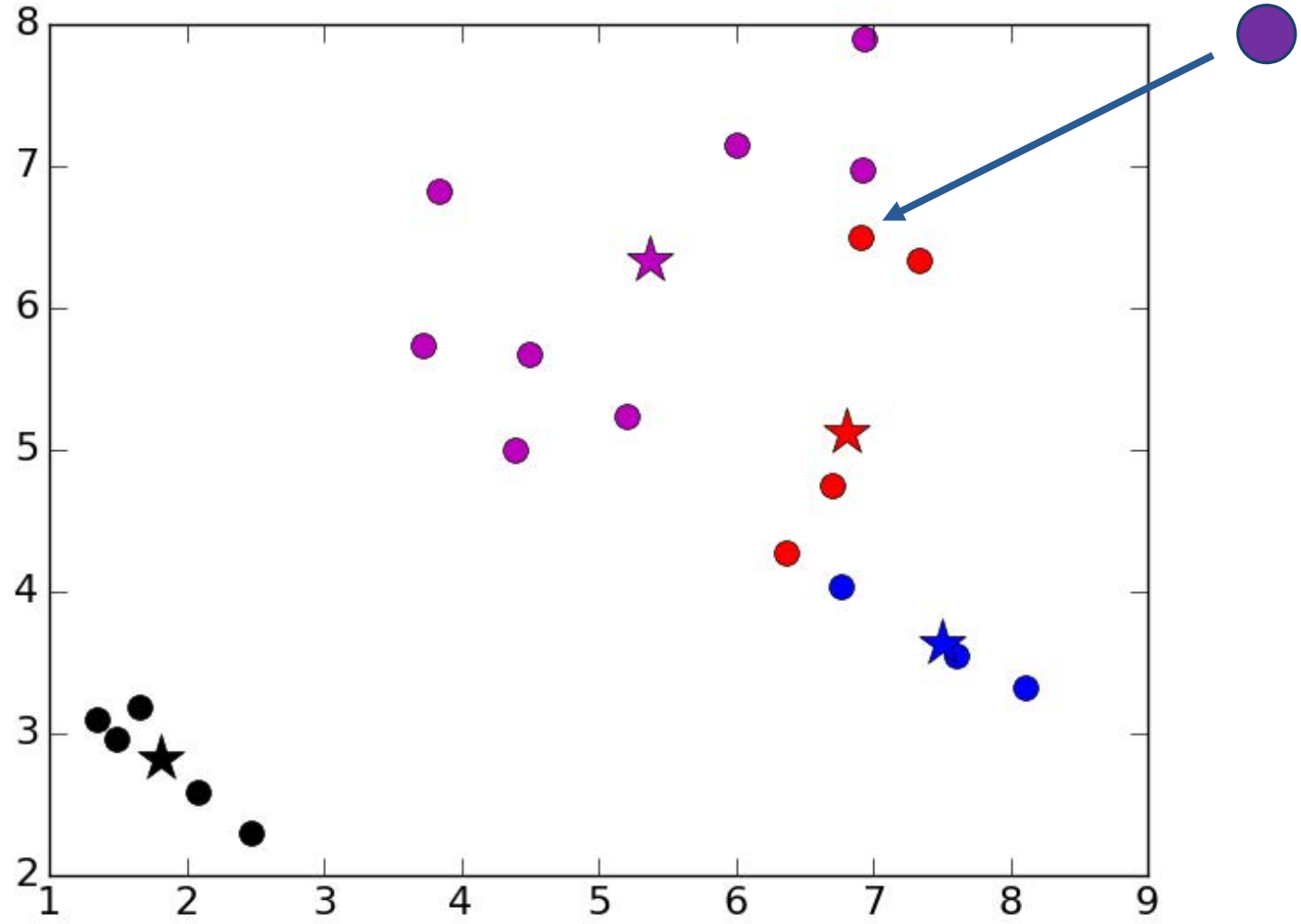
Assign each example to nearest centroid



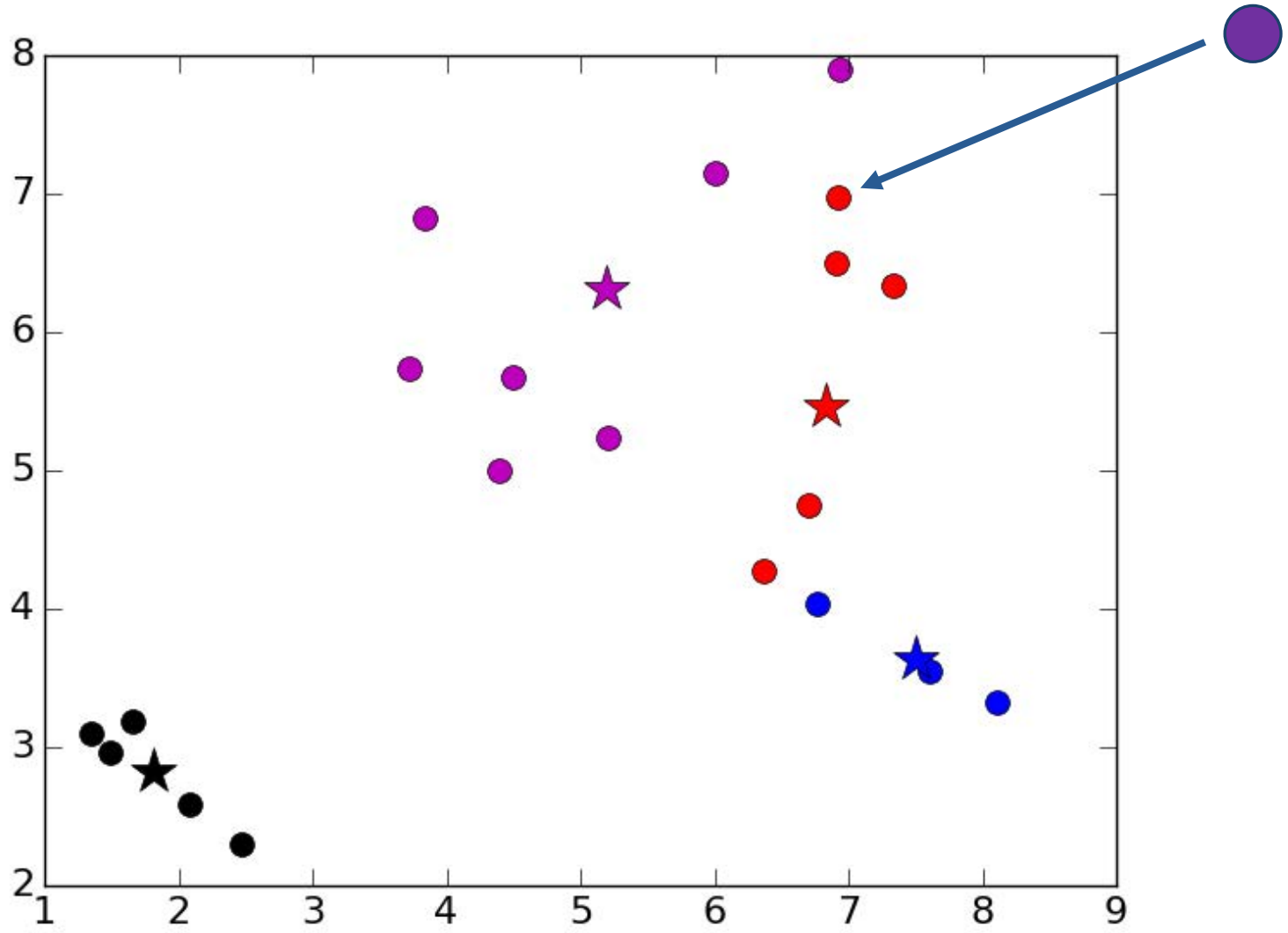
Iteration 1

Compute new centroids – will not correspond to examples

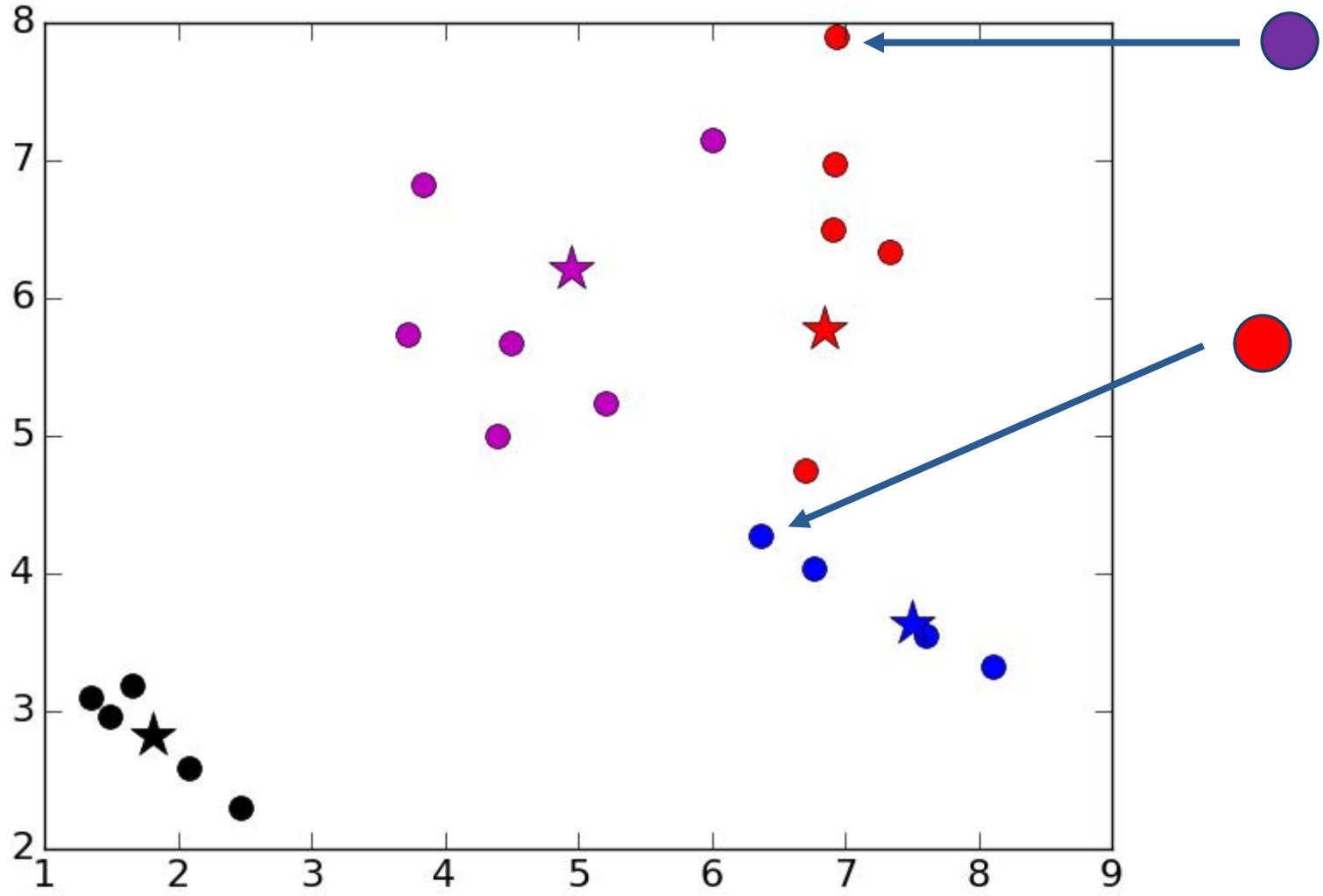
Assign each example to nearest centroid, note switch in label



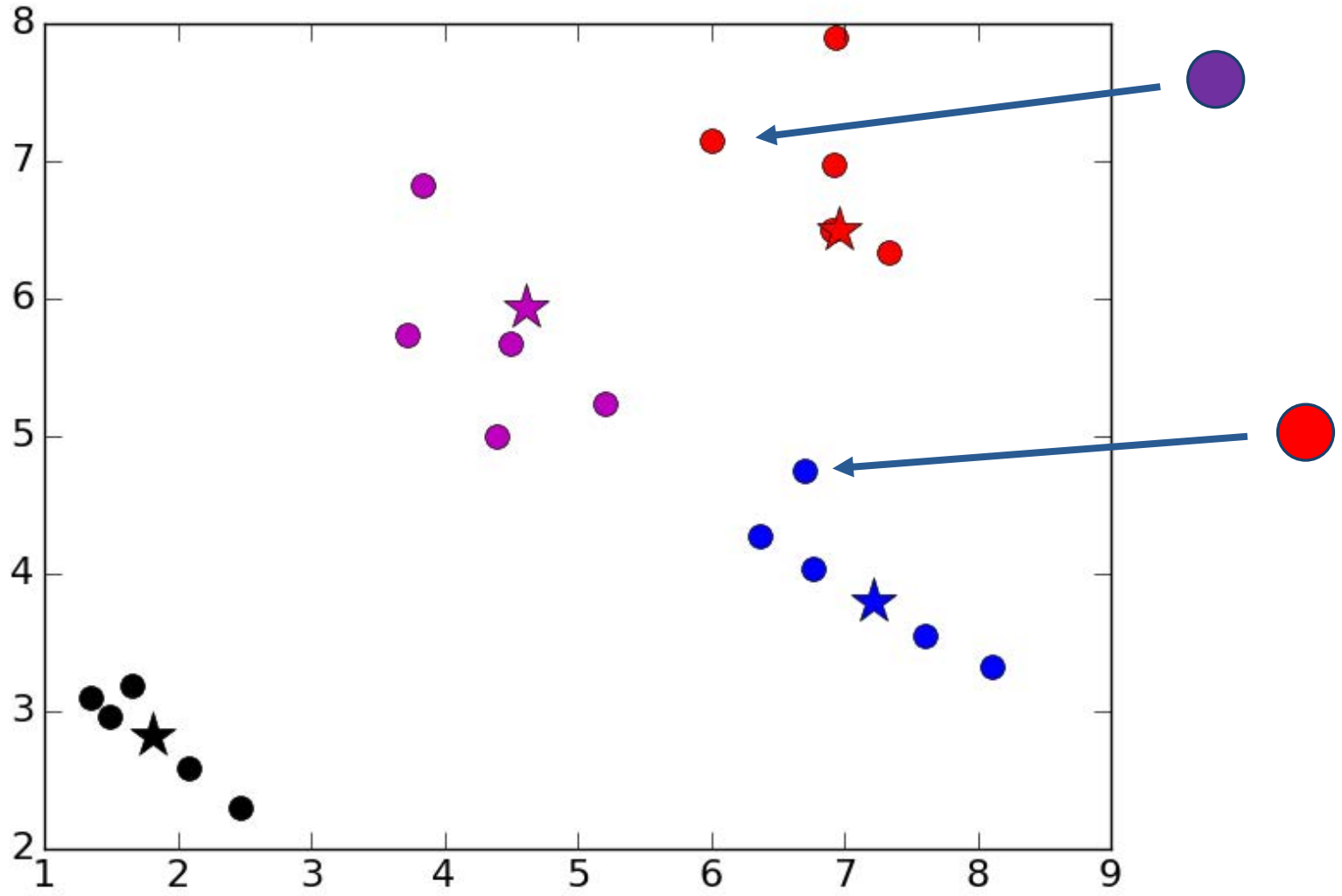
Iteration 2



Iteration 3

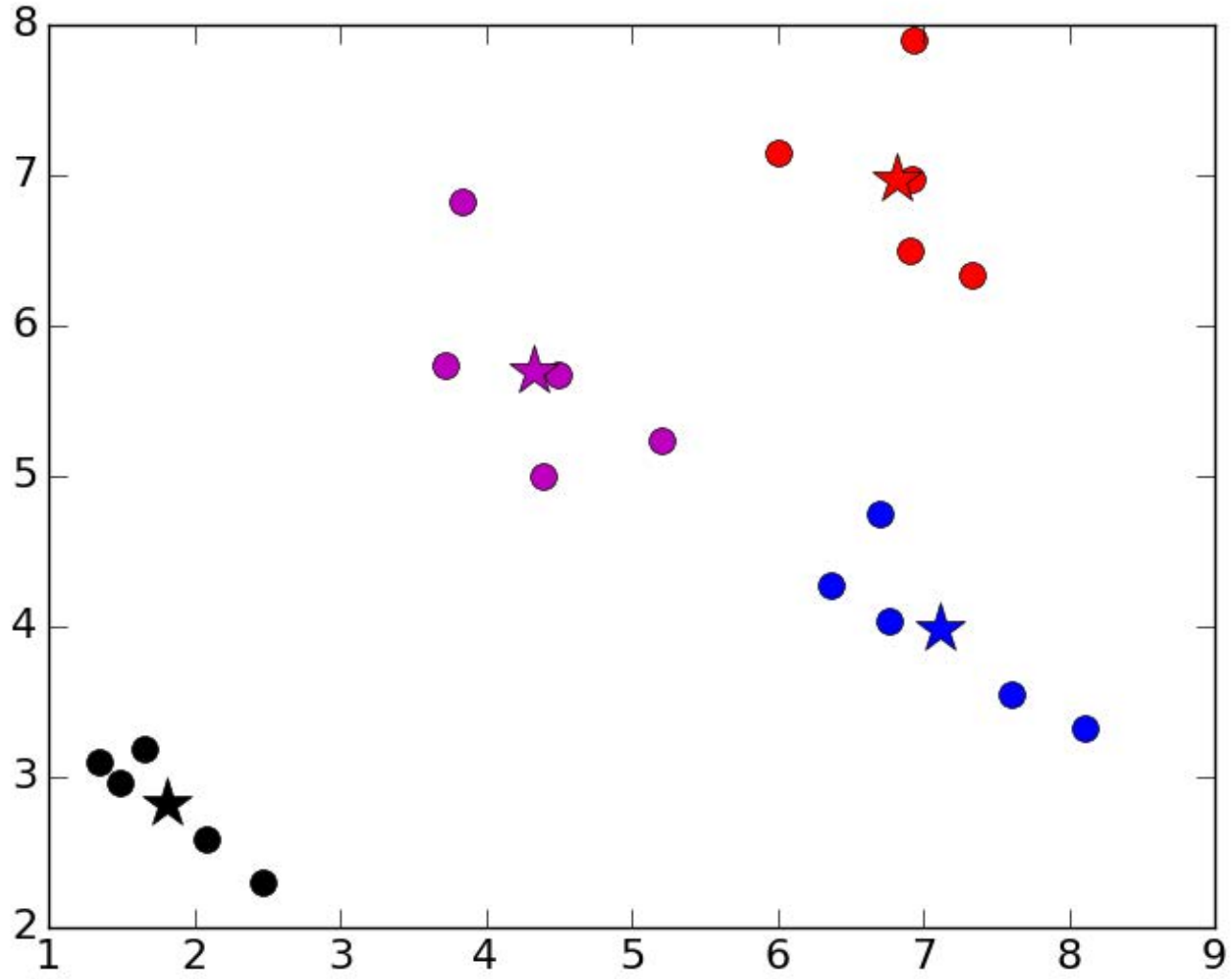


Iteration 4



Iteration 5

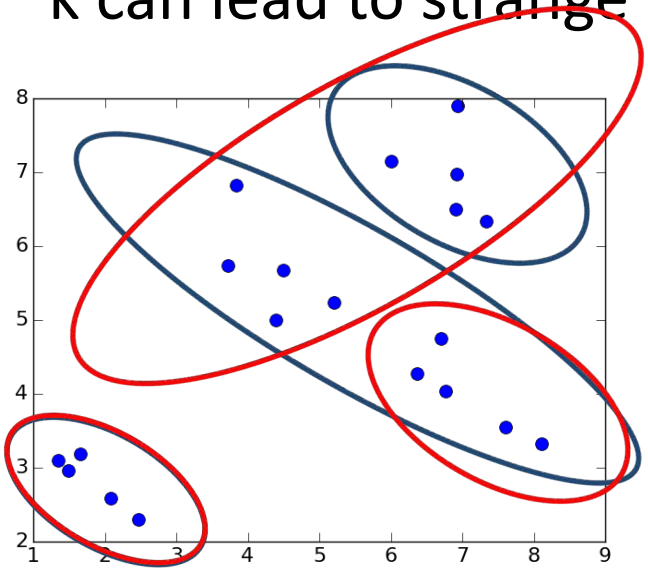
Centroids move, but no examples switch – can halt



Poll

Issues with k-means

- Choosing the "wrong" k can lead to strange results
  - Consider k = 3
  -

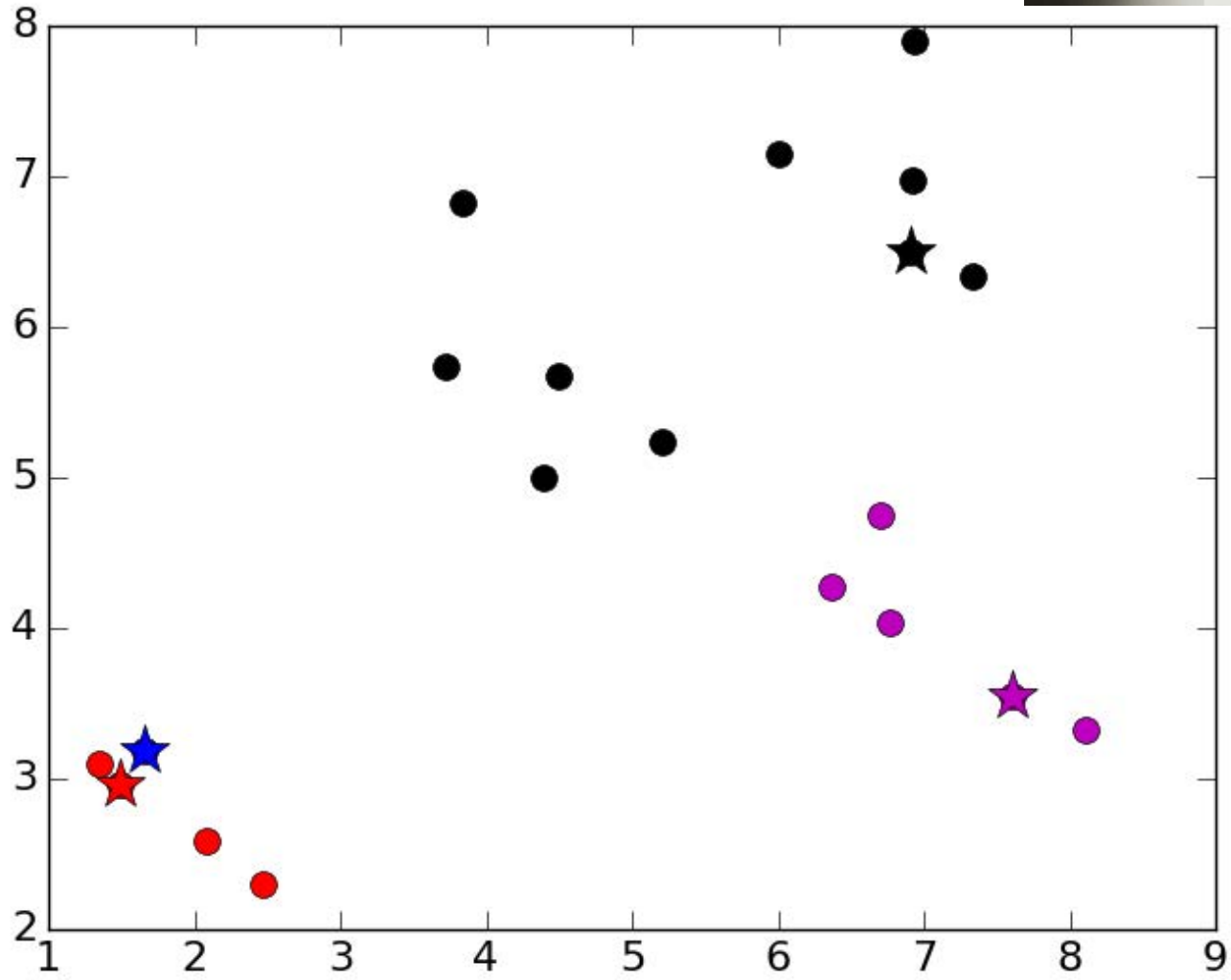


- Result can depend upon
  - Choice of initial centroids
  - Number of iterations
  - Greedy algorithm can find different local optima

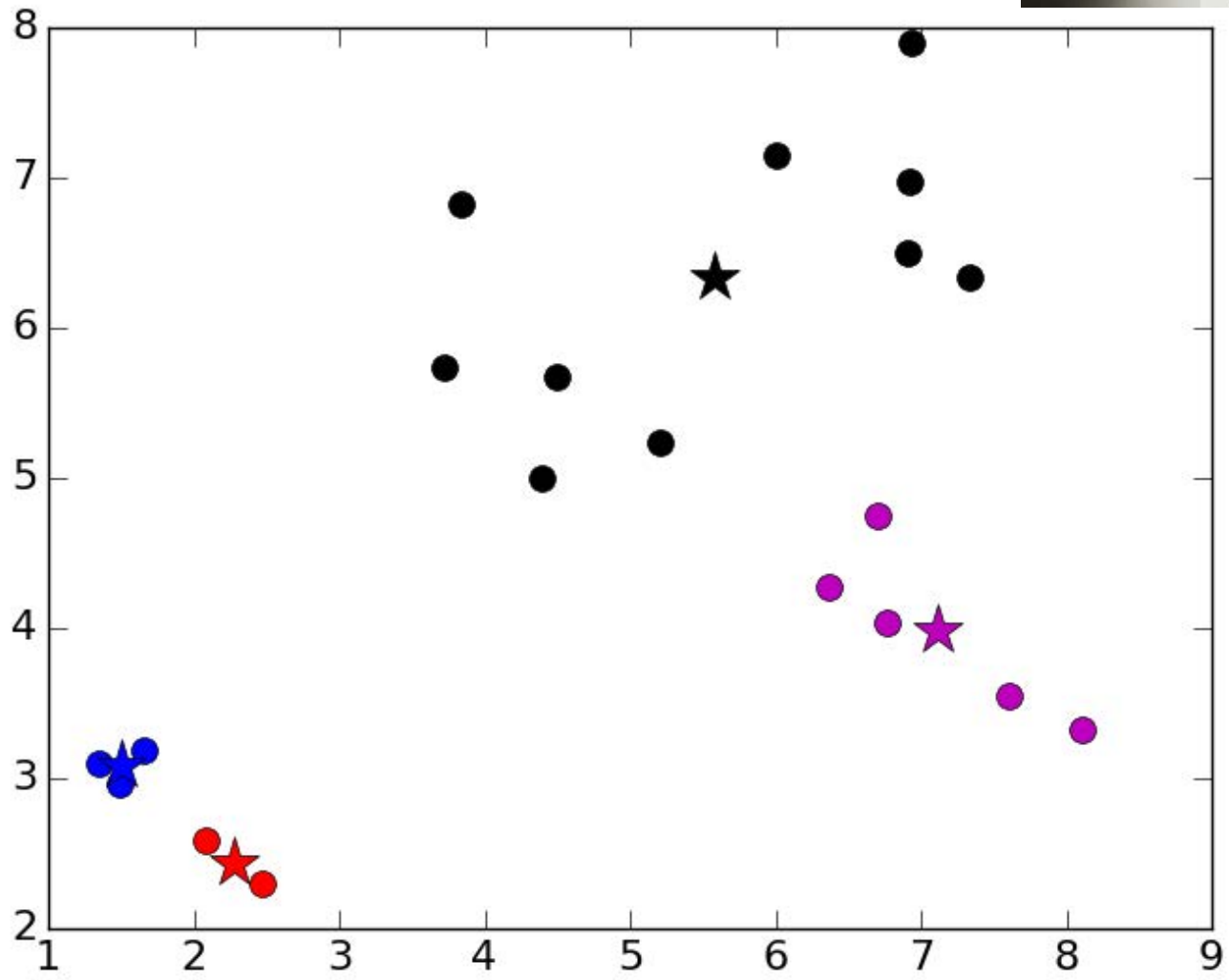
How to Choose K

- *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
    - Find a natural selection for number of clusters

Unlucky Initial Centroids



Converges On



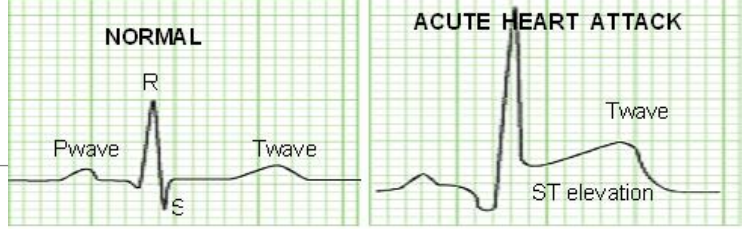
Mitigating Dependence on Initial Centroids

Try multiple sets of randomly chosen initial centroids

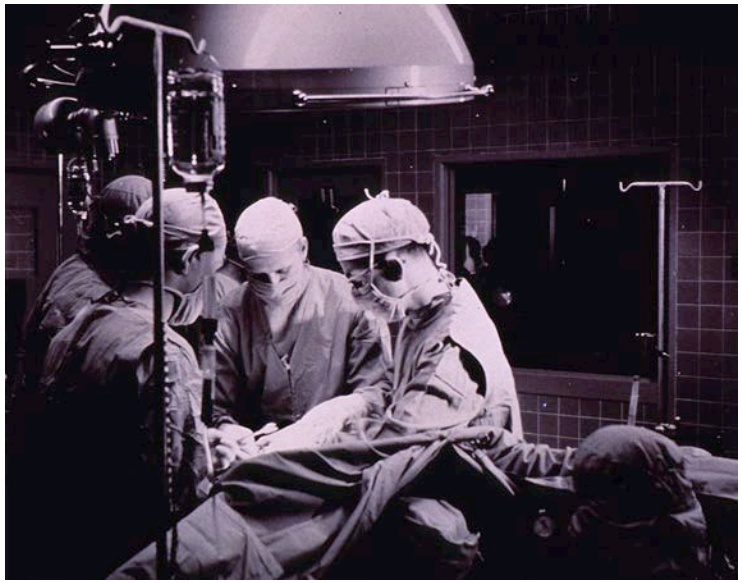
Select “best” result

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

An Example



- Risk of death from heart attack
- Hypothesis is that certain measurable factors may be predictive of mortality
- Is outcome (death) correlated with features?
  - Correlated but not causal
  - Probabilistic, not deterministic
    - Could have all four conditions indicative of mortality, but be okay
  - E.g., older people with multiple heart attacks at higher risk, but not deterministic
- Approach: Cluster based on attribute values; examine purity of clusters relative to outcomes

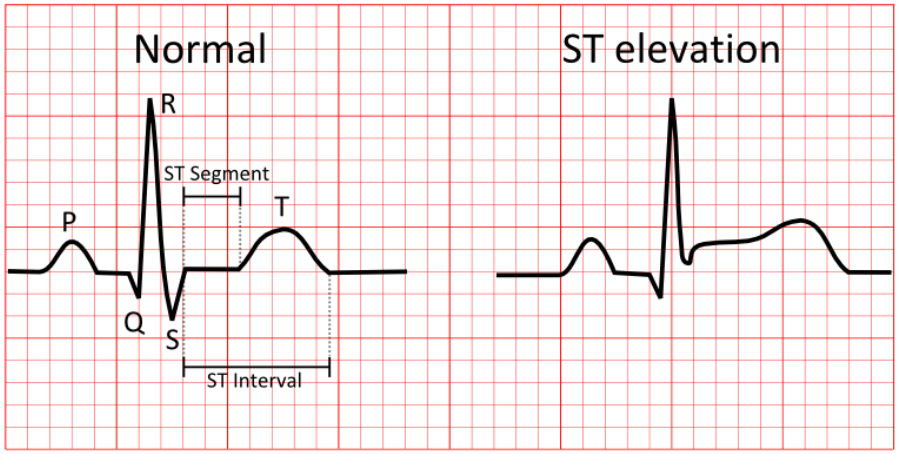


Why cluster?

- Are there sub-populations that might emerge from the data?
- Could these reflect different variants of disease?

An Example

- Data
  - Large number of patients
    - 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
- Approach
  - Cluster
  - 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

Outcome: 1 means mortality

Want to see if can correlate outcome with values of attributes; but not used in clustering

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)

    def __str__(self):
        return 'f{self.name}:{self.features}:{self.label}'
```

Class Cluster

```
class Cluster(object):

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

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

Class Cluster, cont.

```
def computeCentroid(self):
    vals = np.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
```

A generator: see discussion in textbook

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

```
class Patient(Example):
    pass

def getData():
    #read in data
    hrList, stElevList, ageList, prevACSList, classList = [],[],[],[],[]
    cardiacData = open('cardiacData.txt', 'r')
    for l in cardiacData:
        l = l.split(',')
        hrList.append(int(l[0]))
        stElevList.append(int(l[1]))
        ageList.append(int(l[2]))
        prevACSList.append(int(l[3]))
        classList.append(int(l[4]))
    #Build points
    points = []
    for i in range(len(hrList)):
        features = np.array([hrList[i], prevACSList[i],\
                                    stElevList[i], ageList[i]])
        pIndex = str(i)
        points.append(Patient('P'+ pIndex, features, classList[i]))
    return points
```

kmeans

```
def kmeans(examples, k, verbose = False):
    #Get k randomly chosen initial centroids, create cluster for each
    ...

    #Iterate until centroids do not change

        #Create a list containing k distinct empty lists

        #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

    return clusters

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"""
    posFracs = []
    for c in clustering:
        numPts = 0
        numPos = 0
        for p in c.members():
            numPts += 1
            if p.getLabel() == 1:
                numPos += 1
        fracPos = numPos/numPts
        posFracs.append(fracPos)
        print(f'Cluster of size {numPts}, frac. pos. = {fracPos:.3f}, ' +
              f'num. pos. = {numPos}')
    return np.array(posFracs)
```

Examining Results

```
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)
```

Run It

What do you think? Good result?

What's the Problem

- Features at vastly different scales
- ST elevation binary
- Number of heart attacks low single digits
- Heart rate double or triple digits

Scaling

```
def scaleData(vals):
    #assumes vals an array of numbers
    mean = sum(vals)/len(vals)
    sd = numpy.std(vals)
    vals = vals - mean
    return vals/sd
```

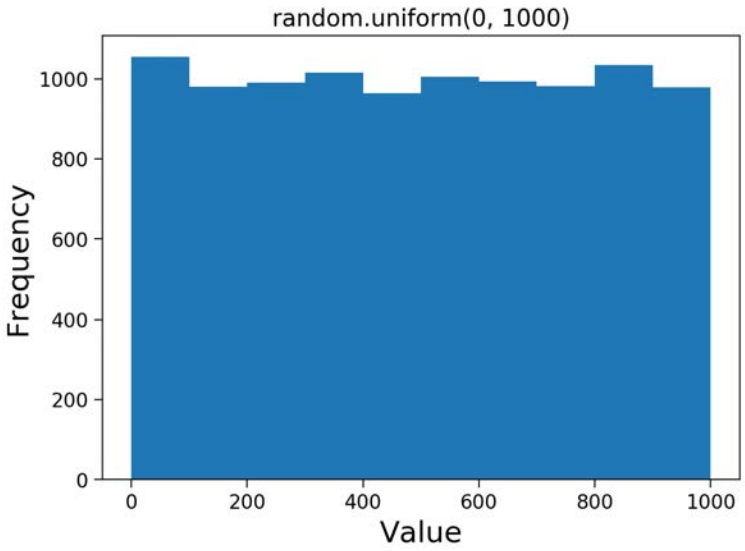
Poll

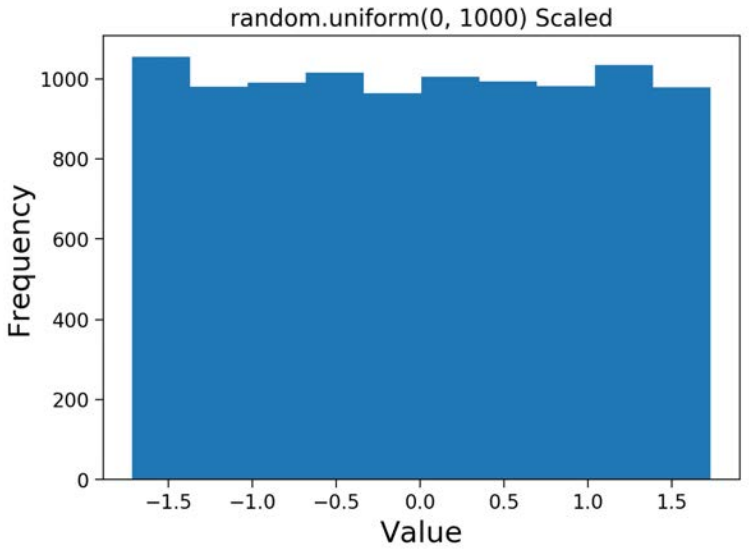
Z-Scaling

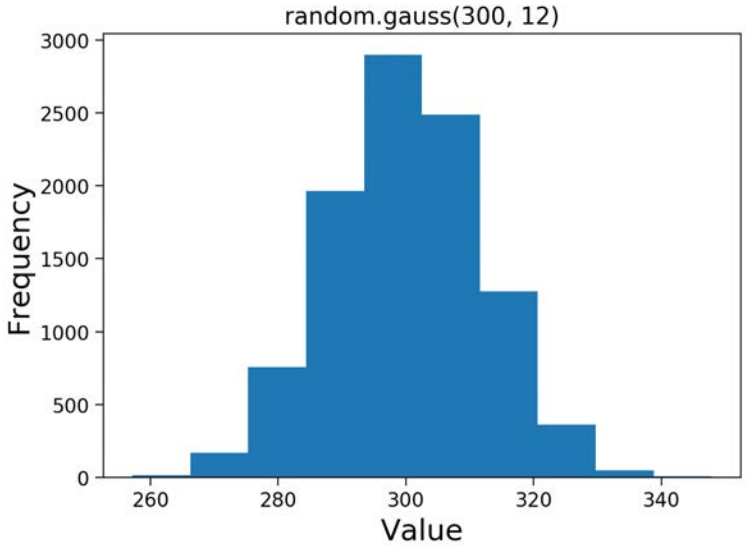
Mean = ?

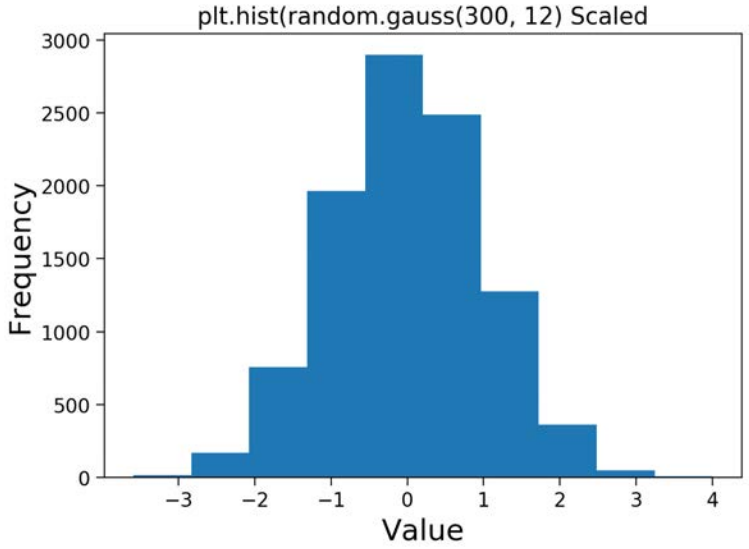
Std = ?

What Z-scaling Does









Scaling Data

```
def getData(toScale = False):
    #read in data
    ...
    if toScale:
        hrList = scaleData(hrList)
        stElevList = scaleData(stElevList)
        ageList = scaleData(ageList)
        prevACSList = scaleData(prevACSList)
    #Build points
    ...
    return points
```

Result of Running It with Scaling

Try patients = getData(True) #scale features

Test k-means (k = 2)
Cluster of size 224, frac. pos. = 0.290, num. pos. = 65
Cluster of size 26, frac. pos. = 0.692, num. pos. = 18

Happy with sensitivity? Or at least happier?
Where are most of the deaths?

One Way to Think About Question

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

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

Percentage correctly found
Percentage correctly rejected

Cluster of size 224 with fraction of positives = 0.2902

Cluster of size 26 with fraction of positives = 0.6923

If we use cluster membership to classify, then

- first cluster has sensitivity of .78 but specificity of .05
- second cluster has sensitivity of .22 but specificity of .95

Picking second cluster misses most of the positives

A Hypothesis

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

```
patients = getData(True)
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, frac. pos. = 0.290, num. pos. = 65
Cluster of size 26, frac. pos. = 0.692, num. pos. = 18

Test k-means (k = 4)
Cluster of size 26, frac. pos. = 0.692, num. pos. = 18
Cluster of size 86, frac. pos. = 0.081, num. pos. = 7
Cluster of size 76, frac. pos. = 0.711, num. pos. = 54
Cluster of size 62, frac. pos. = 0.065, num. pos. = 4

Test k-means (k = 6)
Cluster of size 49, frac. pos. = 0.020, num. pos. = 1
Cluster of size 26, frac. pos. = 0.692, num. pos. = 18
Cluster of size 45, frac. pos. = 0.089, num. pos. = 4
Cluster of size 54, frac. pos. = 0.093, num. pos. = 5
Cluster of size 36, frac. pos. = 0.778, num. pos. = 28
Cluster of size 40, frac. pos. = 0.675, num. pos. = 27

Coming Up

- In the next lecture, we will see examples of supervised learning algorithms
- And there will be a quiz