RECAP
k-means: A clustering algorithm...
**k-means algorithm**

0. **Initialize cluster centers** 
   \[ \mu_1, \mu_2, \ldots, \mu_k \]

1. **Assign observations to closest cluster center**
   \[ z_i \leftarrow \text{arg min}_j \| \mu_j - x_i \|_2^2 \]

   Inferred label for obs \( i \), whereas supervised learning has given label \( y_i \)

2. **Revise cluster centers as mean of assigned observations**
   \[ \mu_j \leftarrow \frac{1}{\sum_{i \in C_j} 1} \sum_{i \in C_j} x_i \]

3. **Repeat 1.+2. until convergence**

\( k = 3 \) clusters
k-means algorithm

0. Initialize cluster centers
1. Assign observations to closest cluster center
2. Revise cluster centers as mean of assigned observations

\[ \mu_j = \frac{1}{n_j} \sum_{i: z_i = j} x_i \]
Convergence of k-means

Converges to:

- **Global optimum**
- **Local optimum**
- **neither**

---

**k-means++ pros/cons**

Computationally costly relative to random initialization, but the subsequent k-means often converges more rapidly

Tends to **improve quality of local optimum** and **lower runtime**
Cluster heterogeneity

Measure of quality of given clustering:
\[
\sum_{j=1}^{k} \sum_{i: z_i = j} ||\mu_j - x_i||^2
\]

Lower is better!

How to choose k?

Lowest possible cluster heterogeneity

# of clusters k
Limitations & failure modes of k-means

Learn user preferences
Set of clustered documents read by user

Cluster 1
Cluster 2
Cluster 3
Cluster 4

Use feedback to learn user preferences over topics
Uncertainty in cluster assignments

Cluster 1

Cluster 3

Cluster 4

Slightly closer to Cluster 4 than Cluster 2, but count fully for Cluster 4?

Uncertainty in cluster assignments

Cluster 1

Cluster 3

Cluster 4

Hard assignments don’t tell full story
Other limitations of k-means

Assign observations to closest cluster center

\[ z_i \leftarrow \arg \min_j ||\mu_j - x_i||_2 \]

- Can use weighted Euclidean, but requires known weights
- Only center matters
- Equivalent to assuming spherically symmetric clusters
- Still assumes all clusters have the same axis-aligned ellipses

Failure modes of k-means

- Disparate cluster sizes
- Overlapping clusters
- Different shaped/oriented clusters
Motivates probabilistic model: Mixture model

- Provides **soft assignments** of observations to clusters (uncertainty in assignment)
  - e.g., 54% chance document is **world news**, 45% **science**, 1% **sports**, and 0% **entertainment**

- Accounts for cluster **shapes** not just **centers**

- Enables **learning weightings** of dimensions
  - e.g., how much to weight each word in the vocabulary when computing cluster assignment

**Summary for k-means**
What you can do now...

- Describe the input (unlabeled observations) and output (labels) of a clustering algorithm
- Determine whether a task is supervised or unsupervised
- Cluster documents using k-means
- Describe potential applications of clustering
Hierarchical clustering

Why hierarchical clustering?
Lots of data are hierarchical in nature

Product categories

...
Lots of data are hierarchical in nature

Other motivations for hierarchical clustering

- Avoid choosing # clusters beforehand

- **Dendrograms** help visualize different clustering granularities
  - No need to rerun algorithm

- Most algorithms allow user to choose any distance metric
  - k-means restricted us to Euclidean distance
Other motivations for hierarchical clustering

Can often find more complex shapes than k-means or Gaussian mixture models

What about these?
Recall challenging clusters to discover...

Two main types of algorithms

**Divisive**, *a.k.a.* top-down: Start with all data in one big cluster and recursively split.
- Example: *recursive k-means*

**Agglomerative** *a.k.a.* bottom-up: Start with each data point as its own cluster. Merge clusters until all points are in one big cluster.
- Example: *single linkage*
Divisive clustering

**Divisive** in pictures – level 1
Divisive in pictures – level 2

Divisive: Recursive k-mean

Wikipedia
**Divisive: Recursive k-means**

- **Wikipedia**
  - Athletes
  - Non-athletes

- **Baseball**
- **Soccer/Ice hockey**
- **Musicians, artists, actors**
- **Scholars, politicians, government officials**
Divisive choices to be made

- Which algorithm to recurse
- How many clusters per split
- When to split vs. stop
  - Max cluster size: number of points in cluster falls below threshold
  - Max cluster radius: distance to furthest point falls below threshold
  - Specified # clusters: split until pre-specified # clusters is reached

Agglomerative clustering
Agglomerative: Single linkage

1. Initialize each point to be its own cluster
Agglomerative: Single linkage

2. Define distance between clusters to be:

\[
\text{distance}(C_1, C_2) = \min_{x_i \in C_1, x_j \in C_2} d(x_i, x_j)
\]

Agglomerative: Single linkage

3. Merge the two closest clusters
Agglomerative: Single linkage

4. Repeat step 3 until all points are in one cluster
**Agglomerative: Single linkage**

4. Repeat step 3 until all points are in one cluster
Agglomerative: Single linkage

4. Repeat step 3 until all points are in one cluster

Clusters of clusters

Just like our picture for divisive clustering...
The dendrogram for agglomerative clustering

• x axis shows data points (carefully ordered)
• y-axis shows distance between pair of clusters
The dendrogram

- x axis shows data points (carefully ordered)
- y-axis shows distance between pair of clusters

Height here indicates **min distance** between blue pts and green pts (2 clusters)

The dendrogram

Path shows all clusters to which a point belongs and the order in which clusters merge
Extracting a partition

Choose a distance $D^*$ at which to cut dendogram

Every branch that crosses $D^*$ becomes a separate cluster
Agglomerative clustering details

**Agglomerative choices to be made**

- Distance metric: \(d(x_i, x_j)\)
- Linkage function: e.g., \(\min_{x_i \in C_1, x_j \in C_2} d(x_i, x_j)\)
- Where and how to cut dendrogram
More on cutting dendrogram

- For visualization, smaller # clusters is preferable
- For tasks like outlier detection, cut based on:
  - Distance threshold
  - Inconsistency coefficient
    - Compare height of merge to average merge heights below
    - If top merge is substantially higher, then it is joining two subsets that are relatively far apart compared to the members of each subset internally
    - Still have to choose a threshold to cut at, but now in terms of "inconsistency" rather than distance
- No cutting method is "incorrect", some are just more useful than others

Computational considerations

- Computing all pairs of distances is expensive
  - Brute force algorithm is $O(N^2 \log(N))$
  - Smart implementations use triangle inequality to rule out candidate pairs
- Best known algorithm is $O(N^2)$
Statistical issues

Chaining: Distant points clustered together if there is a chain of pairwise close points between

Other linkage functions can be more robust, but restrict the shapes of clusters that can be found
- Complete linkage: max pairwise distance between clusters
- Ward criterion: min increase in within-cluster variance at each merge

Summary for hierarchical clustering
What you can do now...

• Motivate why you might want to perform hierarchical clustering
• Compare and contrast divisive and agglomerative clustering approaches
• Describe what the nodes and branches (and branch heights) represent in a dendogram
• Build a dendogram using a variety of linkage functions
  – Single linkage
  – Complete linkage
  – Wald’s criterion
• Cut a dendogram to return a specific clustering
  – List options for how to cut the dendogram
• Discuss computational challenges in performing hierarchical clustering

Dimensionality reduction

STAT/CSE 416: Intro to Machine Learning
Emily Fox
University of Washington
May 15, 2018
Embedding

Example: Embedding images to visualize data

Images with thousands or millions of pixels

Can we give each image a coordinate, such that similar images are near each other?

Embedding words

[Joseph Turian]
Dimensionality reduction

- Input data may have thousands or millions of dimensions!
  - e.g., text data has
- **Dimensionality reduction**: represent data with fewer dimensions
  - **easier learning** – fewer parameters
  - **visualization** – hard to visualize more than 3D or 4D
  - discover “intrinsic dimensionality” of data
    - high dimensional data that is truly lower dimensional
Lower dimensional projections

- Rather than picking a subset of the features, we can create new features that are combinations of existing features.

- Let’s see this in the unsupervised setting:
  - just x, but no y

Linear projection...

- Project onto 1-dimension
Linear projection and reconstruction

Reconstruction: Only knowing z, what was \((x[1], x[2])\)?

What if we project onto \(d\) orthogonal vectors?

Perfect reconstruction!
If I had to choose one of these vectors, which do I prefer?

What about overall single vectors?

Consider extreme data example:
Principal component analysis (PCA) – Basic idea

- Project d-dimensional data into k-dimensional space while preserving as much information as possible:
  - e.g., project space of 10000 words into 3-dimensions
  - e.g., project 3-d into 2-d

- Choose projection with minimum reconstruction error

“PCA explained visually”

http://setosa.io/ev/principal-component-analysis/
Basic PCA algorithm

- Form data matrix $X$
  - Each row is a different data point...like our typical data tables

- Recenter: subtract the mean from each row of $X \rightarrow X_c$

- Spread/orientation calculation: Compute the covariance matrix $\Sigma$:
  \[ \Sigma_{ts} = \frac{1}{N} \sum_{i=1}^{N} x_{c,i}^t x_{c,i}^s \]

- Find basis:
  - Compute eigendecomposition of $\Sigma$
  - Select $(u[1] ... u[k])$ to be eigenvectors with largest eigenvalues

- Project data: Project each data point onto each vector

Reconstruction

Using our principal components, reconstruct observation in original domain:
Eigenfaces \[\text{Turk, Pentland '91}\]

Input images:

Principal components:

Eigenfaces reconstruction

Each image corresponds to adding 8 principal components:
Embedding images

What is this figure showing?

Scaling up

- Covariance matrix can be really big!
  - $\Sigma$ is $d$ by $d$
  - Say, only 10000 features
  - finding eigenvectors is very slow...

- Use singular value decomposition (SVD)
  - finds up to $k$ eigenvectors
  - great implementations available
Summary for principal component analysis

What you need to know

• Dimensionality reduction
  – why and when it’s important

• Principal component analysis
  – high-level intuition for what the algorithm is doing
  – goal of minimizing reconstruction error
  – know to use SVD to (more efficiently) solve for basis vectors