What is clustering?

• Grouping items that “belong together” (i.e. have similar features)

• Unsupervised: we only use the features X, not the labels Y

• This is useful because we may not have any labels but we can still detect patterns
Why do we cluster?

• Summarizing data
  – Look at large amounts of data
  – Represent a large continuous vector with the cluster number

• Counting
  – Computing feature histograms

• Prediction
  – Data points in the same cluster may have the same labels

Slide credit: J. Hays, D. Hoiem
Counting and classification via clustering

- Compute a histogram to summarize the data
- After clustering, ask a human to label each group (cluster)
Image segmentation via clustering

• Separate image into coherent “objects”

Source: L. Lazebnik
Unsupervised discovery

- We don’t know what the objects in red boxes are, but we know they tend to occur in similar context
- If features = the context, objects in red will cluster together
- Then ask human for a label on one example from the cluster, and keep learning new object categories iteratively

Lee and Grauman, “Object-Graphs for Context-Aware Category Discovery”, CVPR 2010
Today and next class

• Clustering: motivation and applications
• Algorithms
  – K-means (iterate between finding centers and assigning points)
  – Mean-shift (find modes in the data)
  – Hierarchical clustering (start with all points in separate clusters and merge)
  – Normalized cuts (split nodes in a graph based on similarity)
These intensities define the three groups.
We could label every pixel in the image according to which of these primary intensities it is.
  • i.e., *segment* the image based on the intensity feature.
What if the image isn’t quite so simple?
• Now how to determine the three main intensities that define our groups?
• We need to **cluster**.
• Goal: choose three “centers” as the representative intensities, and label every pixel according to which of these centers it is nearest to.

• Best cluster centers are those that minimize SSD between all points and their nearest cluster center $c_i$:

$$\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} ||p - c_i||^2$$
Clustering

• With this objective, it is a “chicken and egg” problem:
  – If we knew the **cluster centers**, we could allocate points to groups by assigning each to its closest center.
  – If we knew the **group memberships**, we could get the centers by computing the mean per group.

Source: K. Grauman
K-means clustering

- Basic idea: randomly initialize the $k$ cluster centers, and iterate between the two steps we just saw.

1. *Randomly* initialize the cluster centers, $c_1, \ldots, c_K$
2. *Given cluster centers*, determine points in each cluster
   - For each point $p$, find the closest $c_i$. Put $p$ into cluster $i$
3. *Given points in each cluster*, solve for $c_i$
   - Set $c_i$ to be the mean of points in cluster $i$
4. If $c_i$ have changed, repeat Step 2

Properties
- Will always converge to *some* solution
- Can be a “local minimum”
  - does not always find the global minimum of objective function:

$$\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} ||p - c_i||^2$$

Source: Steve Seitz
K-means

1. Ask user how many clusters they’d like. 
   *(e.g. $k=5$)*
K-means

1. Ask user how many clusters they’d like. \((e.g. \, k=5)\)

2. Randomly guess \(k\) cluster Center locations

Source: A. Moore
K-means

1. Ask user how many clusters they’d like. *(e.g. k=5)*

2. Randomly guess k cluster Center locations

3. Each datapoint finds out which Center it’s closest to. (Thus each Center “owns” a set of datapoints)
K-means

1. Ask user how many clusters they’d like. *(e.g. k=5)*

2. Randomly guess k cluster Center locations

3. Each datapoint finds out which Center it’s closest to.

4. Each Center finds the centroid of the points it owns

Source: A. Moore
K-means

1. Ask user how many clusters they’d like. *(e.g. k=5)*

2. Randomly guess k cluster Center locations

3. Each datapoint finds out which Center it’s closest to.

4. Each Center finds the centroid of the points it owns...

5. ...and jumps there

6. ...Repeat until terminated!

Source: A. Moore
K-means converges to a local minimum

Figure from Wikipedia
K-means clustering

• Java demo
http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

• Matlab demo
http://www.cs.pitt.edu/~kovashka/cs1699/kmeans_demo.m
Time Complexity

- Let $n =$ number of instances, $d =$ dimensionality of the features, $k =$ number of clusters
- Assume computing distance between two instances is $O(d)$
- Reassigning clusters:
  - $O(kn)$ distance computations, or $O(knd)$
- Computing centroids:
  - Each instance vector gets added once to a centroid: $O(nd)$
- Assume these two steps are each done once for a fixed number of iterations $I$: $O(Iknd)$
  - Linear in all relevant factors

Adapted from Ray Mooney
Another way of writing objective

- **K-means:** Let \( r_{nk} = 1 \) if instance \( n \) belongs to cluster \( k \), 0 otherwise

\[
\mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}
\]

\[
J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \| x_n - \mu_k \|^2
\]

- **K-medoids** (more general distances):

\[
\tilde{J} = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \mathcal{V}(x_n, \mu_k)
\]
Distance Metrics

- Euclidian distance ($L_2$ norm):
  \[ L_2(\vec{x}, \vec{y}) = \sum_{i=1}^{m} (x_i - y_i)^2 \]

- $L_1$ norm:
  \[ L_1(\vec{x}, \vec{y}) = \sum_{i=1}^{m} |x_i - y_i| \]

- Cosine Similarity (transform to a distance by subtracting from 1):
  \[ 1 - \frac{\vec{x} \cdot \vec{y}}{||\vec{x}|| \cdot ||\vec{y}||} \]
Segmentation as clustering

Depending on what we choose as the feature space, we can group pixels in different ways.

Grouping pixels based on intensity similarity

Feature space: intensity value (1-d)

Source: K. Grauman
quantization of the feature space; segmentation label map

Source: K. Grauman
Segmentation as clustering

Depending on what we choose as the feature space, we can group pixels in different ways.

Grouping pixels based on color similarity

Feature space: color value (3-d)

Source: K. Grauman
K-means: pros and cons

Pros
• Simple, fast to compute
• Converges to local minimum of within-cluster squared error

Cons/issues
• Setting k?
  – One way: silhouette coefficient
• Sensitive to initial centers
  – Use heuristics or output of another method
• Sensitive to outliers
• Detects spherical clusters

Adapted from K. Grauman
Today

• Clustering: motivation and applications
• Algorithms
  – K-means (iterate between finding centers and assigning points)
  – Mean-shift (find modes in the data)
  – Hierarchical clustering (start with all points in separate clusters and merge)
  – Normalized cuts (split nodes in a graph based on similarity)
Mean shift algorithm

- The mean shift algorithm seeks *modes* or local maxima of density in the feature space.

Source: K. Grauman
Kernel density estimation

\[ K(x_i - \mu) = e^{-\frac{||x_i - \mu||^2}{2\sigma^2}} \]

Adapted from D. Hoiem
Mean shift

Search window

Center of mass

Mean Shift vector

Slide by Y. Ukrainitz & B. Sarel
Mean shift

Search window
Center of mass
Mean Shift vector

Slide by Y. Ukrainitz & B. Sarel
Mean shift
Mean shift

Search window

Center of mass

Mean Shift vector

Slide by Y. Ukrainitz & B. Sarel
Mean shift

Search window
Center of mass
Mean Shift vector

Slide by Y. Ukrainitz & B. Sarel
Points in same cluster converge
Mean shift clustering

- Cluster: all data points in the attraction basin of a mode
- Attraction basin: the region for which all trajectories lead to the same mode
Computing the Mean Shift

Simple Mean Shift procedure:
• Compute mean shift vector

• Translate the Kernel window by \( m(x) \)

\[
m(x) = \frac{\sum_{i=1}^{n} x_i K(x_i - x)}{\sum_{i=1}^{n} K(x_i - x)} - x
\]

\[
K(x_i - x) = e^{-\frac{||x_i - x||^2}{\sigma}}
\]

Adapted from Y. Ukrainitz & B. Sarel
Mean shift clustering/segmentation

- Compute features for each point (intensity, word counts, etc)
- Initialize windows at individual feature points
- Perform mean shift for each window until convergence
- Merge windows that end up near the same “peak” or mode

Source: D. Hoiem
Mean shift segmentation results

http://www.caip.rutgers.edu/~comanici/MSPAMI/msPamiResults.html
Mean shift segmentation results
Mean shift

• **Pros:**
  – Does not assume shape on clusters
  – Robust to outliers

• **Cons/issues:**
  – Need to choose window size
  – Expensive: $O(I n^2 d)$
    • Search for neighbors could be sped up in lower dimensions *
  – Does not scale well with dimension of feature space

* [http://lear.inrialpes.fr/people/triggs/events/iccv03/cdrom/iccv03/0456_georgescu.pdf](http://lear.inrialpes.fr/people/triggs/events/iccv03/cdrom/iccv03/0456_georgescu.pdf)
Mean-shift reading

• Nicely written mean-shift explanation (with math)
  http://saravanananthirumuruganathan.wordpress.com/2010/04/01/introduction-to-mean-shift-algorithm/
  • Includes .m code for mean-shift clustering

• Mean-shift paper by Comaniciu and Meer
  http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.76.8968

• Adaptive mean shift in higher dimensions
  http://lear.inrialpes.fr/people/triggs/events/iccv03/cdrom/iccv03/0456_georgescu.pdf
  https://pdfs.semanticscholar.org/d8a4/d6c60d0b833a82cd92059891a6980ff54526.pdf

Source: K. Grauman
Today

- Clustering: motivation and applications
- Algorithms
  - K-means (iterate between finding centers and assigning points)
  - Mean-shift (find modes in the data)
  - Hierarchical clustering (start with all points in separate clusters and merge)
  - Normalized cuts (split nodes in a graph based on similarity)
Hierarchical Agglomerative Clustering (HAC)

• Assumes a *similarity function* for determining the similarity of two instances.
• Starts with all instances in separate clusters and then repeatedly joins the two clusters that are most similar until there is only one cluster.
• The history of merging forms a binary tree or hierarchy.
HAC Algorithm

Start with all instances in their own cluster.
Until there is only one cluster:
   Among the current clusters, determine the two clusters, $c_i$ and $c_j$, that are most similar.
   Replace $c_i$ and $c_j$ with a single cluster $c_i \cup c_j$
Agglomerative clustering

1. Say “Every point is its own cluster”
Agglomerative clustering

1. Say “Every point is its own cluster”
2. Find “most similar” pair of clusters

Copyright © 2001, 2004, Andrew W. Moore
Agglomerative clustering

1. Say “Every point is its own cluster”
2. Find “most similar” pair of clusters
3. Merge it into a parent cluster
Agglomerative clustering

1. Say “Every point is its own cluster”
2. Find “most similar” pair of clusters
3. Merge it into a parent cluster
4. Repeat
Agglomerative clustering

1. Say “Every point is its own cluster”
2. Find “most similar” pair of clusters
3. Merge it into a parent cluster
4. Repeat
Agglomerative clustering

How many clusters?
- Clustering creates a dendrogram (a tree)
- To get final clusters, pick a threshold
  - max number of clusters or
  - max distance within clusters (y axis)

Adapted from J. Hays
Cluster Similarity

- How to compute similarity of two clusters each possibly containing multiple instances?

  - **Single Link:** Similarity of *two most similar* members.
    \[
    \text{sim}(c_i, c_j) = \max_{x \in c_i, y \in c_j} \text{sim}(x, y)
    \]

  - **Complete Link:** Similarity of *two least similar* members.
    \[
    \text{sim}(c_i, c_j) = \min_{x \in c_i, y \in c_j} \text{sim}(x, y)
    \]

  - **Group Average:** Average similarity between members.

Adapted from Ray Mooney
Today

- Clustering: motivation and applications
- Algorithms
  - K-means (iterate between finding centers and assigning points)
  - Mean-shift (find modes in the data)
  - Hierarchical clustering (start with all points in separate clusters and merge)
  - Normalized cuts (split nodes in a graph based on similarity)
Images as graphs

**Fully-connected** graph

- node (vertex) for every pixel
- link between every pair of pixels, \( p, q \)
- affinity weight \( w_{pq} \) for each link (edge)
  - \( w_{pq} \) measures similarity
    - similarity is *inversely proportional* to difference (in color and position…)

Source: Steve Seitz
Segmentation by Graph Cuts

Break Graph into Segments

- Want to delete links that cross **between** segments
- Easiest to break links that have low similarity (low weight)
  - similar pixels should be in the same segments
  - dissimilar pixels should be in different segments

Source: Steve Seitz
Cuts in a graph: Min cut

Link Cut
- set of links whose removal makes a graph disconnected
- cost of a cut:
  \[ \text{cut}(A, B) = \sum_{p \in A, q \in B} w_{p,q} \]

Find minimum cut
- gives you a segmentation
- fast algorithms exist for doing this

Source: Steve Seitz
Minimum cut

- Problem with minimum cut:
  Weight of cut proportional to number of edges in the cut; tends to produce small, isolated components.

Fig. 1. A case where minimum cut gives a bad partition.

[Shi & Malik, 2000 PAMI]

Source: K. Grauman
Cuts in a graph: Normalized cut

- Normalize for size of segments:

\[ \text{cut}(A, B) = \sum_{p \in A, q \in B} w_{p,q} \]

\[ \text{Ncut}(A, B) = \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(A, B)}{\text{assoc}(B, V)} \]

\[ \text{assoc}(A, V) = \text{sum of weights of all edges that touch A} \]

- Ncut value small when we get two clusters with many edges with high weights within them, and few edges of low weight between them


Adapted from Steve Seitz
How to evaluate performance?

- Might depend on application
- Purity

\[ \text{purity}(\Omega, \mathcal{C}) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j| \]

where \( \Omega = \{\omega_1, \omega_2, \ldots, \omega_K\} \) is the set of clusters

and \( \mathcal{C} = \{c_1, c_2, \ldots, c_J\} \) is the set of classes

---

How to evaluate performance

• See Murphy Sec. 25.1 for another two metrics (Rand index and mutual information)
Clustering Strategies

• **K-means**  
  – Iteratively re-assign points to the nearest cluster center

• **Mean-shift clustering**  
  – Estimate modes

• **Graph cuts**  
  – Split the nodes in a graph based on assigned links with similarity weights

• **Agglomerative clustering**  
  – Start with each point as its own cluster and iteratively merge the closest clusters