

Curve Fitting and Introduction to Machine Learning

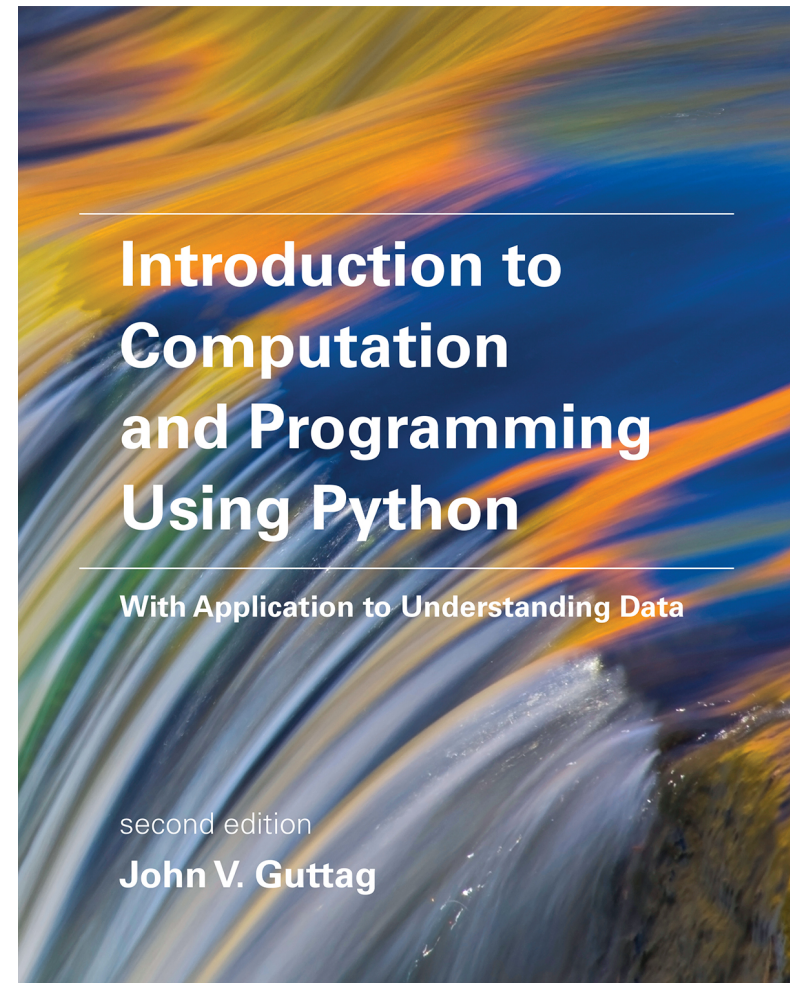


Eric Grimson

MIT Department of Electrical Engineering and
Computer Science


Assigned Reading

- Today:
 - Chapters 18, 22
- Next lecture:
 - Chapter 23
 - Sections 24.1-24.3



No Lecture on Wednesday





11.S188 / 11.S952

Hack the City:

Data Science for Public Good



Are we living under the risk of flooding?



How do trees reflect local quality-of-life and social injustice?



What are crime patterns in your neighborhood?



Where are Airbnb around us?
Do they increase local rent?

This workshop teaches how to apply data science for public good in Cambridge. Through hands-on exercises and tutorials, you will build mini projects with fun real-world data, create playful visualization, and formulate data science questions to discover and support local community.

Class Meetings: Jan 22-31, 2-4 pm

Credits: 3 credits

Room: 9-450

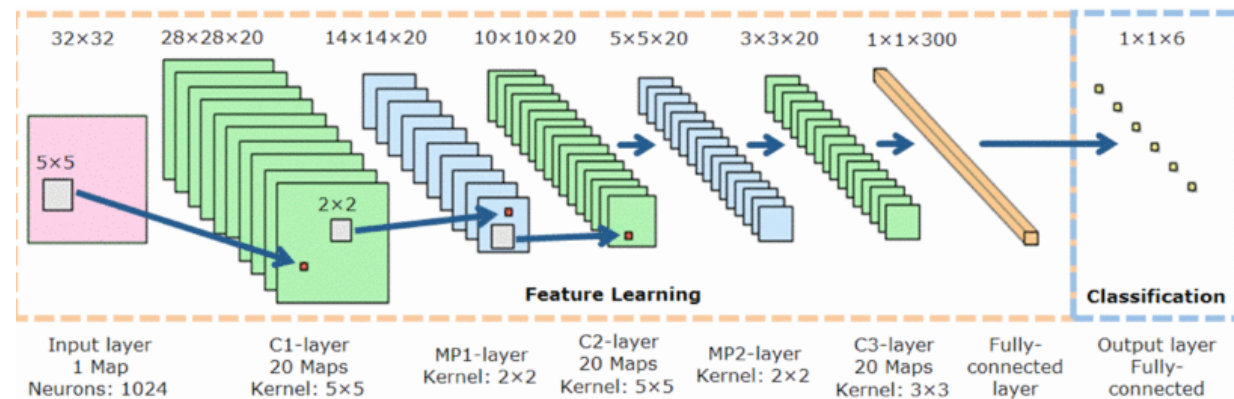
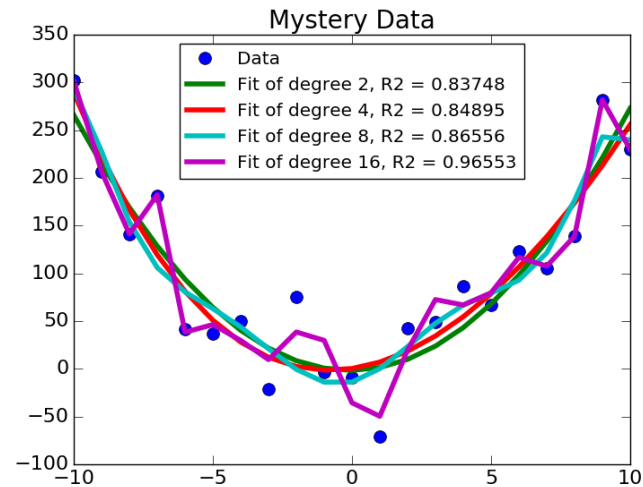
Prerequisites: 6.00 or permission of instructor

Instructors: Yuan Lai (DUSP) Nina Lutz (Media Lab)

U
R
B
A
N
C
S
C
O
U
R
S
E
ELEVEN+ SIX
F

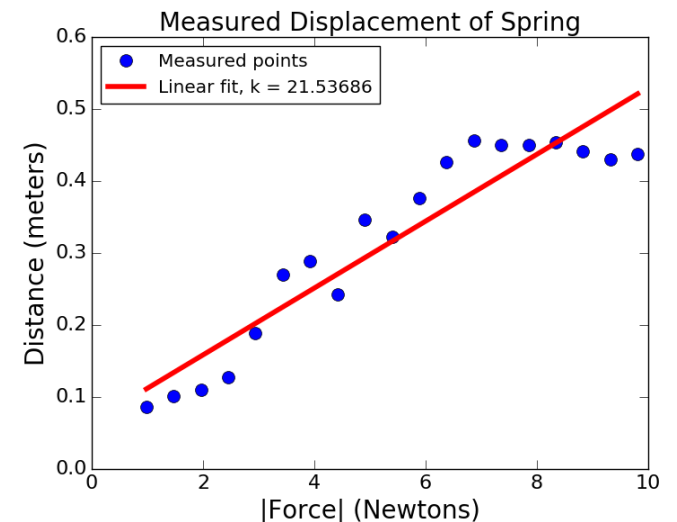
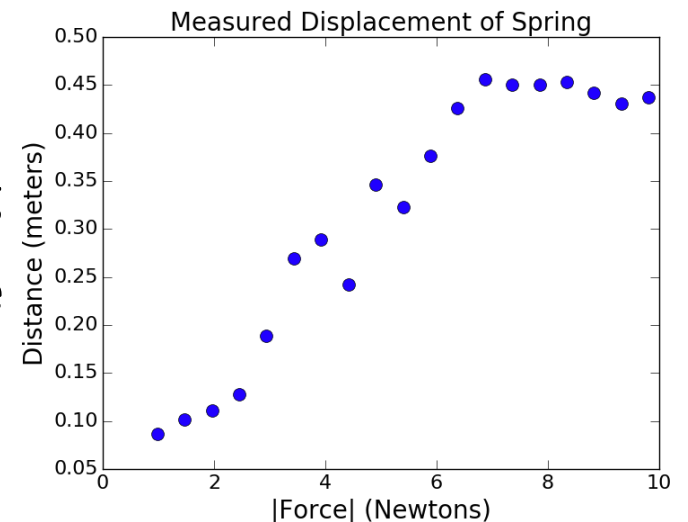
Two Topics Today

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

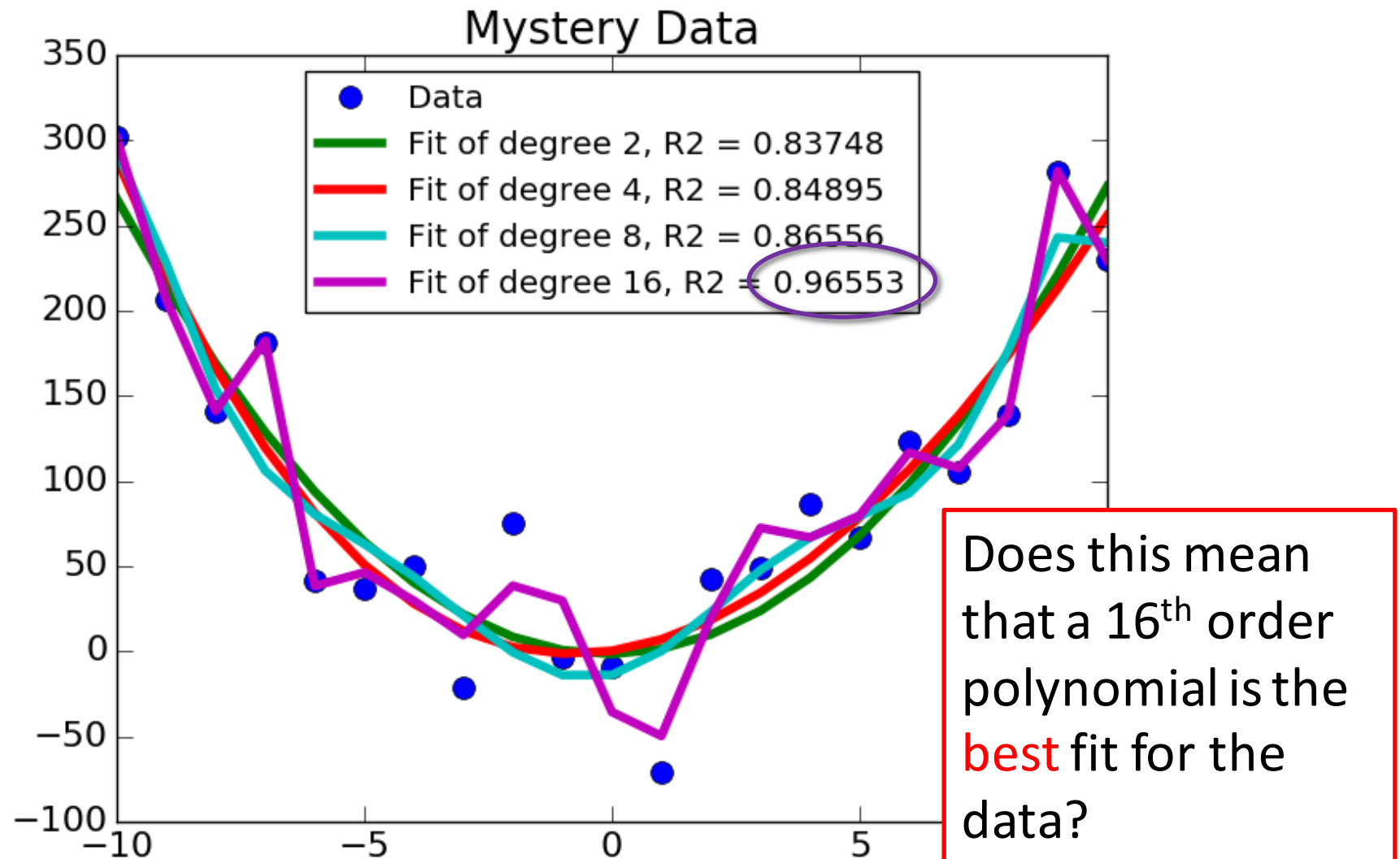


Why Fit Curves?

- We have a set of numerical data, relating observed values with different inputs (e.g., spring displacement versus expansion force)
- When we **fit** a curve to the data, we are finding a relationship between an independent variable (mass or force) and an estimated value of a dependent variable (distance or displacement)
- To decide how well a curve fits, we need a way to measure the goodness of fit –called the **objective function** (e.g., least squares)
- Given the objective function, we also need an algorithm to find the curve that minimizes it (we used linear regression to find best polynomial of a given order)
- Result in our example is a curve that predicts displacement as function of force
- R^2 (coefficient of determination) measures quality of fit



Four Curves Fit to the Same Data

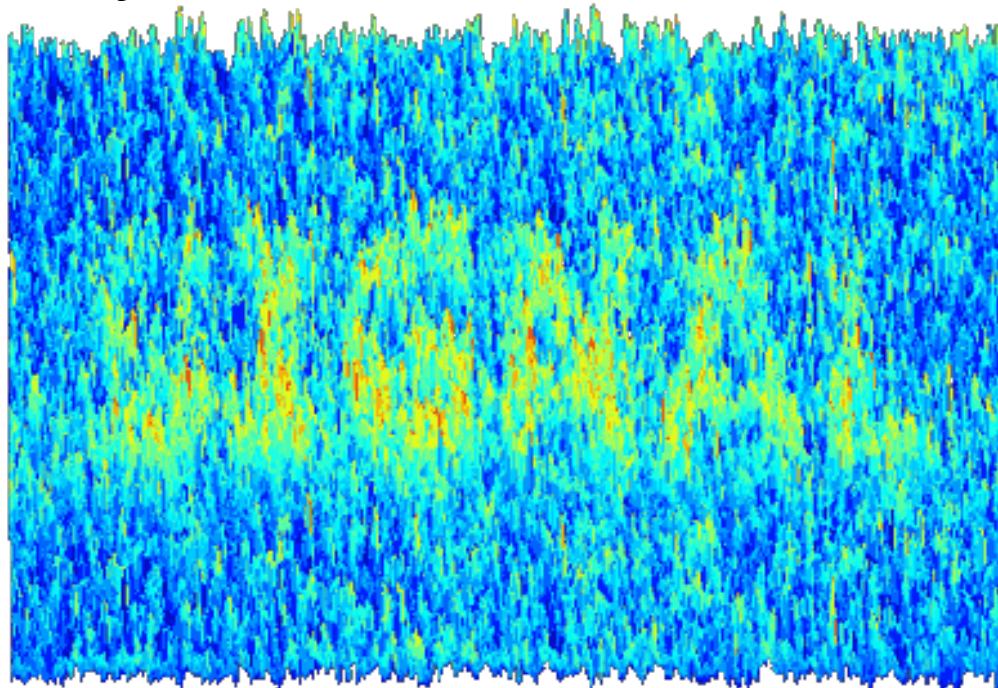


Does Tightest = Best?

- 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 new patient
- A good model helps us do both of these things

The Key Question

- If a model helps us to understand underlying process and to predict responses to new inputs, then need to ask: to what extent is the model actually shaped by the underlying process we are trying to understand?
- When the model is complex, it runs the risk of fitting the noise, not just the data



Training versus Testing

- One way to separate out impact of noise on our model is to take advantage of fact that each time we sample a system:
 - Signal will be roughly the same
 - Noise will typically be different (if it is random)
- So if we can make multiple trials, we might be able to separate effect of noise from underlying signal
- Use one 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')
```

Create two different data sets of same system

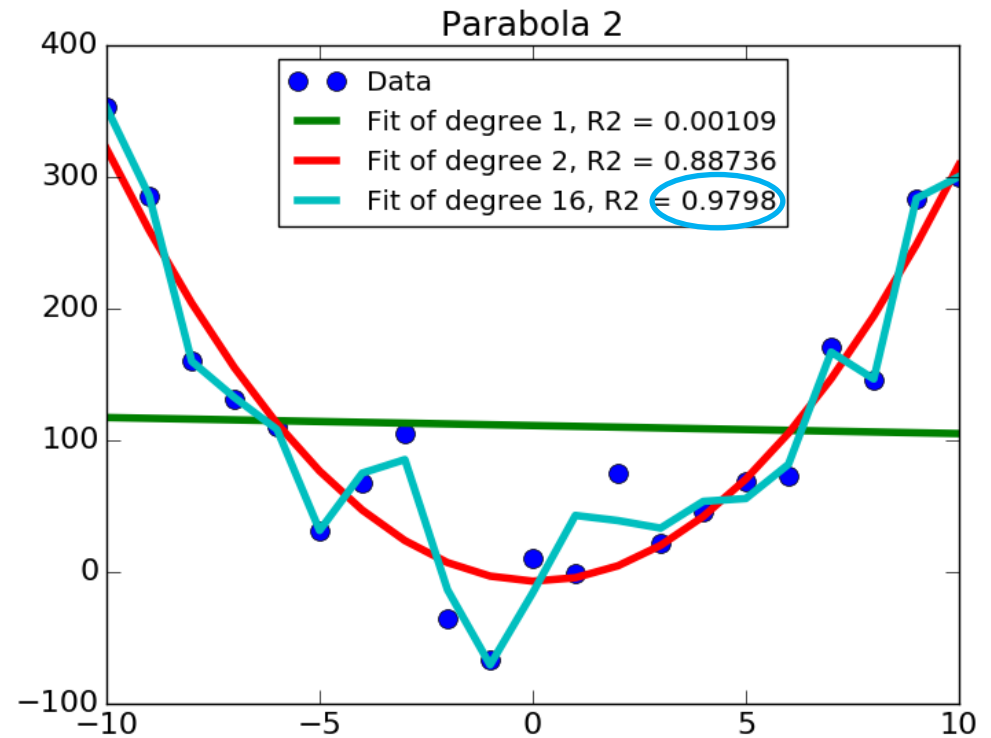
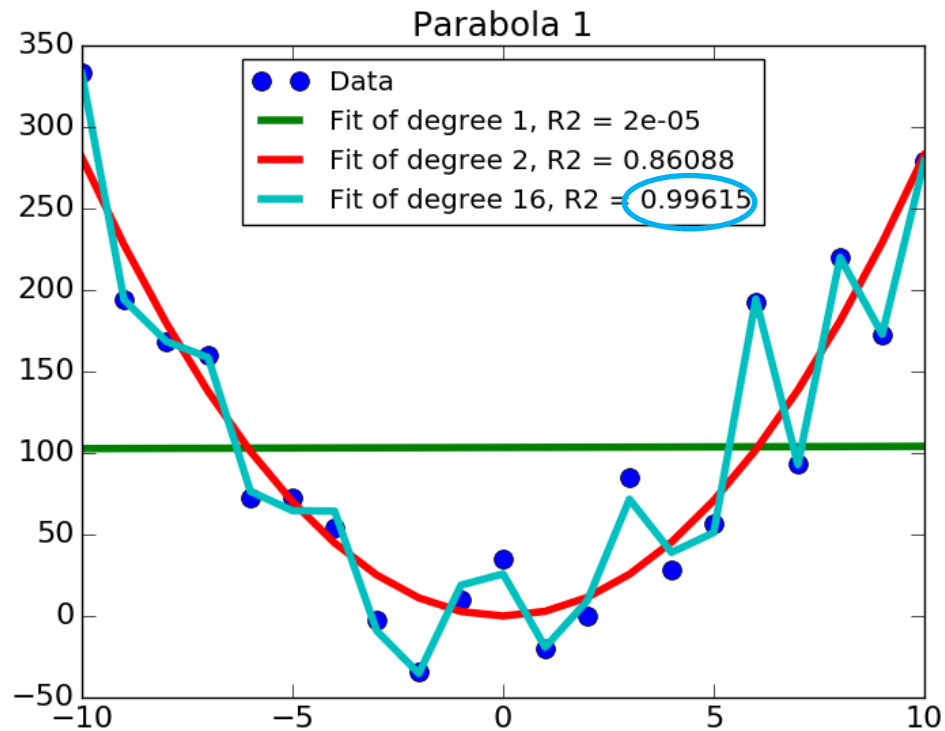
```
degrees = (1, 2, 16)
xVals1, yVals1 = getData('parabola1.txt')
models1 = genFits(xVals1, yVals1, degrees)
testFits(models1, degrees, xVals1, yVals1, 'Parabola 1')
```

Fit models to first data set

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

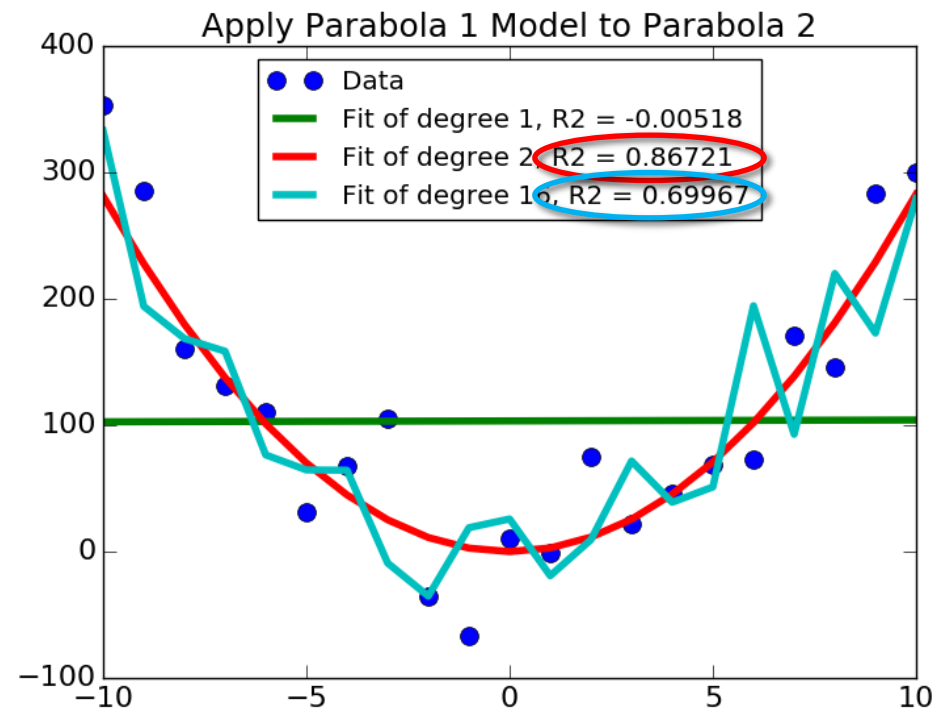
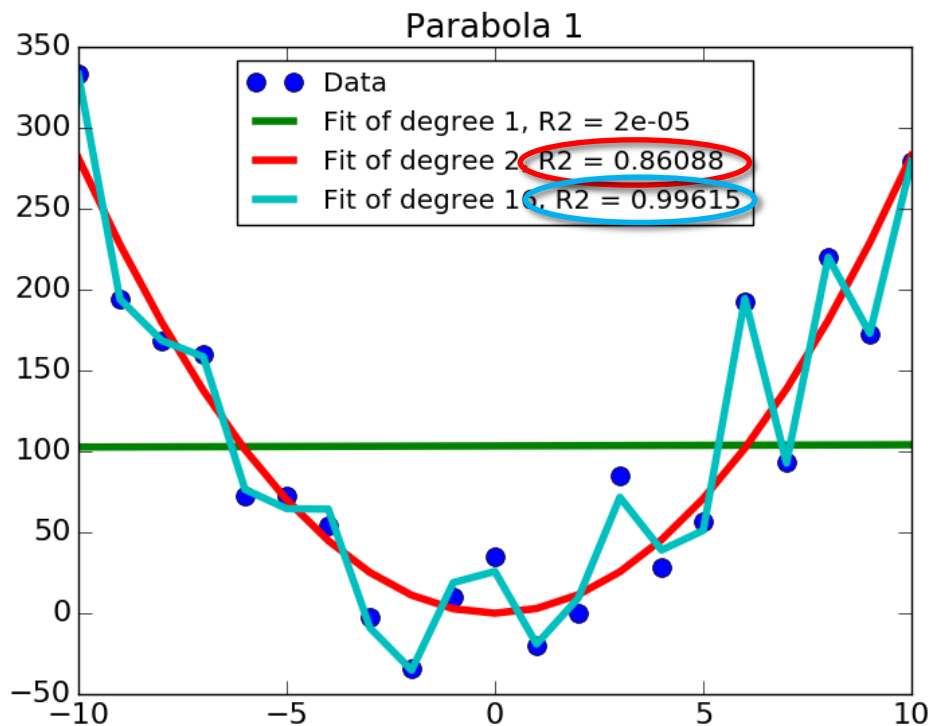
Fit models to second data set

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')
```

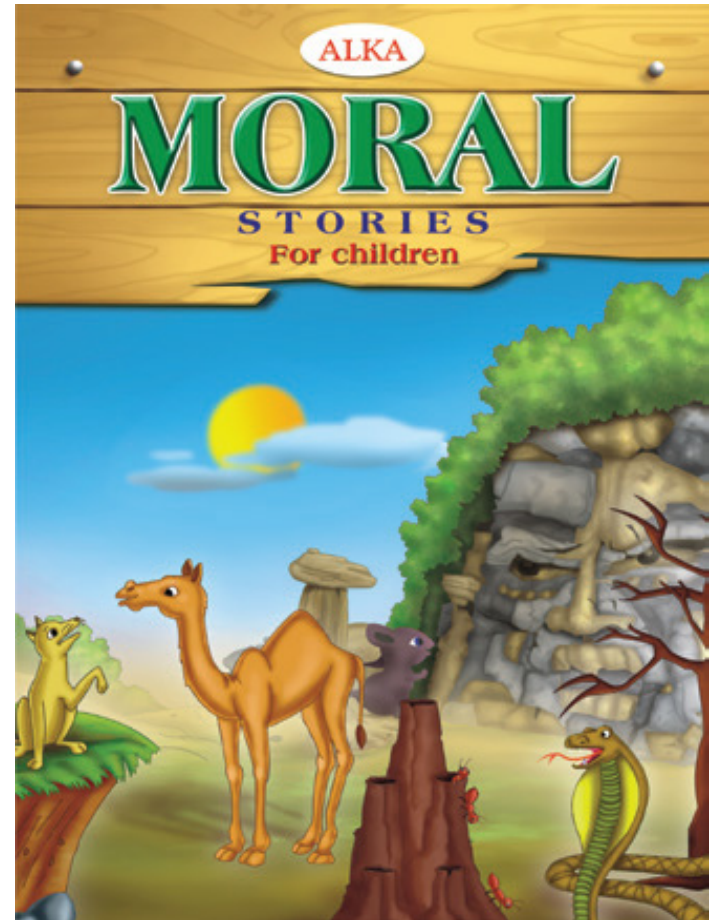


Fit models from first data set to training data

Fit models from first data set to test data

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 **hold out** set



Usually, 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)
```

Split the data into two sets:
one for training,
one for testing

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')
```

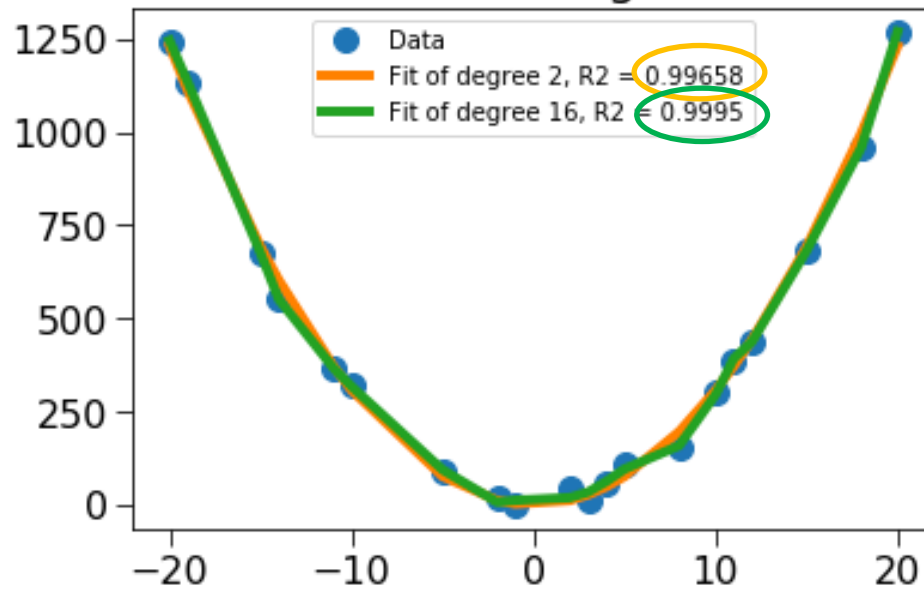
Build models based on training data

Test against training data

Test against test data

Validating a Model

Fit to Training Data



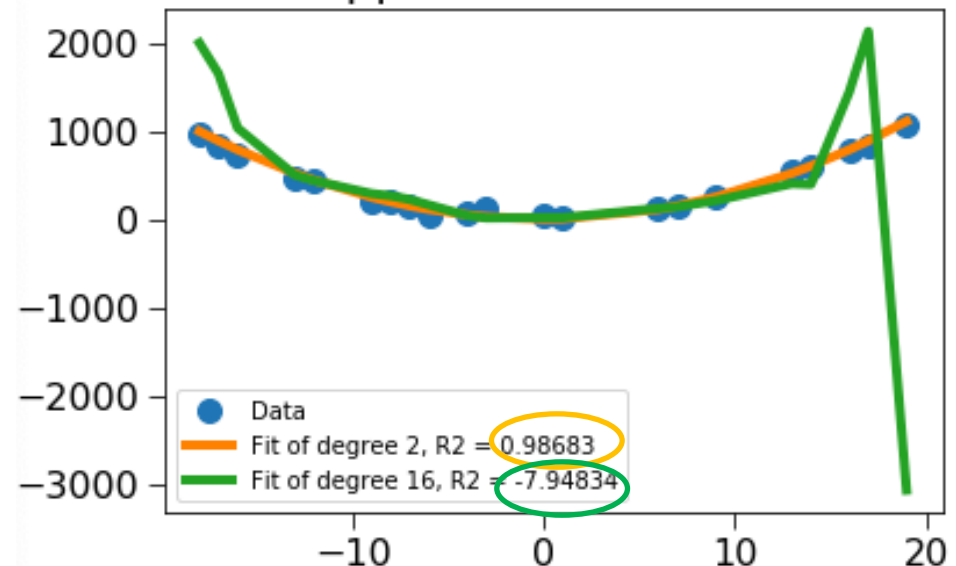
degree 2 model (rounded)

$$y = 3.08x^2 - 0.02x + 0.46$$

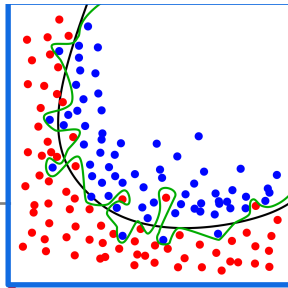
degree 16 model (rounded)

$$y = -0.01x^6 - 0.04x^5 + 0.44x^4 + 0.52x^3 - 2.24x^2 + 1.51x + 16.94$$

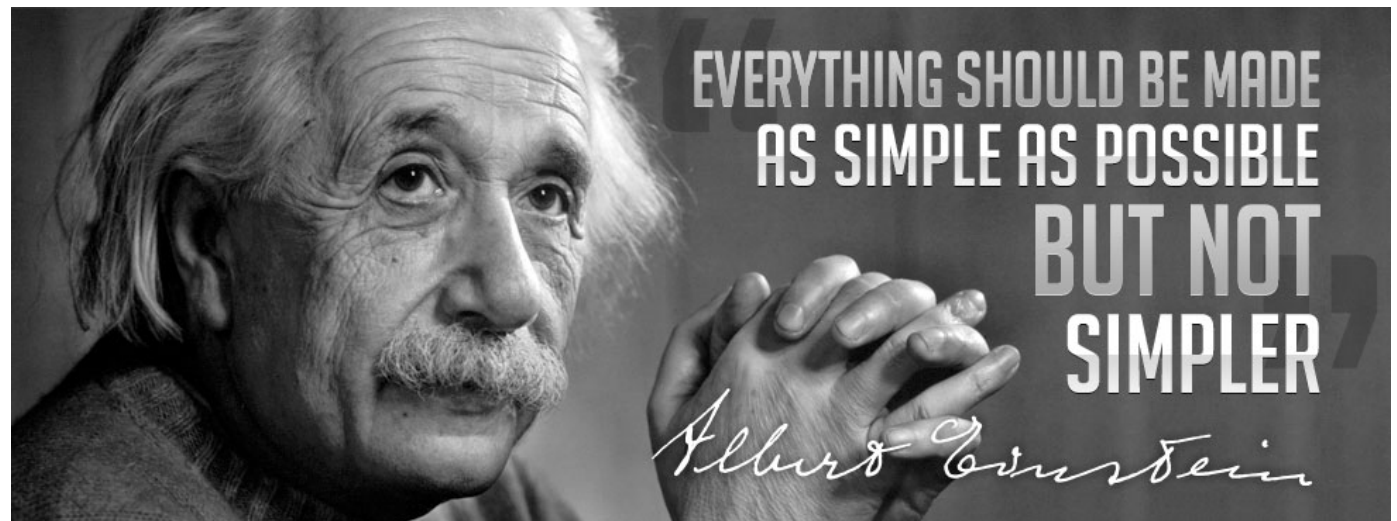
Applied to Test Data



The Take Home Message

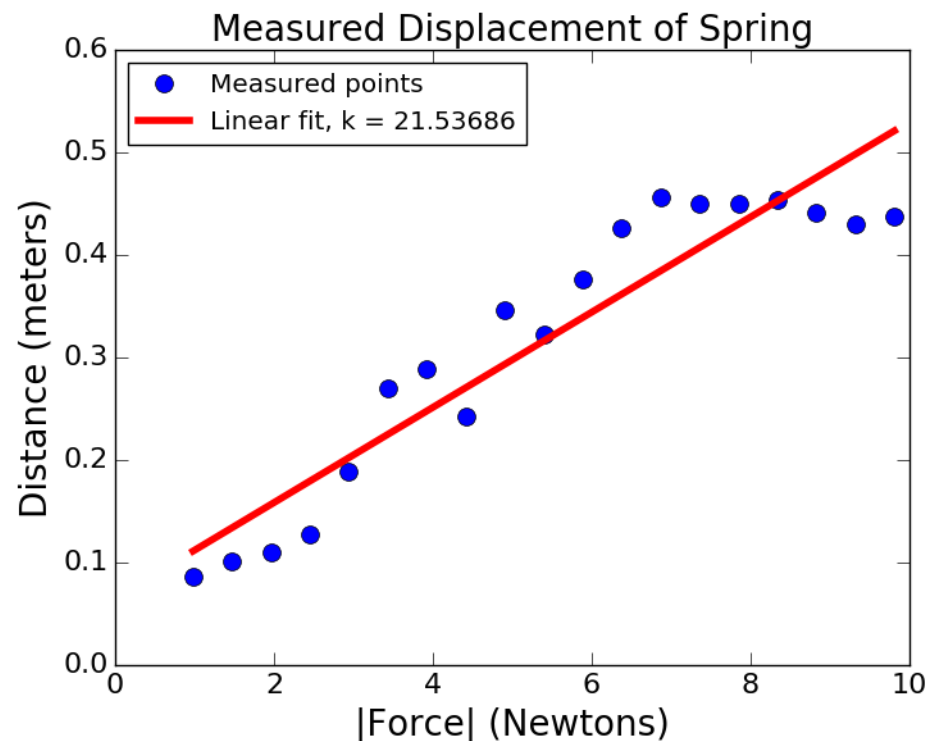


- Choosing an overly-complex model leads to **overfitting** to the training data
- Increases the risk of a model that works poorly on data not included in the training set
- On the other hand choosing an insufficiently complex model has other problems
 - As we saw when we fit a line to data that was basically parabolic



One Last Thought

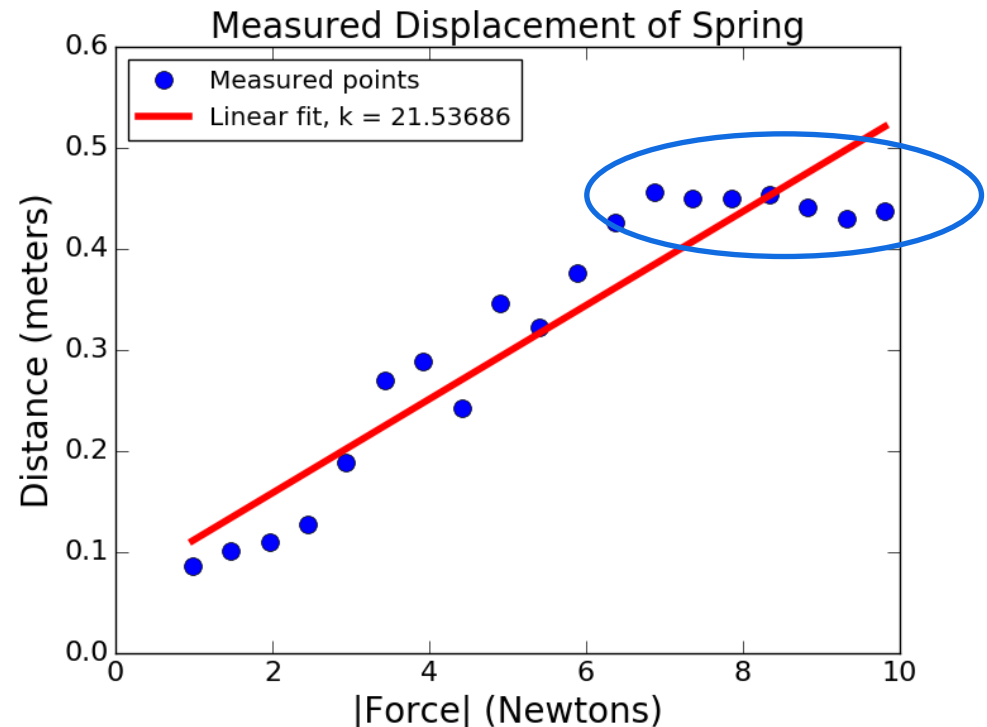
- Combining model information with goodness of fit can provide additional insight
- Consider the order 1 fit to the original spring data



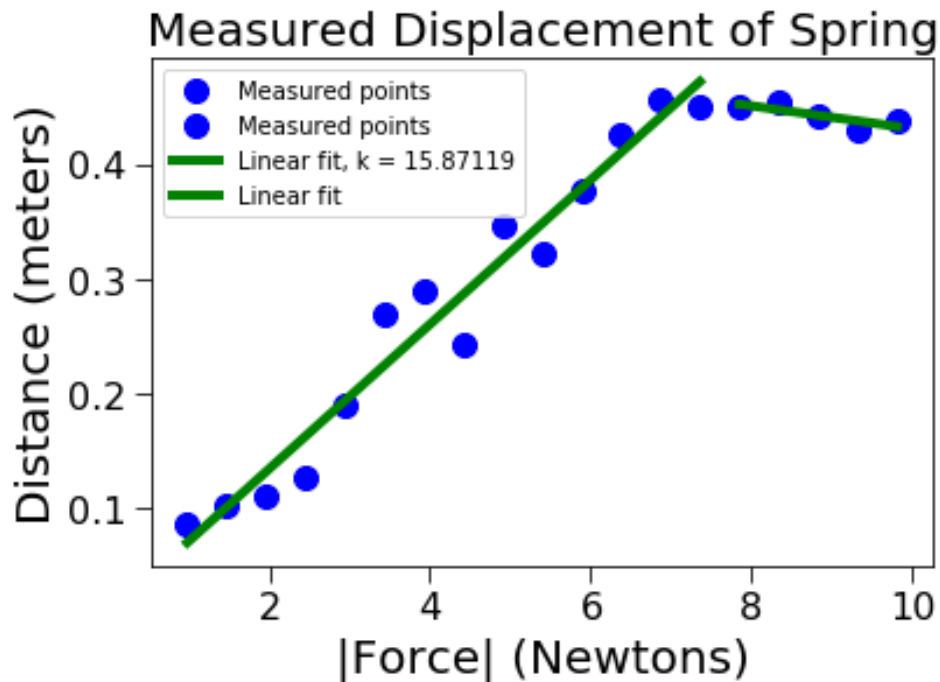
- R^2 value is .8815 – which is decent

One Last Thought

- Visual inspection suggests that perhaps some different process comes into play for large forces?
- Remember model said Hooke's law applied up to some maximum force
- Could search for point at which to break data into two sets, and fit models to both sets of data separately
 - Look for break that minimizes sum of residual error in both parts?



Seems Better?



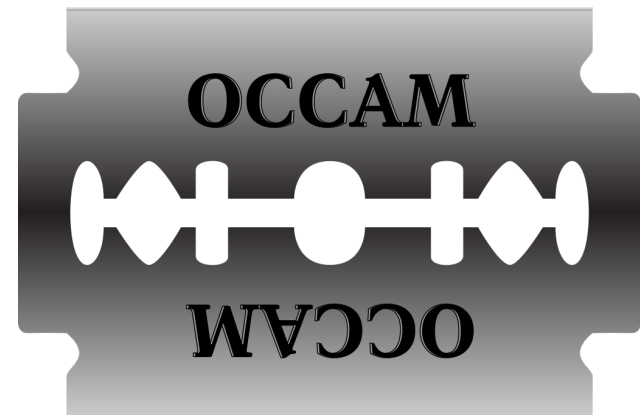
But only try if you have
sound logical argument
about why it makes
sense to have different
fits for different regions.

Otherwise, just another
way to overfit

- R^2 value for first part now .9581; for second part .6784; without break, have R^2 of .8815
- And we probably have a better estimate of k (old est. 21.53)

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
 - Validation
 - Simplicity



Occam's Razor

- *“Frustra fit per plura quod potest fieri per pauciora”*
 - *“It is futile to do with more things that which can be done with fewer”*
- Among competing hypotheses, the one with the fewest assumptions should be selected



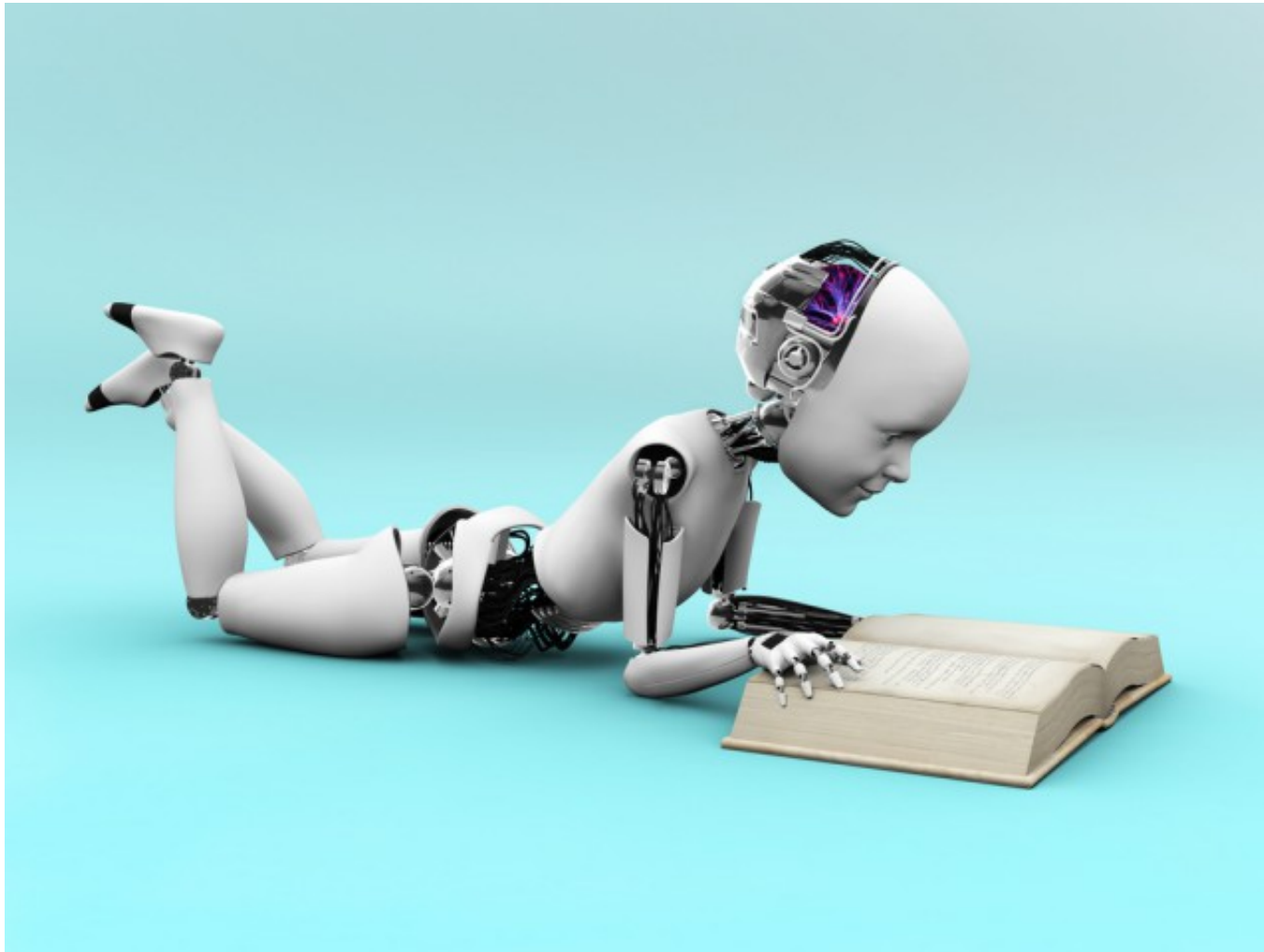
William of Occam
1287-1347



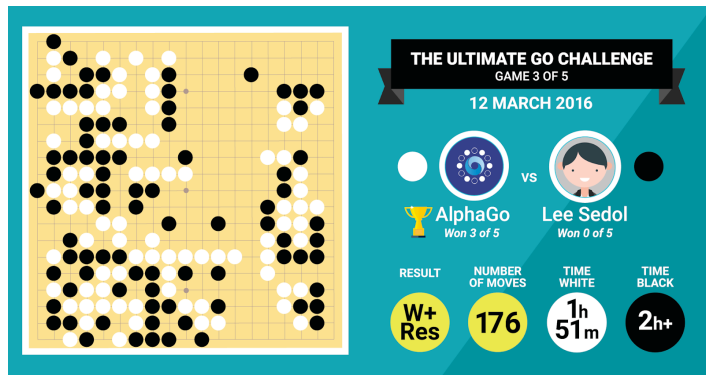
Switching gears



On to Machine Learning



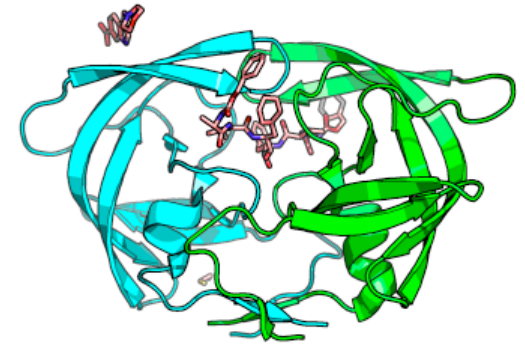
Machine Learning Is Everywhere



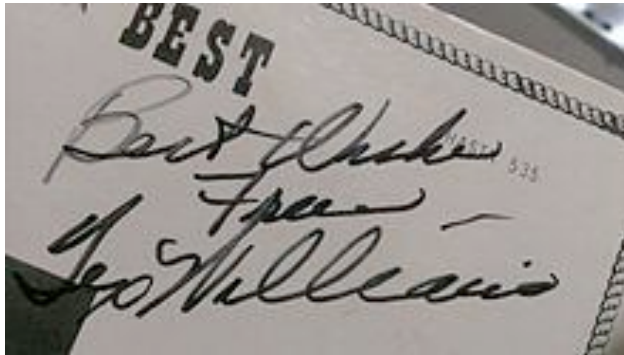
AlphaGo



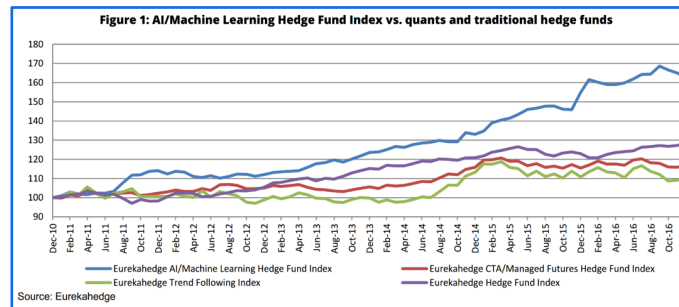
Recommendation systems



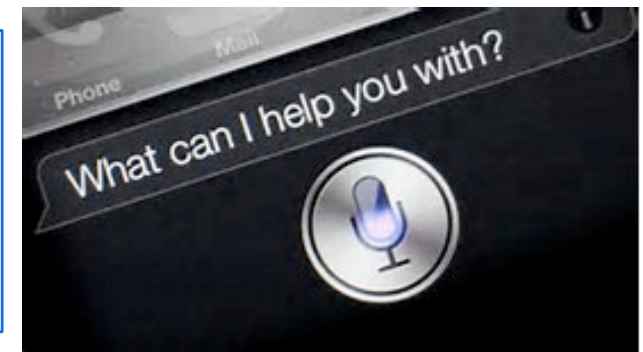
Drug discovery



Character recognition



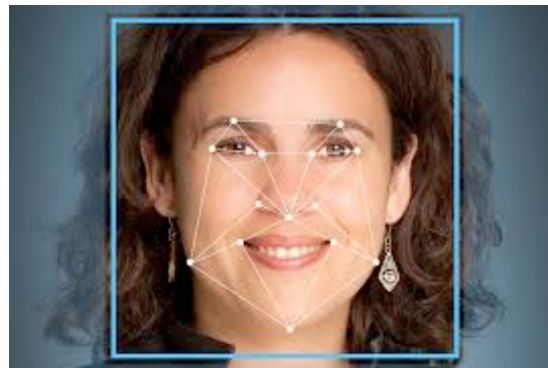
Hedge fund stock predictions



Voice assistants



Assisted driving



Face detection/recognition



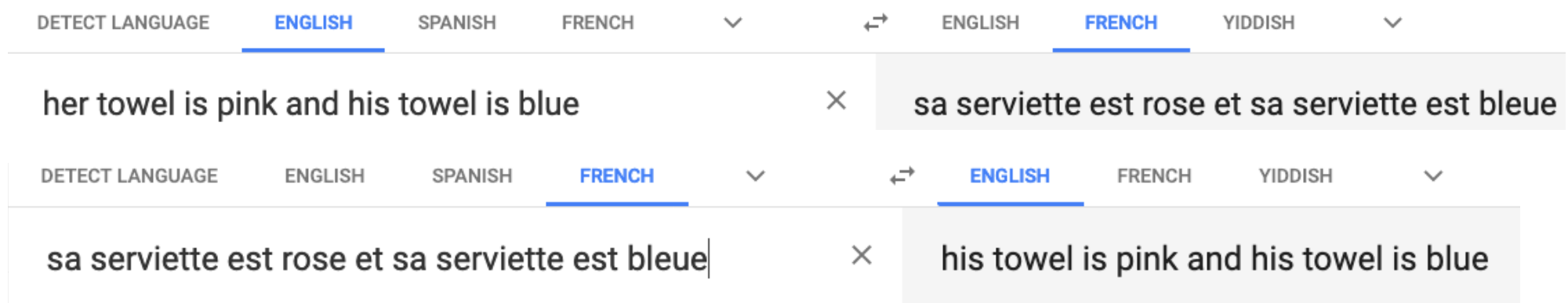
Cancer diagnosis

Success stories: Speech & Language

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

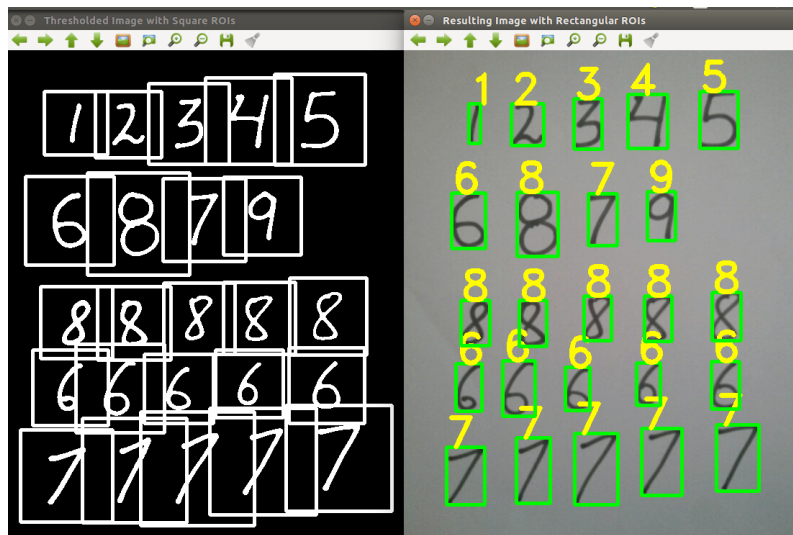


Translating from English to Russian back to English resulted in:
"The vodka is great but the meat is rotten"
What was the original English sentence?

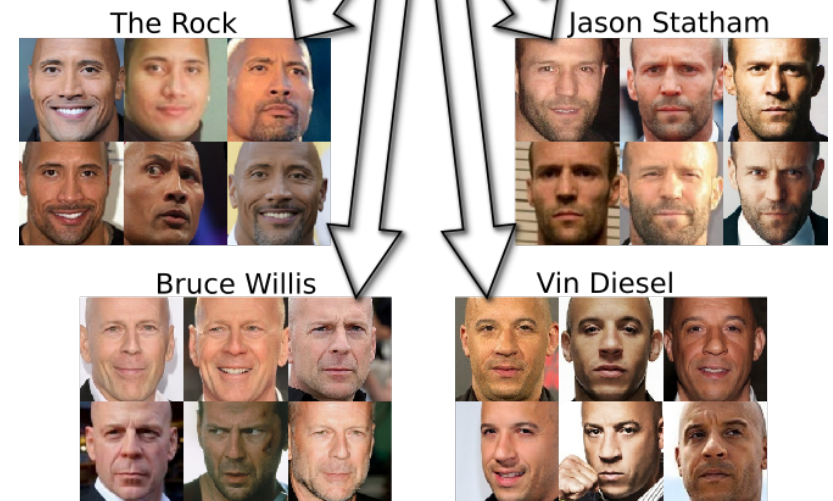


Success stories: Vision

- Face recognition
- Postal service uses handwriting recognition



Deep Learning Magic

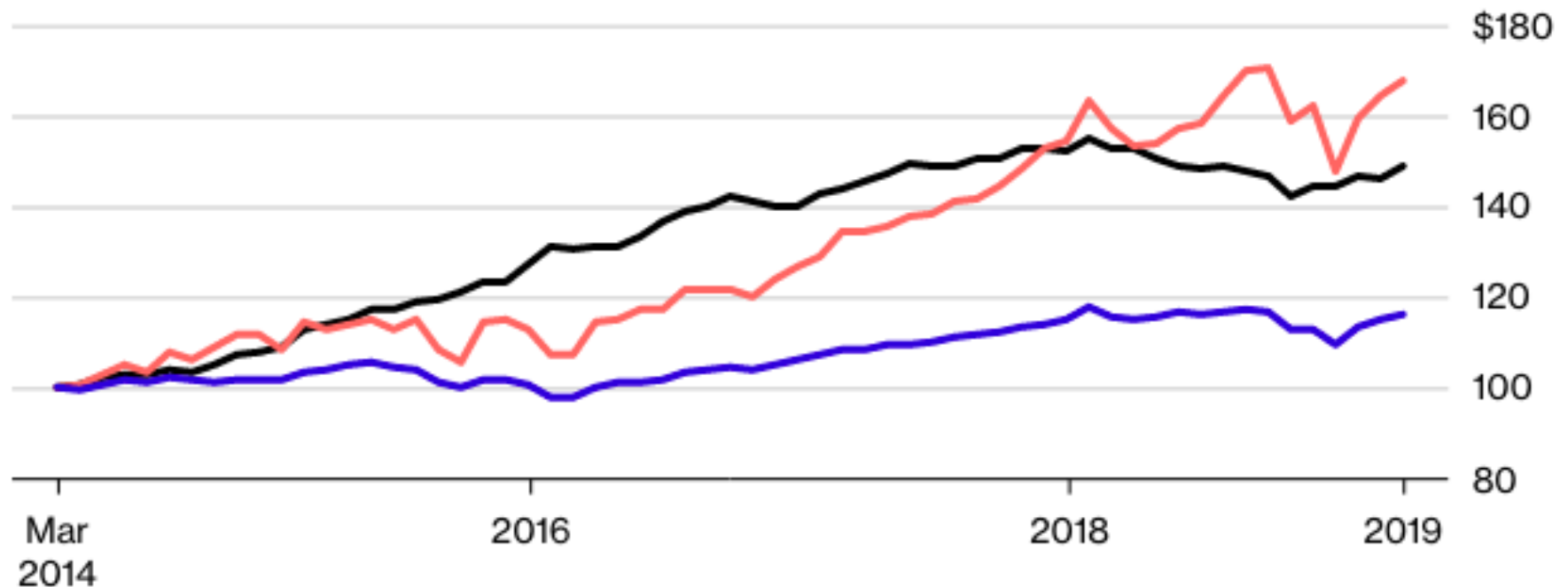


Success stories: Finance

Robot Investors

AI hedge fund managers are beating human peers, but not stock benchmarks

✍ Eureka hedge AI Index / S&P 500 / Hedge Fund index*



Source: Eureka hedge, 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

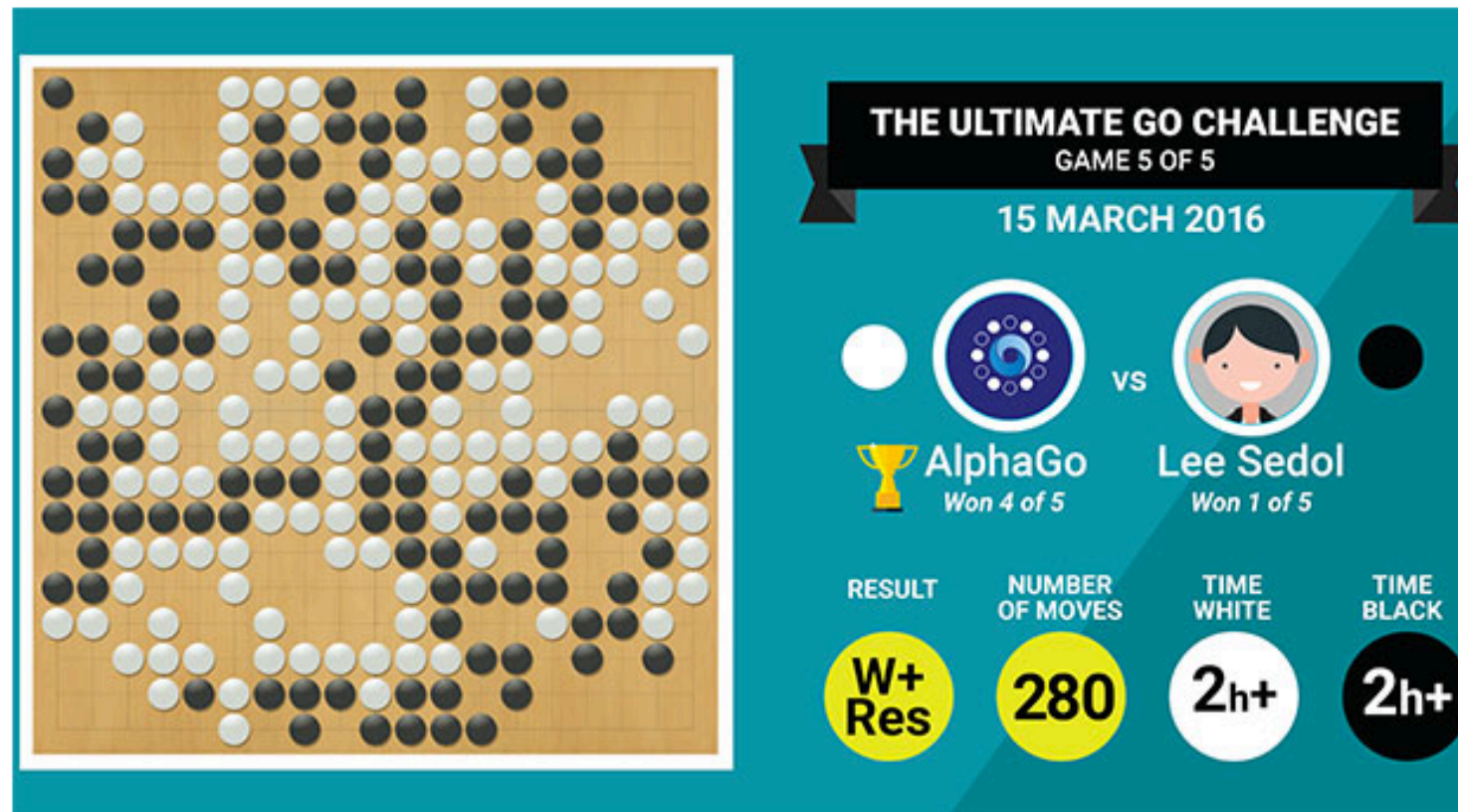


Image: Google

AlphaGo
Early 2016

AlphaGo learned to play Go by learning a model from training data selected by people (the training sets were the “programming”)

AlphaGo Zero learned to play Go with no human input, just by playing against itself, based on an objective function and a set of rules

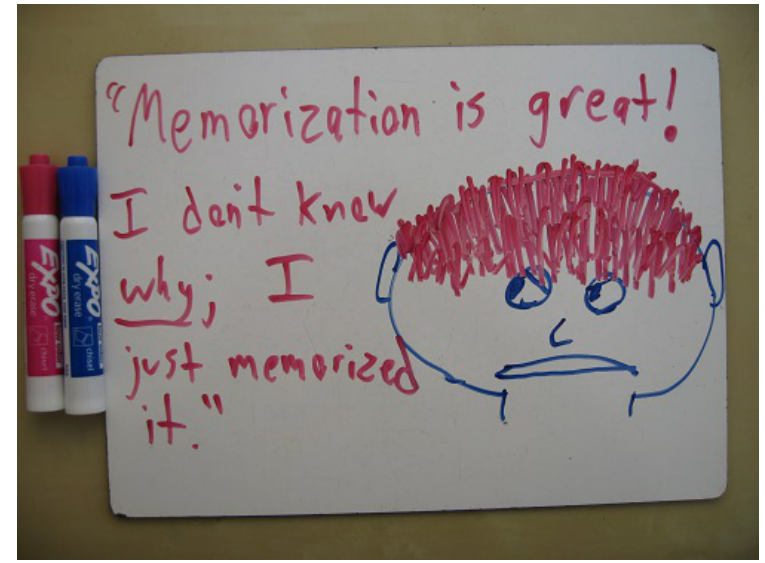
The plan ahead

- Machine learning is a huge topic
 - Courses covering machine learning include 6.008, 6.036, 6.860, 6.862, 6.867, 9.520, 9.54, 9.66
 - Topic is large component of other courses, e.g., in natural language processing, computational biology, computer vision, robotics, other areas
- In 6.0002, we will
 - Provide an introduction to the basic ideas:
 - Given a set of examples of inputs and (possibly) outcomes, want to learn model that describes the underlying process
 - How to measure similarity (distance) between examples?
 - How to group examples based on distance to create models?
 - Introduce classification methods (learning with outcomes), such as “k nearest neighbor” methods
 - Introduce clustering methods (learning without outcomes), such as “k-means”

How Are Things Learned?

■ Memorization

- Accumulation of individual facts
- Limited by
 - Time to observe facts
 - Memory to store facts
 - Can't deduce new information



■ Generalization

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

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



What Is Machine Learning?

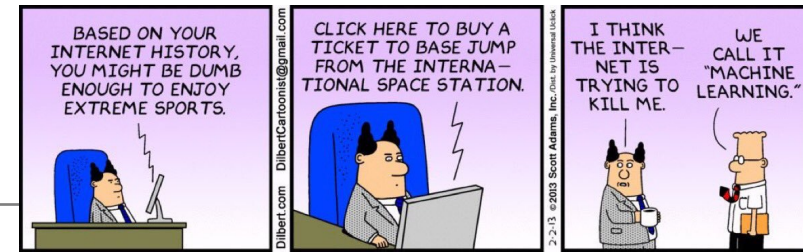
- All useful programs “learn” something
- In the first lecture of 6.0001 we looked at an algorithm for finding (learning?) square roots
- We recently looked at using linear regression to find (learn) a model of a collection of points associating system responses to input values
- We could argue that 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?

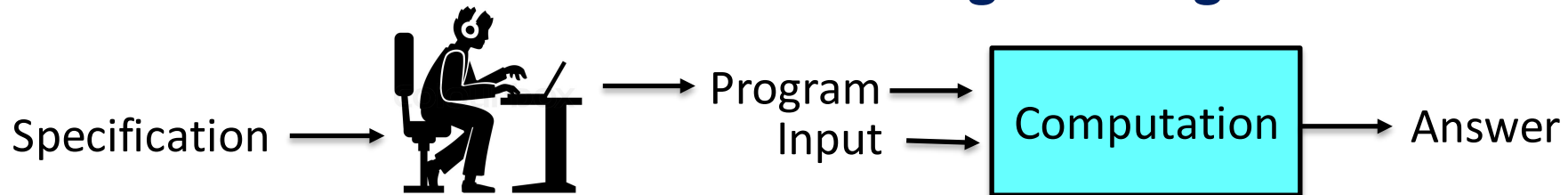


- Early definition of machine learning:
 - *"Field of study that gives computers the ability to learn without being explicitly programmed."* Arthur Samuel (1959)
 - Computer pioneer who wrote first self-learning program, which played checkers – learned from "experience"
 - Invented alpha-beta pruning – 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)

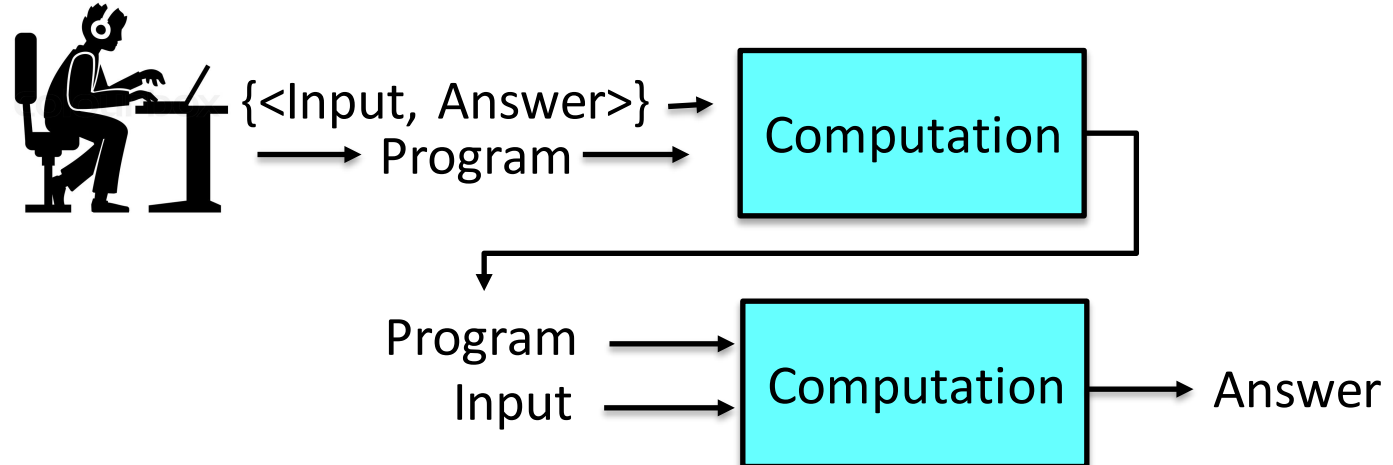
What Is Machine Learning



Traditional Programming

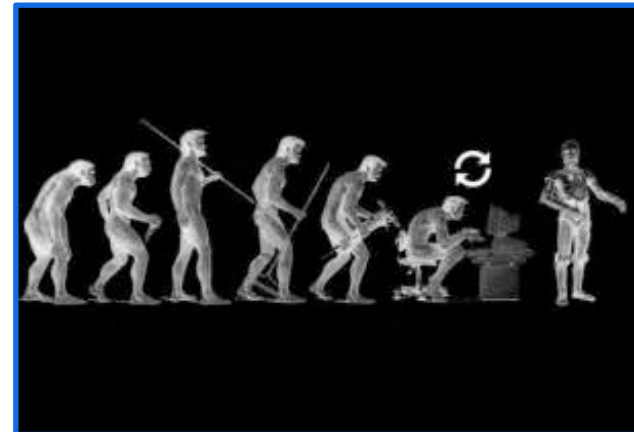


Supervised Machine Learning



Many approaches to Machine Learning

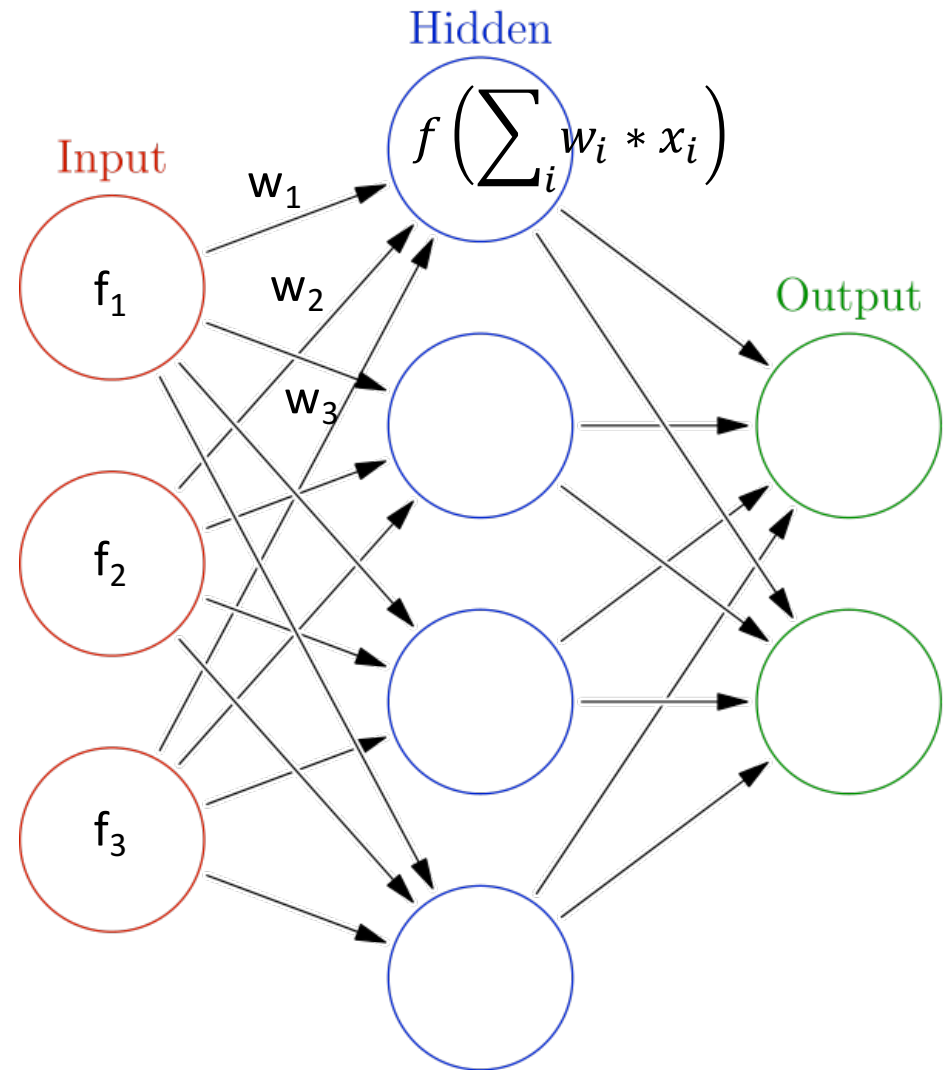
- Reinforcement learning
- Artificial neural networks (ANN's)
 - Convolutional neural nets (CNN's)
- Bayesian networks
- Support vector machines
- Genetic algorithms
- Clustering methods
 - K-means
- Classification methods
 - K nearest neighbors



- Want to show you a high level view of ANN's (current popular method)
- Then we will spend some time looking at simple clustering and classification methods

Artificial Neural Networks (ANNs)

- ANN is
 - Simplified model of the network of neurons in the brain
 - Set of interconnected nodes
 - Weights of connections can be inhibitory or reinforcing
- Weights are trained via the back-propagation algorithm, using a lot of training data
- Early ANNs typically only had one hidden layer, due to a lack of data and limited computation power to train more complex models
 - Modern ANNs have up to 1000 layers (SenseTime)



An aside on back propagation

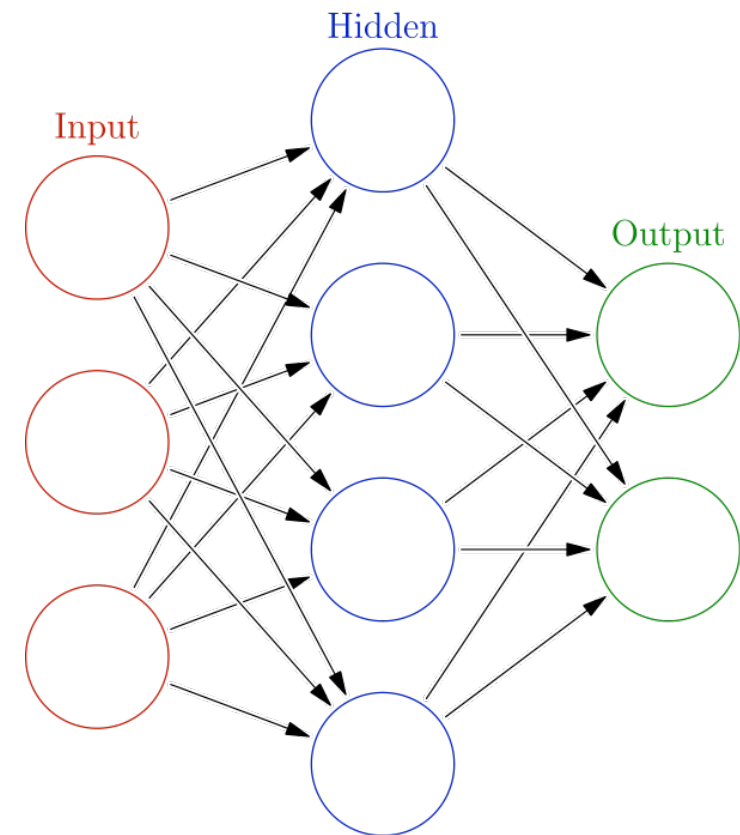
- For every node n , value is a function of inputs x and weights w - in simple case

$$n = f\left(\sum_i w_i * x_i\right)$$

- Given an input vector x and set of intermediate layers, system computes a set of outputs (e.g., each output node could be a label, and inputs are set of features associated with an example)
- Can measure error over a set of examples between computed output y' and actual labeled example y

You've seen this! $\sum_{\text{examples}} [y' - y]^2$

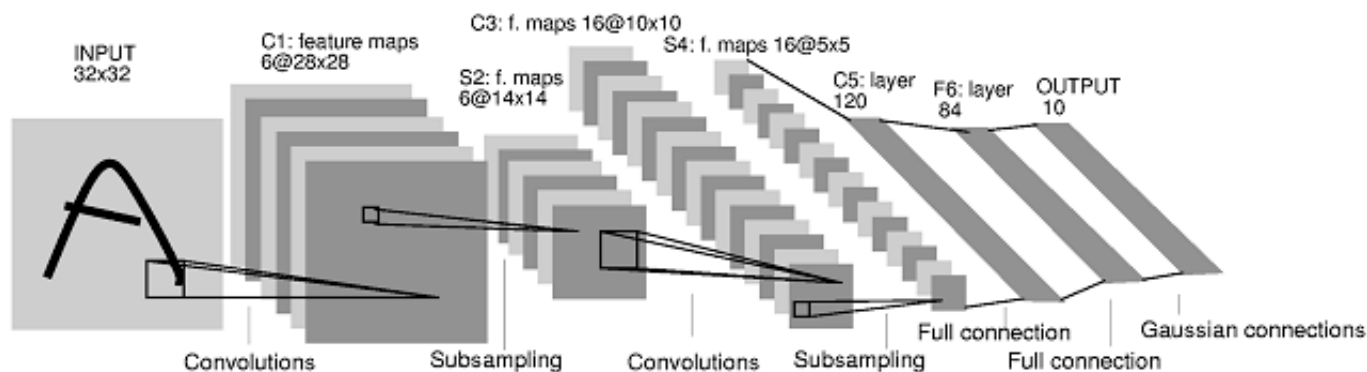
- Use gradient descent on the weights to compute new set of weights that minimize error
- Repeat with new training set of examples



You've seen this!

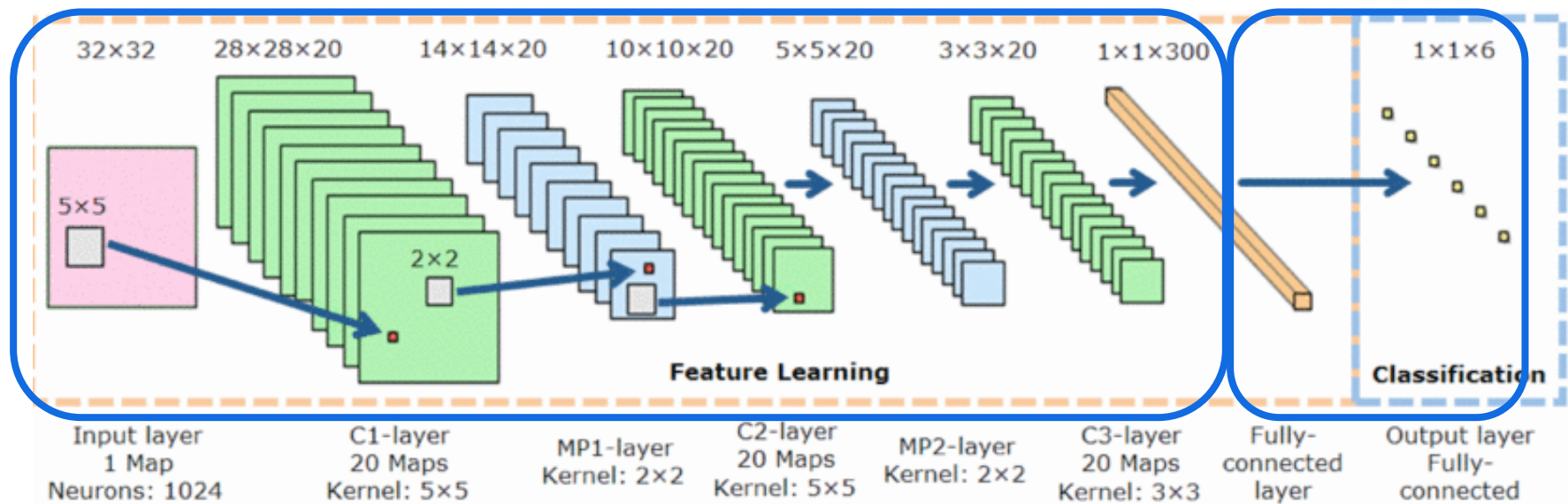
Deep Learning

- Deep Learning refers to complex neural networks
 - Many hidden layers characterizing different aspects of the observation
 - Many powerful supervised/unsupervised training algorithms
- Many kinds of deep learning networks – convolutional neural nets particularly popular
- Deep learning currently a very popular method in machine learning



Convolutional Neural Net: example

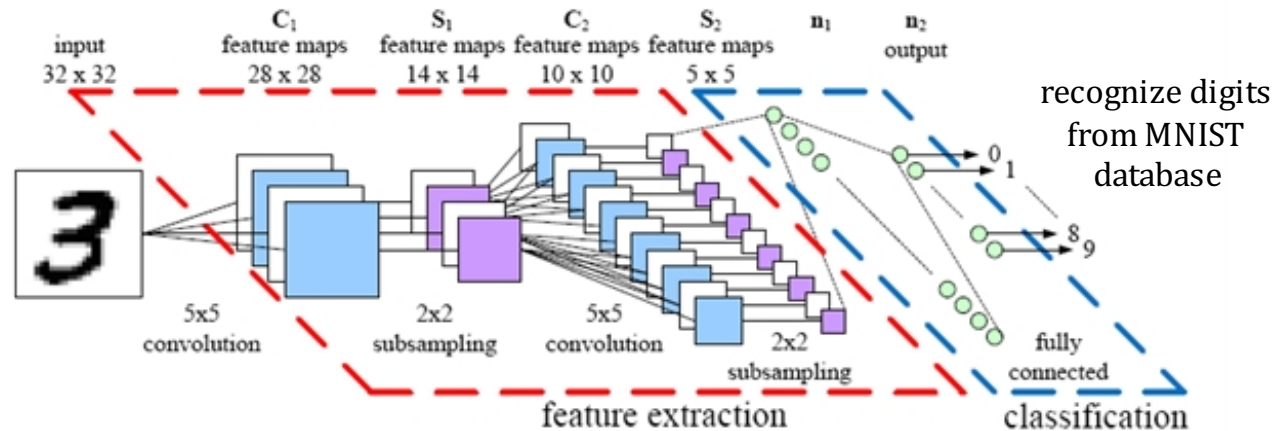
- Convolve image with set of filters (initially inspired by human system, today weights of filters learned as part of the training)
- Pool and down-sample outputs (e.g., max pooling)
- Repeat multiple times (until have single layer of values)
- Train weights for classifier (e.g., using logistical regression)



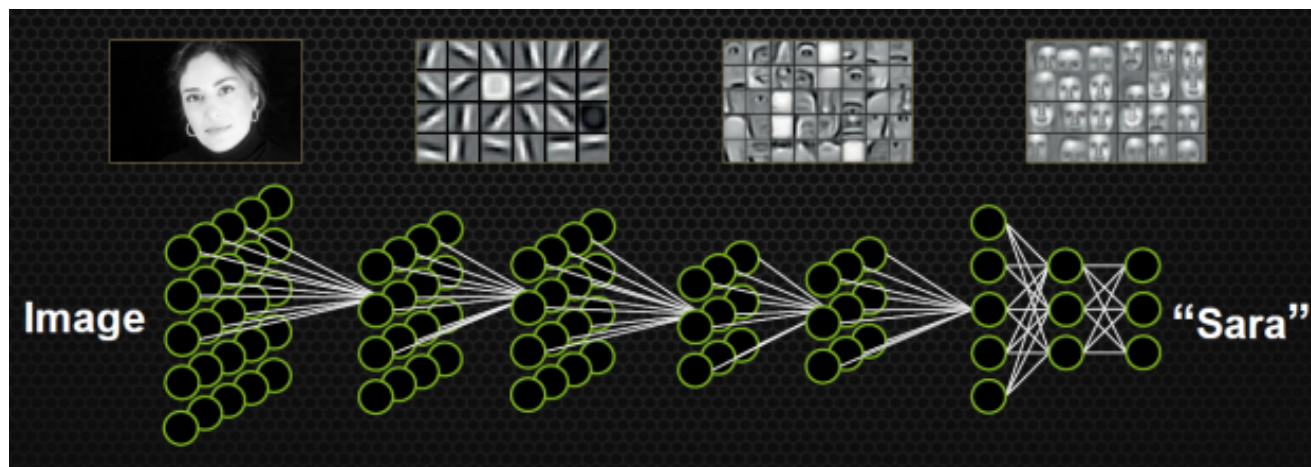
Extracting features into a vector

Classifier (next time)

Convolutional Neural Net: applications



LeCun, Bottou, Bengio, Haffner (1998)



*unsupervised learning of
hierarchical representations*

Lee, Grosse, Ranganath, Ng (2009)

Masks initially hand coded; now learned through training

SenseTime's Face Recognition

- Uses an ANN with over 1000 layers
- Face verification – 99.53% accuracy
- Face identification – 96.0% accuracy (with error of 0.001%)
 - Has been run against databases with 100M+ images
- Recognize faces with wide range of changes



Elva

Ramy

Coral

Basic ML 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**

Basic ML Paradigm

- Observe set of examples:
training data

Cat

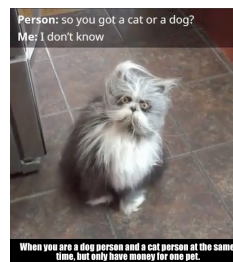
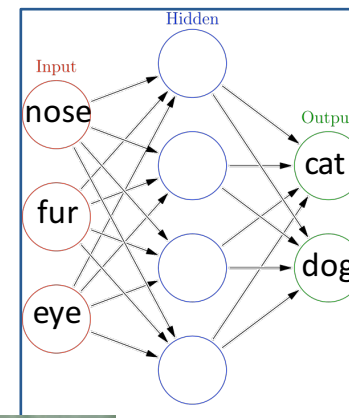


- Infer something about
process that generated
that data

Dog



- Use inference model to
make predictions about
previously unseen data:
test data



Are these cats or dogs?



Cat

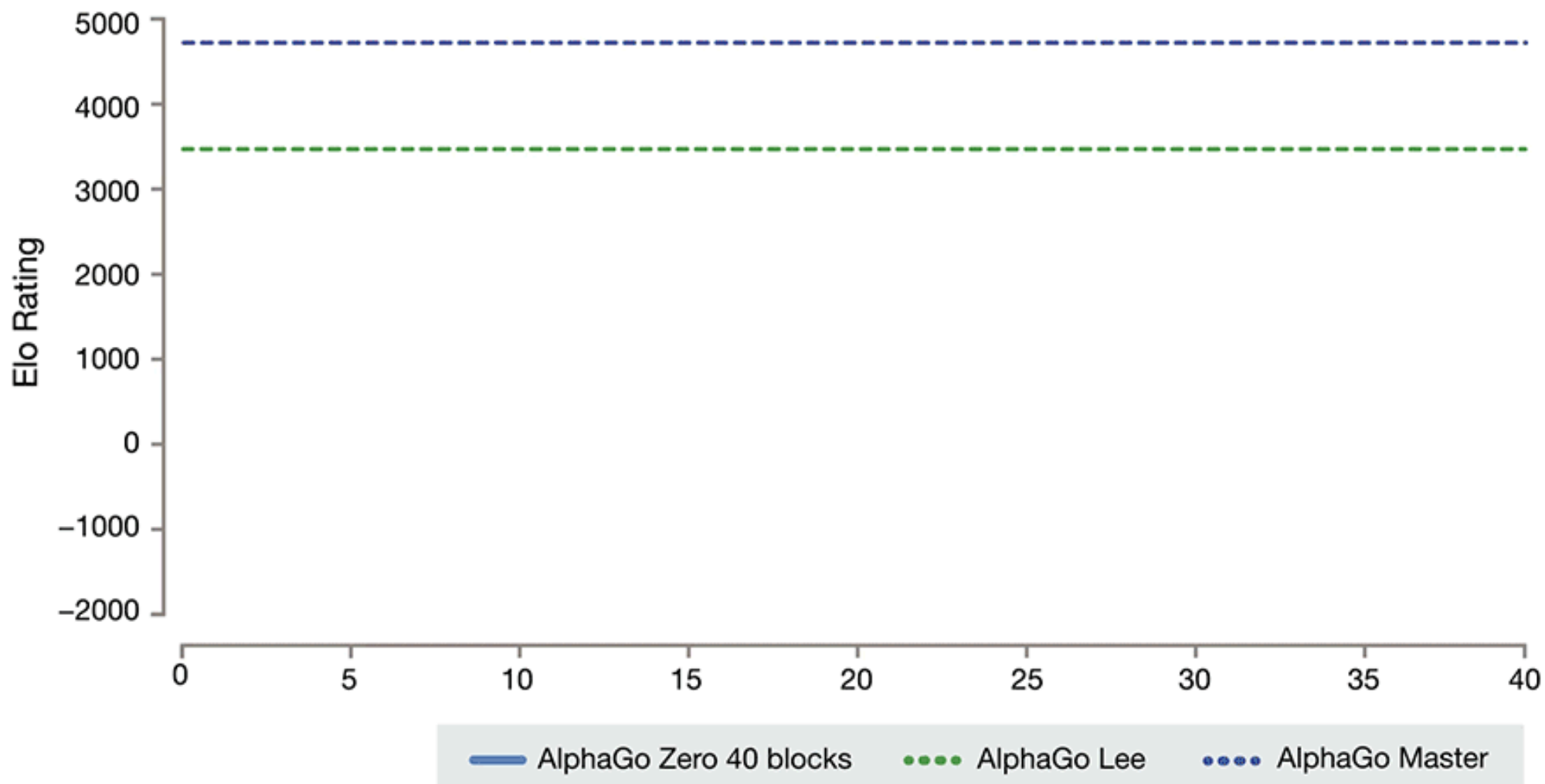


Dog

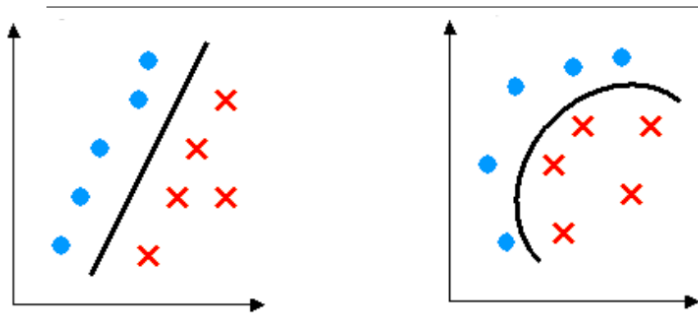
Returning to DeepMind's AlphaGo

- 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

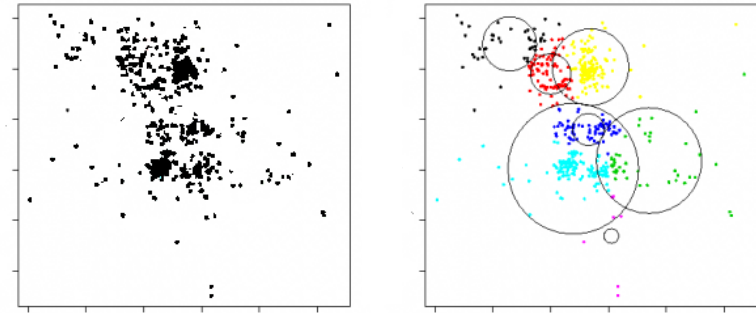


Learning Paradigms



Source: Utah CS

Supervised Learning

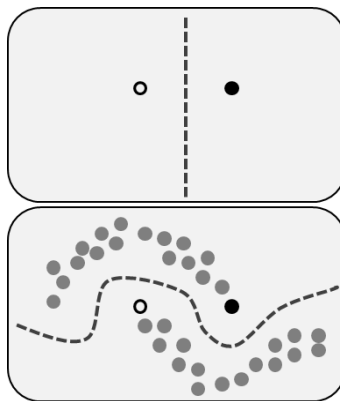


Raw Data

Source: Quora

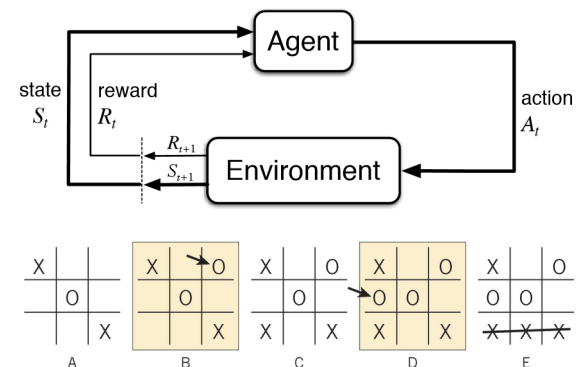
Clustered Data

Unsupervised Learning



Source: Wikipedia

Semi-Supervised Learning

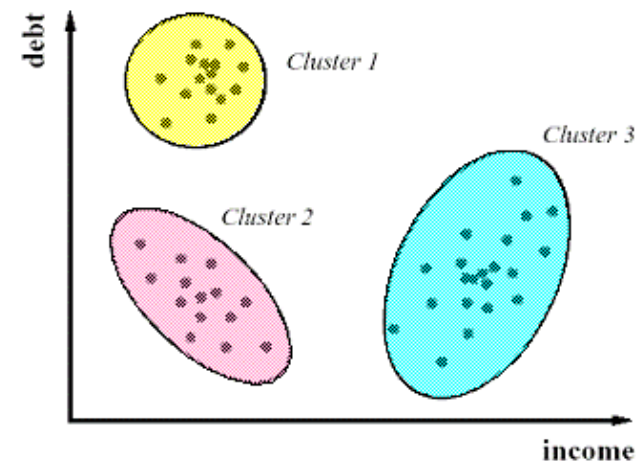
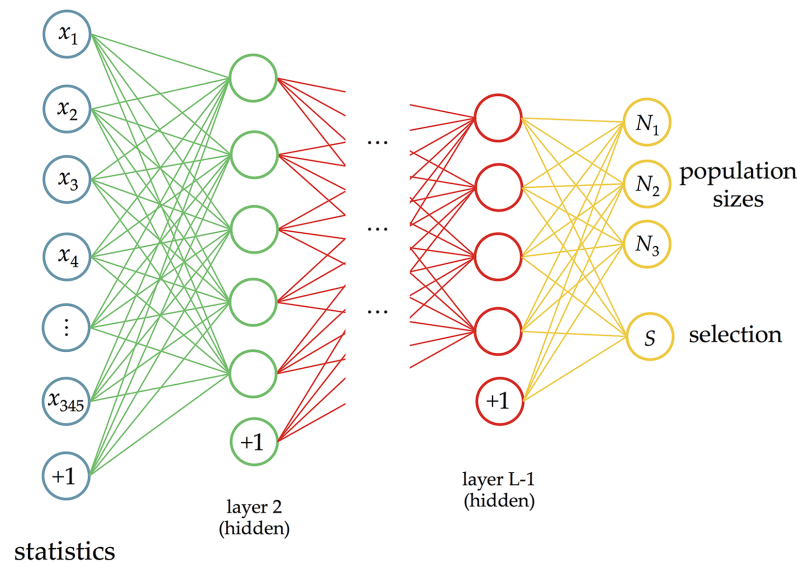


Source: Stanford CS and Nature

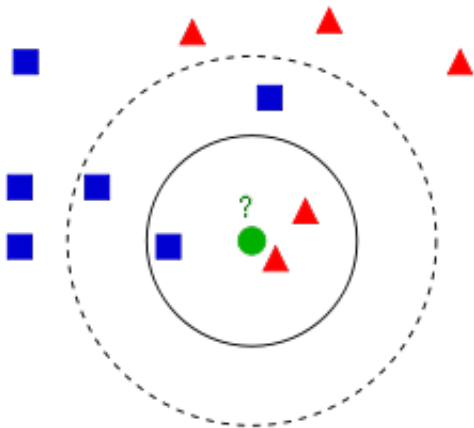
Reinforcement Learning

Two Broad Classes

- **Supervised:** given a set of feature/label pairs, find a rule that predicts the label associated with a previously unseen input
 - New examples labeled by applying learned process
- **Unsupervised:** given a set of feature vectors (without labels) group them into “natural clusters”
 - New examples labeled by “nearest” cluster

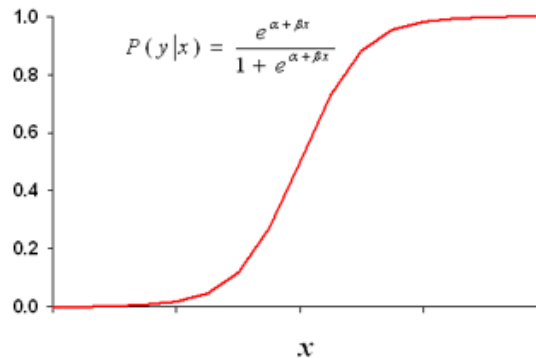


Some Algorithms



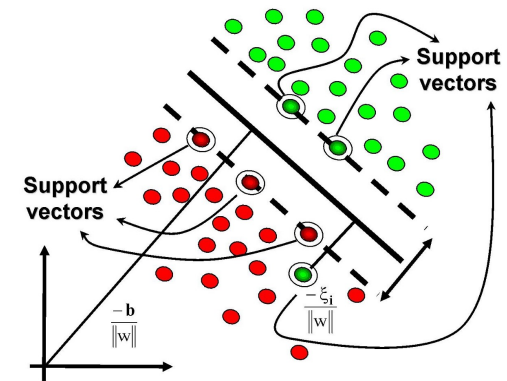
<https://onlinecourses.science.psu.edu/stat507/node/18>

KNN
1951 (Fix et al.)



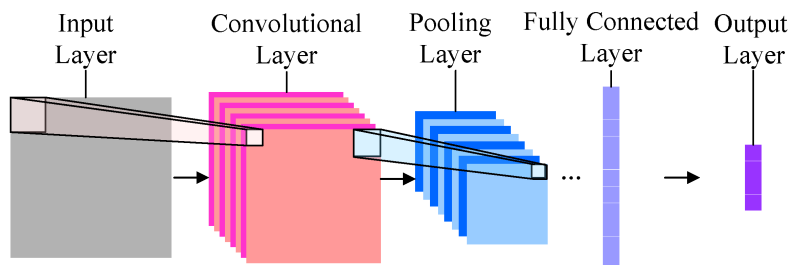
<https://onlinecourses.science.psu.edu/stat507/node/18>

Logistic Regression
1958 (Cox)



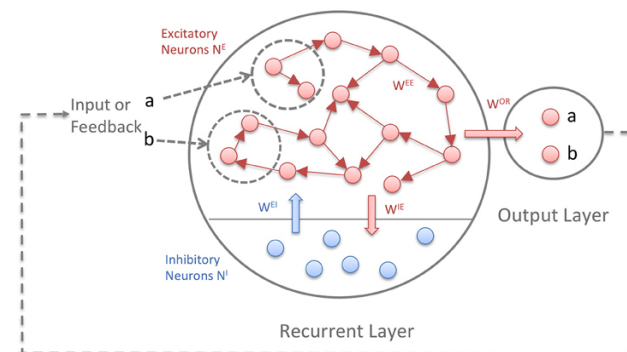
Source: https://medium.com/@haydar_ai/learning-data-science-day-11-support-vector-machines-8ef06da91bfc

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



Source: <http://www.mdpi.com>

Convolutional Neural Networks
1957/1986/1998/2006/2012



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

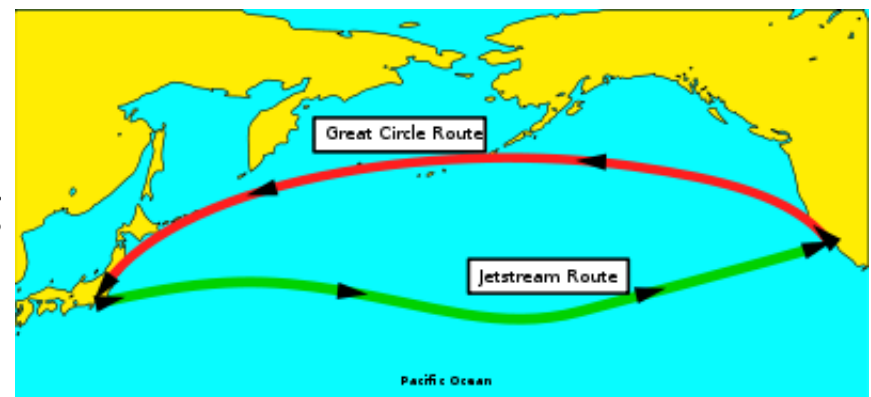
Recurrent Neural Networks

All ML Methods Require:

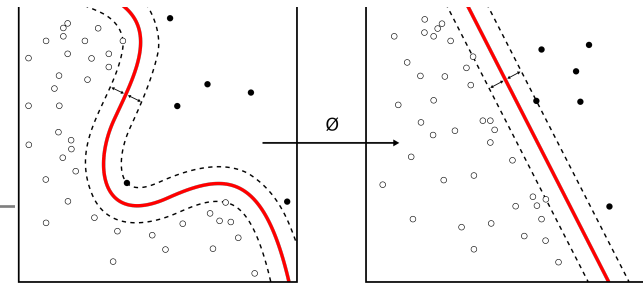
- Choosing training data and evaluation method
 - Representation of the features
 - Distance metric for feature vectors
 - Objective function and constraints
 - Optimization method for learning the model
- } Rest of Today
- } Next lecture

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?
 - Relative scales of axes
 - Distance metric

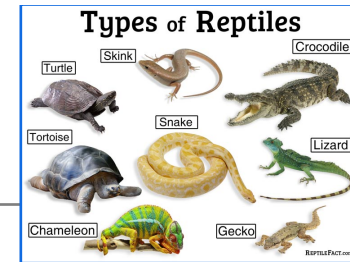


Feature Representation



- 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, who 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 **over fit**, e.g., birth month, eye color, first letter of last name
- Want to maximize ratio of useful input to irrelevant input in choice of features
 - **Signal-to-Noise Ratio (SNR)**
- How I choose to represent the range of each feature may make it easier or harder to separate different classes

An Example



Features

Label

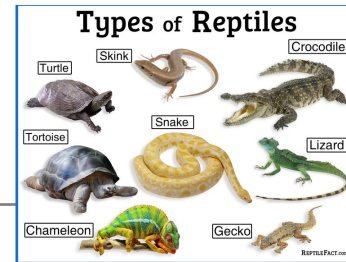
Name	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile
Cobra	True	True	True	True	0	Yes

Initial model:

- Everything is a reptile (no features needed)



An Example



Features

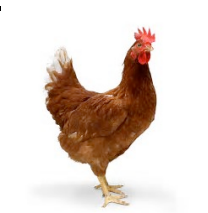
Label

Name	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile
Cobra	True	True	True	True	0	Yes
Chicken	True	True	False	False	2	No

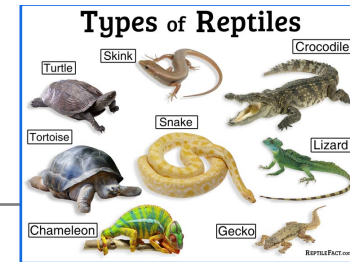
New model to use feature:

- Poisonous or # legs

Assume learning algorithm chooses
Poisonous
Looking for simplest model that explains data



An Example



Features

Label

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

Current model:

- Poisonous

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

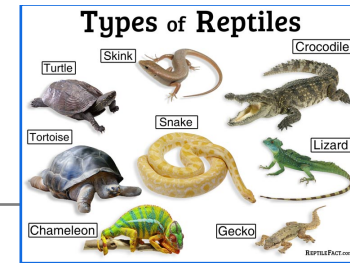
Need to refine model

New Model:

- # legs = 0



An Example



Features

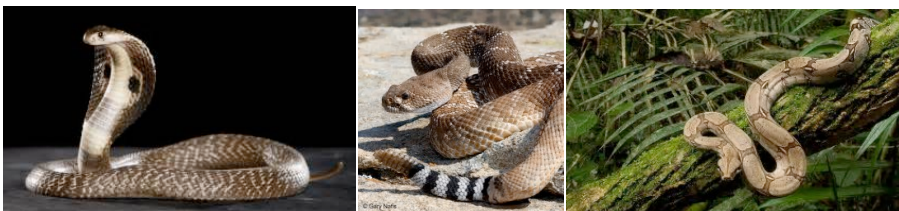
Label

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

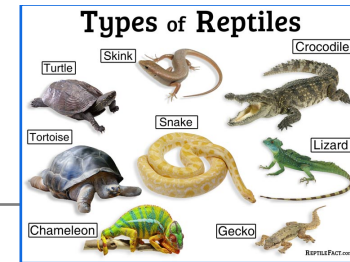
Current model:

- # legs = 0

Still okay



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
Alligator	True	True	False	True	4	Yes

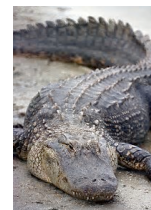
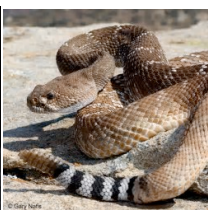
Current model:

- # legs = 0

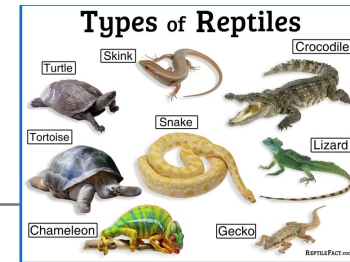
New model:

- Scales
- Cold blooded
- Legs != 2

Alligator doesn't fit model, but is a 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
Alligator	True	True	False	True	4	Yes
Dart frog	True	False	True	False	4	No

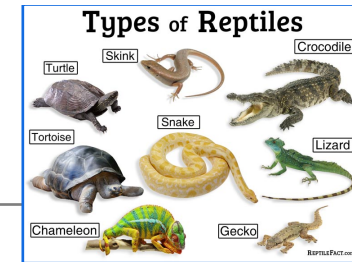
Current model:

- Has scales
- Cold blooded
- Legs != 2

Still okay



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
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

Time to find a way to generate classification rules

Five Minute Break



How do we learn to assign labels to examples?

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

Issues to consider in measuring distances

■ Feature engineering:

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

You've seen this! – variant of overfitting

You've seen this! – also a variant of overfitting

Measuring Distance Between Animals

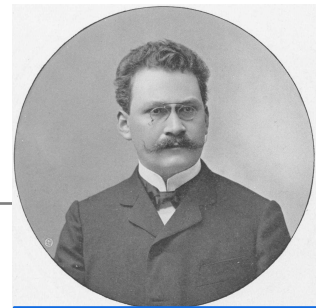
- We can think of our animal examples as consisting of four binary features and one integer feature – 5D space
- One way to learn to separate reptiles from non-reptiles is to measure the distance between pairs of examples, and use that:
 - To cluster nearby examples into a common class (unlabeled data), 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

Features = [egg, scales, poisonous, cold, # legs]

Minkowski Metric



Hermann
Minkowski

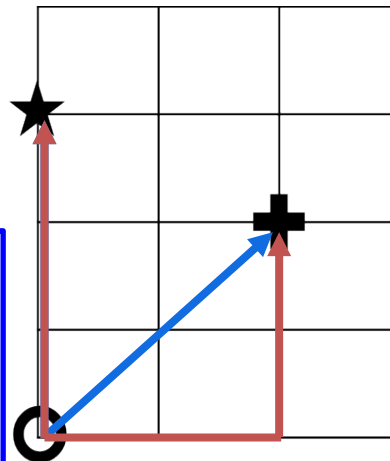
$$\text{dist}(X1, X2, p) = \left(\sum_{k=1}^{\text{len}} \text{abs}(X1_k - X2_k)^p \right)^{1/p}$$

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

Features = [egg, scales, poisonous, cold, # legs]

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

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

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



Euclidean Distance Between Animals

```
rattlesnake = [1,1,1,1,0]  
boa constrictor = [0,1,0,1,0]  
dartFrog = [1,0,1,0,4]
```

See plt.table in code

	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 either is to dart frog

Add an Alligator

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



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 (Binarization)

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]

Features = [egg, scales,
poisonous, cold, has legs]

	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

Binarization vs. Scaling

- Suppose we care about number of legs, not just whether animal has legs
- Scaling a more general solution
 - Scale each feature separately

```
def scaleFeature(vals):  
    vals = np.array(vals)  
    mean = sum(vals)/len(vals)  
    sd = np.std(vals)  
    vals = vals - mean  
    return vals/sd
```

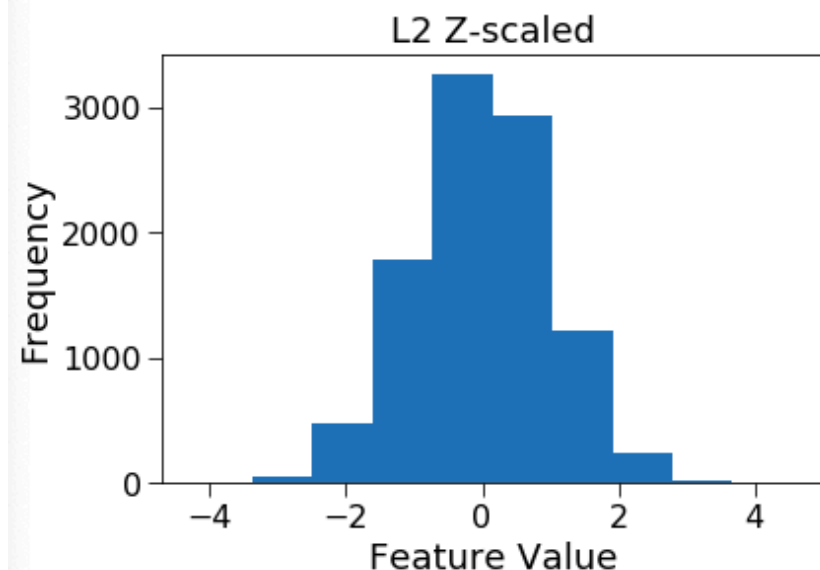
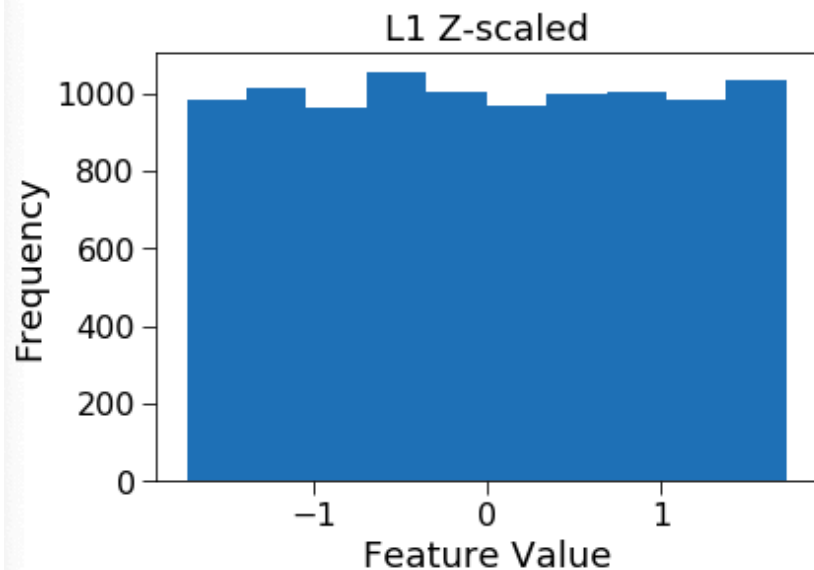
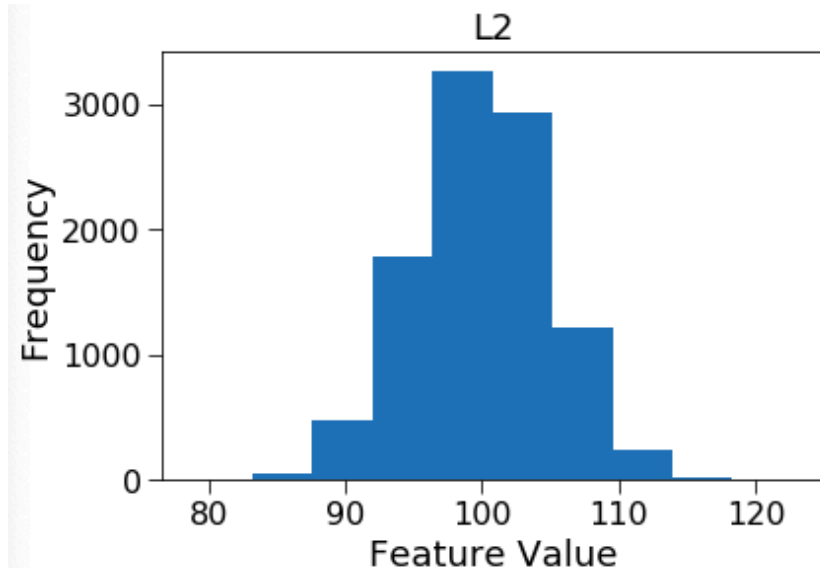
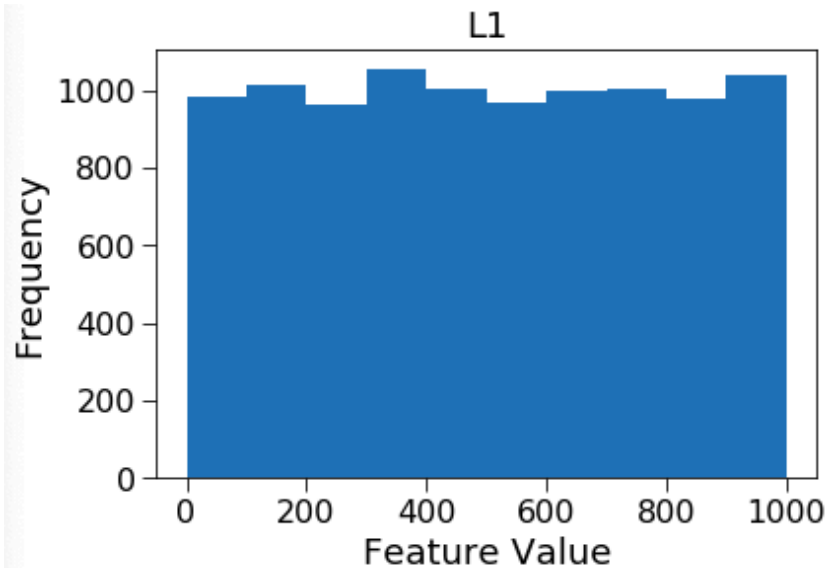
Z-Scaling

Mean = ?

Std = ?

Try applying z-scaling to uniform and Gaussian distributions

Scaling Changes Values But Not Shapes



Other Distance Metrics

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



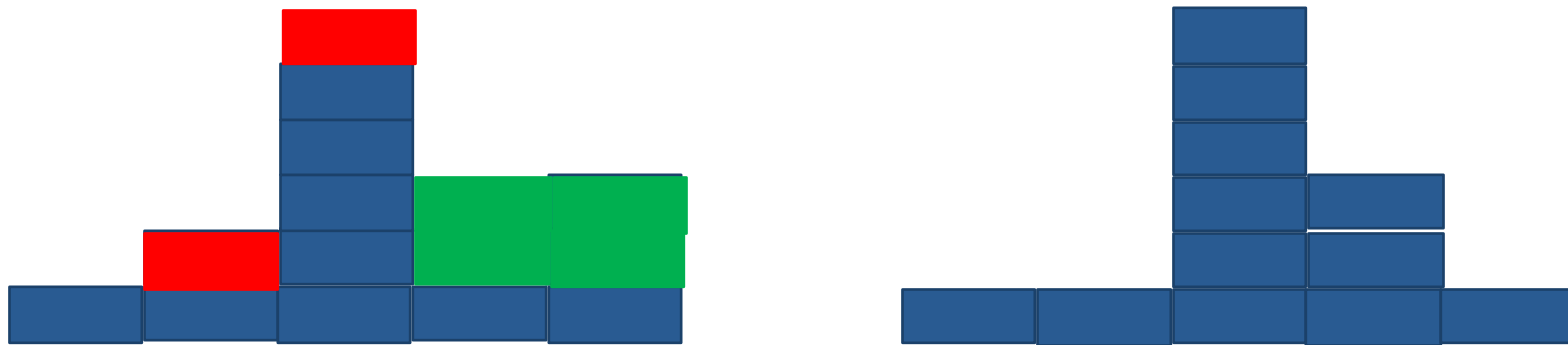
Earth Mover's Distance



- 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



Example: Earth Mover's distance



- Cost is 1 mass unit by 1 distance unit – 1
- Cost is 2 mass units by 1 distance unit – 2
- Total cost is 3 mass-distance units

Some Observations



- Ordering of bins on axis is important, since distance to move “dirt” depends on this
- Makes sense to use EMD when
 - Applying to probability distributions or to other histograms with an inherent order to bins
 - Intensities in an image
 - Colors in an image
 - Or applying to settings with inherent spatial ordering
 - Object movement in frames of a video sequence
 - Words in a text document

Coming Up

- In the next lecture, we will see examples of learning algorithms
 - When given unlabeled data, try to find clusters of examples near each other
 - When given labeled data, learn to classify examples

