A Quick Look at Machine Learning

- As you know, the amount of digital data is growing at an enormous rate. E.g.
  - 5 hours of video uploaded to Youtube every second
  - 6000 Tweets per second
  - 60000+ Google search queries per second
  - more than 300,000 carrier-based SMS text messages (US alone) per second (~2018)
  - More than 750,000 WhatsApp messages per second (2018)
- Everybody – companies, governments, researchers – wants to "mine" digital data for useful information
- Statistical machine learning is one of the approaches to tackle this "big data" problem. This is an area right now full of
  - hype
  - high-paying jobs for people with expertise

Statistical machine learning

- Term first defined long ago – 1959 by Arthur Samuel (perhaps best known publicly for checkers-playing programs): “Field of study that gives computers the ability to learn without being explicitly programmed”
- No precise universally-agreed-upon definition, but often generally used to mean:
  - Software that automatically learns to make useful inferences from implicit patterns in data
  - Usually, machine learning involves observing a set of examples that represent incomplete information about a phenomenon, and then attempting to infer something about the process that generated those examples
    - Practically, this might mean, for instance, that we could classify the examples into different groups according to some features and/or predict features/properties of new examples

Statistical machine learning

The examples that machine learning generally starts with are typically called "training data"

E.g. given partial descriptions of some people:

Abraham Lincoln: USA, President, 193 cm tall	A
George Washington: USA, President, 189 cm tall	A
Benjamin Harrison: USA, President, 168 cm tall	B
James Madison: USA, President, 163 cm tall	B
Louis Napoleon: France, President, 169 cm tall	B
Charles de Gaulle: France, President, 196 cm tall	A

E.g. and labels separating the people into two sets

A: {A. Lincoln, G. Washington, C. de Gaulle}

B: {B. Harrison, J. Madison, L. Napoleon}

One might reasonably infer ("learn") that the process assigning the people to the sets distinguishes tall from short presidents

The term **feature vector** is used for the partial descriptions of data items – each element of the vector (here three – country, position, height) describes some aspect (feature) of the item

Statistical machine learning

There are *many* approaches to/algorithms/ methods for machine learning – neural networks, support vector machines, decision trees, Bayesian networks, hidden Markov models, reinforcement learning, k-means clustering – but, generally, they all **try to learn a model that is a generalization of the provided examples** (the training data)

Supervised vs. unsupervised learning

Broadly speaking, algorithms can be classified as *supervised* or *unsupervised*

- Supervised learning:
  - Start with a set of feature vector and label pairs. The goal is to derive from these examples a rule that predicts the label associated with previously unseen feature vectors. Thus, in the A/B (tall/short presidents) example, given
    - "Thomas Jefferson: USA, President, 189 cm tall"

    the algorithm should predict set A as the proper label
  - Supervised learning is used for many tasks, including things like detecting fraudulent credit card use and recommending movies.

Supervised vs. unsupervised learning

Broadly speaking, algorithms can be classified as *supervised* or *unsupervised*

- Unsupervised learning:
  - Again start with a set of feature vectors. But no labels this time. The goal is to uncover/discover latent structure in the set of feature vectors. For example, given just the feature vectors from the presidents example, an unsupervised learning algorithm might separate the people into short vs. tall groups or American vs. French groups.
  - Many common unsupervised machine learning techniques are designed to find **clusters** of similar feature vectors – e.g., in genetics, groups of related genes.
  - We will learn one of the most common unsupervised basic clustering algorithms – k-means clustering.
    - Many other supervised and unsupervised techniques are substantially more complex - a day or two won't do ☺

Feature vectors

- When using machine learning, choosing “good” features (feature extraction) for your feature vectors is very important. The things you want to study often have many potential features but if you select too many “the noise” (features irrelevant to your problem) can distract from “the signal”
  - Irrelevant features can lead to a bad model, especially if the number of features is high relative to number of sample/examples
  - Irrelevant features can greatly slow the learning process. Machine learning algorithms are often quite computationally expensive, with complexity growing both with number of examples and number of features.
  - E.g. goal is to learn a model that will predict whether someone likes to drink wine. Some attributes likely relevant: age, nation where they live. Others maybe not: handedness, hair color …
    - But feature extraction is difficult, and often relies on intuition (which, of course, can be wrong …)

Feature vectors

Name	Egg-laying	Scales	Poisonous	Cold-blooded	# legs	Reptile
cobra	Yes	Yes	Yes	Yes	0	Yes
rattlesnake	Yes	Yes	Yes	Yes	0	Yes
boa	No	Yes	No	Yes	0	Yes
alligator	Yes	Yes	No	Yes	4	Yes
dart frog	Yes	No	Yes	No	4	No
salmon	Yes	Yes	No	Yes	0	No
python	Yes	Yes	No	Yes	0	Yes

If we start with just cobra and rattlesnake data, a supervised ML algorithm might "learn" that it should label as reptiles, things that are egg-laying, scaly, poisonous, cold-blooded and legless. But that would incorrectly label a boa constrictor as non-reptile.

With boa data added to the training set, the new inferred rule might become:

scales, cold-blooded, legless <--> reptile

But that fails for alligator … update rule again:

scales, cold-blooded, 0 or 4 legs <--> reptile

Dart frog – okay!

But then … salmon – not okay!

Can try to fix – trying to distinguish, e.g., salmon from alligator but is a losing battle in this case … reptile python has same features, using this feature set, as non-reptile salmon

This is, unfortunately, a common problem in machine learning

Distance metrics

- An important part of the clustering algorithm we'll study (and of many machine learning algorithms) is a **distance metric.** We want a way to evaluate the similarity between two items. E.g. "is a boa constrictor more similar to a rattlesnake or to a dart frog"
- First step in doing this is to convert feature vector into a list of numbers
  rattlesnake: Yes, Yes, Yes, Yes, 0 -> [1, 1, 1, 1, 0]

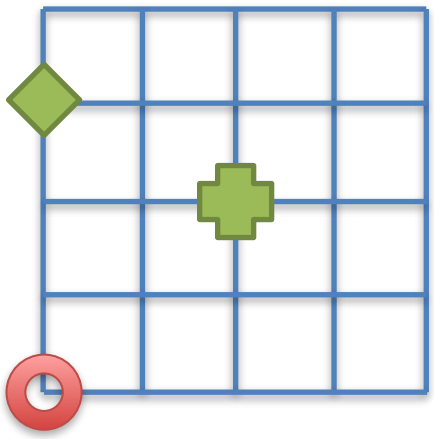
  boa: [0, 1, 0, 1, 0]

  dart frog: [1, 0, 1, 0, 4]
- Common metric: Minkowski distance
  - E.g Distance(V1, V2): square root of sum of squared feature differences.
    - In 2D, this is like normal Euclidean distance between 2 points (x1, y1), (x2, y2) on a plane. And like 3D distance between points in space.
  - But also "Manhattan distance": sum the absolute values of feature differences
    - This is like "walking distance" in a city. If you could fly from an intersection to an intersection that is east one block and north one block, the flying distance would be ~1.4 (square root of 2) blocks. But if you have to walk by road, the distance is 2 – one east, then one north (or one north followed by one east).

Minkowski distance

- Distance(V1, V2, p) = $(\sum_{i=1}^{len} abs(V1_i - V2_i)^p)^{1/p}$

- For p = 2, this is usual Euclidean distance
- For p = 1, “Manhattan” distance



The circle is closer to the diamond (distance 3 vs 4) using p=1, Manhattan distance.

Using p=2, the circle is closer to the cross (distance 2.8… vs 3).

Code: distancemetric.py

Distance metrics

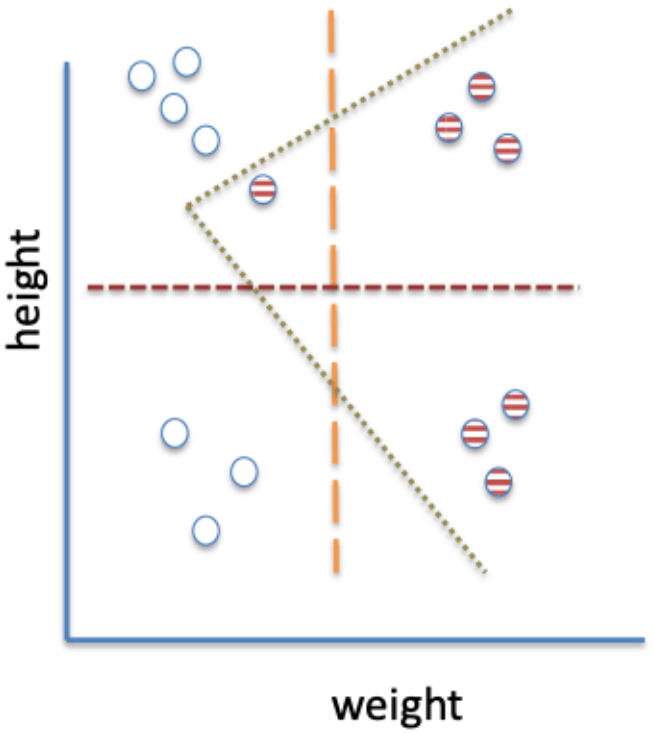
- Note that for the animals example the 0-4 range of number of legs gives extra (too much?) influence to the leg feature
  - Changing it to be a 0 (legless)/1(has legs) feature makes the table more reasonable

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

(1.414 – 2 features different, 1.732 – 3 features different, 2.236 – 5 different)

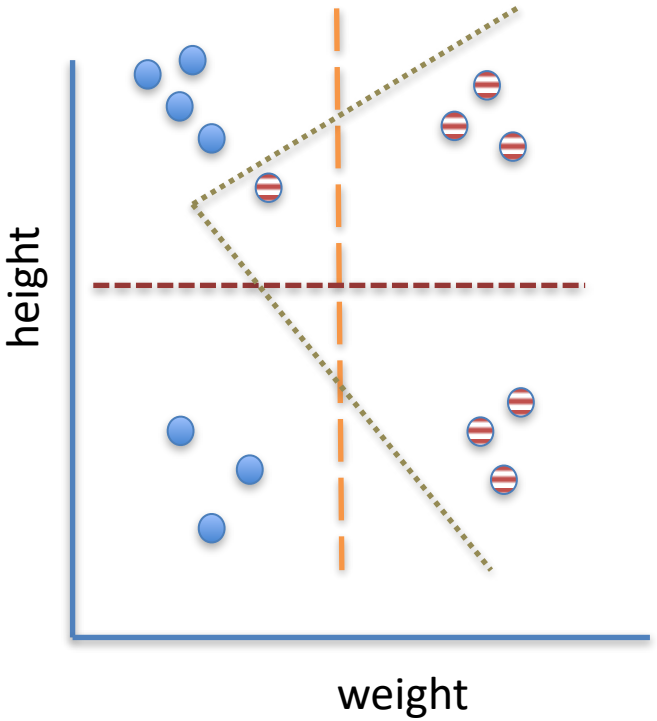
Clustering – the process of organizing objects into groups whose members are similar in some way

- cluster by height – horizontal line
- cluster by weight – vertical line
- cluster by whether or not wearing a striped shirt – two dotted lines (clusters can't always be separated with single straight line)



Clustering – an optimization problem

- General goal: find a set of clusters that optimizes an objective function subject to some constraints.
- That is, given a distance metric that can quantify how close to samples are, want to define objective function that:
  - Minimizes distances between examples in the same cluster, i.e. minimizes dissimilarity of examples within cluster



- This is, of course, vague. The precise details of the chosen objective function are critical to success of the clustering.

Clustering

How should we measure closeness of samples in a cluster? Cluster *variance*

- first compute the mean of the feature vectors of all samples in the cluster:

  sum(c)/float(len(c)) where c is a list of the feature vectors

- then define cluster "variance" (not quite the usual definition of variance) as:

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

Further, we define, for a set of clusters:

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

Clustering

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

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

- cluster "variance" gives us a sense of spread/tightness of items in one cluster
- dissimilarity gives us one number combining all the cluster variances

So, perhaps the optimization problem to be solved for clustering is simply to find a set of clusters that minimizes dissimilarity?? Not quite. Why?

Clustering

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

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

- Simply minimizing dissimilarity would be easy – one cluster for each item -> dissimilarity = 0
- Instead, add a constraint to be maintained while minimizing dissimilarity. Constrain, e.g., distance between clusters, or max number of clusters.
  - Next we'll look at one popular algorithm, k-means clustering. It minimizes dissimilarity subject to constraint of finding exactly k clusters

k-means clustering algorithm

K-means clustering is a popular basic method. Other common ones – e.g. hierarchical clustering – can be better in some situations.

Goal: partition set of samples into k clusters such that:

1. Each sample is in the cluster whose centroid is the closest centroid to that example
2. The dissimilarity of the set of clusters is minimized

Bad news: finding optimal solution computationally very expensive (NP-hard)

Good news: a greedy approach can be used to find a good approximation of the optimum

k-means clustering algorithm

Algorithm:

randomly choose k samples as initial cluster centroids

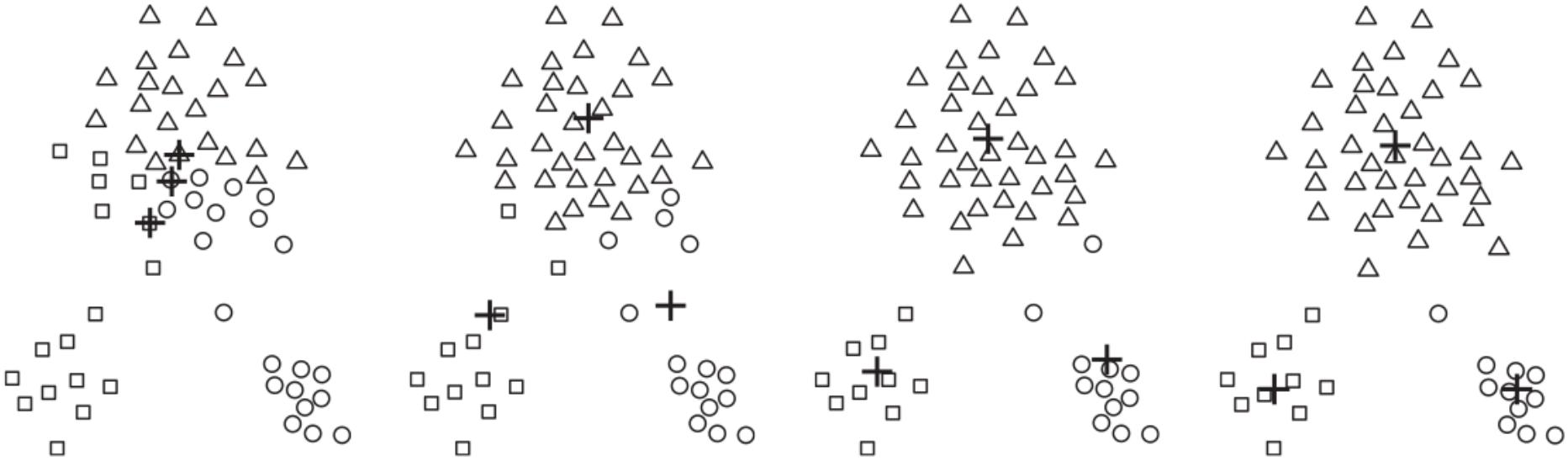
while True:

1) create k clusters by assigning each sample to closest centroid

2) compute k new centroids by averaging the samples in each cluster

3) if none of the centroids differs from the previous iteration, return the current set of clusters

k-means clustering algorithm



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

Source: Intro. to Data Mining, P-N Tan, M. Steinbach, V. Kumar. Addison-Wesley. 2006

K-means resource/demo sites

- Video demo: https://www.youtube.com/watch?v=zHbxbb2ye3E
- K-means clustering in Python: A Practical Guide: https://realpython.com/k-means-clustering-python/
- Interactive demo: https://user.ceng.metu.edu.tr/~akifakkus/courses/ceng574/k-means/

k-means clustering code

Algorithm:

randomly choose k samples as initial centroids

while not finished:

1) "fill" clusters by assigning each sample to cluster whose centroid it's closest to

2) re-compute cluster centroids by averaging the examples in each cluster

3) if none of the centroids differs from the previous iteration, we're finished

Code - kmeans.py

- contains some classes used to store
  - samples (called "examples" sometimes, so class Example)
  - clusters (class Cluster)
- kmeans(…) – the main k-means cluster finding algorithm
- trykmeans(…) – a function to execute several trials of kmeans, choosing best set of clusters from those returned by the trials
- contrivedTest() – simple example two clusters of 2D points
- contrivedTest2() – simple example with three clusters of points
- testTeeth() – animal teeth example, not particularly satisfying results
- Note: contains some commented out pylab code so you can run it in "plain" Python (you can easily reactivate code in plotSamples if you have pylab and want to see a plot of the data)

Italian olive oil example

- "Classification of olive oils from their fatty acid composition" Forina, M., Armanino, C., Lanteri, S., and Tiscornia, E. *Food Research and Data Analysis, 1983*
- *572 olive oils from three "areas"/nine "regions"*
- *K-means can do a good job separating them by region*