Machine Learning (CS 567)

Fall 2008

Time: T-Th 5:00pm - 6:20pm
Location: GFS 118

Instructor: Sofus A. Macskassy (macskass@usc.edu)
Office: SAL 216
Office hours: by appointment

Teaching assistant: Cheol Han (cheolhan@usc.edu)
Office: SAL 229
Office hours: M 2-3pm, W 11-12

Class web page:
http://www-scf.usc.edu/~csci567/index.html
Administrative - Projects

- Attendance is mandatory (points off if not there)
  - Let me know *beforehand* if you cannot make a session (there are 4 sessions, let me know about each)

- Project paper is due on Dec 2
  - Email to me (macskass@usc.edu) + hard copy
  - Also turn in a sheet outlining who did what and each team member should sign it *in my presence* when you turn it in.

- Slides must be emailed to me (macskass@usc.edu) by noon on the day of presentation (pdf or powerpoint), or bring it on a usb key
Administrative - Presentations

• Length:
  – 25 minutes for 4 member teams.
  – 20 minutes for 3 member teams.
  – 15 minutes for single and 2 member teams.
  – These times are very strict and I will stop your talk even if you are not done (and grading will be done according to when I stop you).
  – I suggest you prepare a presentation ~2 minutes less than allotted to have time to move from one group to next.

• Presentation must include:
  – Problem formulation
  – Data
  – Methods used and evaluation metric
    • How is this new or different from prior work.
  – Results
Administrative - Papers

- Final Papers *must* include:
  - Problem formulation
  - Related work
  - What is new
  - Data characteristics
  - Methods used and evaluation metrics
  - Results and significance of your findings
Administrative – Project Grading

- Attendance: 5%
- Presentation: 20%
  - Clarity of presentation (what the problem is, what you did, what your results were and what they mean); did you include necessary elements.
- Paper clarity: 25%
  - Clarity of writing and are the required elements present. Also, after reading the paper, can a reader understand it well enough to replicate your work and results.
- Research: 50%
  - The amount of work you did, the quality of the work, the thoroughness of your evaluation and whether you clearly have applied what you have learned in class in the appropriate manner.
Unsupervised Learning

- In supervised learning, we have data in the form of pairs \(<x,y>\), where \(y = f(x)\). The goal is to approximate \(f\).
- In **unsupervised learning**, the data just contains \(x\)!
- The main goal is to find *structure* in the data
- Potential uses:
  - Visualization of the data
  - Data compression
  - Density estimation: what distribution generated the data?
  - Pre-processing step for supervised learning
  - Novelty detection
- The definition of “ground truth” is often missing (no clear error function, like in supervised learning)
Other potential uses:
- Density Estimation
  - Learn $P(X)$ given training data for $X$
- Clustering
  - Partition data into clusters
- Dimensionality Reduction
  - Discover low-dimensional representation of data
- Blind Source Separation
  - Unmixing multiple signals
Density Estimation

- Given: $S = \{x_1, x_2, \ldots, x_N\}$
- Find: $P(x)$
- Search problem:
  $$\arg\max_h P(S|h) = \arg\max_h \sum_i \log P(x_i|h)$$
Unsupervised Fitting of the Naïve Bayes Model

- $y$ is discrete with $K$ values
  \[ P(x) = \sum_k P(y=k) \prod_j P(x_j \mid y=k) \]
- finite mixture model
- we can think of each $y=k$ as a separate “cluster” of data points
The Expectation-Maximization Algorithm (1): Hard EM

- Learning would be easy if we knew $y_i$ for each $x_i$
- Idea: guess them and then iteratively revise our guesses to maximize $P(S|h)$

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$y_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$y_1$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$y_2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_N$</td>
<td>$y_N$</td>
</tr>
</tbody>
</table>
1. Guess initial $y$ values to get “complete data”

2. M Step: Compute probabilities for hypotheses (model) from complete data [Maximum likelihood estimate of the model parameters]

3. E Step: Classify each example using the current model to get a new $y$ value [Most likely class $\hat{y}$ of each example]

4. Repeat steps 2-3 until convergence

We will come back to EM later.
What is clustering?

• Clustering is grouping similar objects together.
  – To establish prototypes or detect outliers
  – To simplify data for further analysis/learning.
  – To visualize data (in conjunction with dimensionality reduction).

• Clusterings are usually not “right” or “wrong”—different clusterings can reveal different things about the data.

• A more direct measure of “goodness” can be obtained if the clustering is a first step towards supervised learning, or if it is used for data compression.

• There are two major types of clustering: flat and hierarchical
What is clustering?

- Clustering is grouping similar objects together:
  - To establish prototypes or detect outliers
  - To simplify data for further analysis/learning
  - To visualize data (in conjunction with dimensionality reduction).

- Clusterings are usually not "right" or "wrong"—different clusterings can reveal different things about the data.

- A more direct measure of "goodness" can be obtained if the clustering is a first step towards supervised learning, or if it is used for data compression.

- There are two major types of clustering: flat and hierarchical

This is an important point! Whenever you see clustering there is always the question of whether this is the "right" clustering, or how good it is. No one has a good answer to this, since many clusters are generally "reasonable".
**K-means clustering**

- One of the most commonly-used clustering algorithms, because it is easy to implement and quick to run.
- Assumes the objects (instances) to be clustered are $n$-dimensional vectors.
- Assumes that there exists a *distance or similarity measure* between these instances.
- The goal is to *partition* the data in $K$ disjoint subsets.
- Ideally, we want to get a partition that “reflects the structure” of the data.
**K-means clustering with real-valued data**

- **Inputs:**
  - A set of n-dimensional real vectors \( \{x_1, x_2, \ldots, x_m\} \)
  - \( K \), the desired number of clusters.

- **Output:** A mapping of the vectors into K clusters (disjoint subsets), \( C : \{1,\ldots,m\} \rightarrow \{1,\ldots,K\} \)

1. Initialize \( C \) randomly.
2. Repeat:
   a) Compute the **centroid** of each cluster (which is the mean of all the instances in the cluster)
   b) Assign each vector to the cluster whose centroid is closest, in terms of Euclidean distance

Until \( C \) stops changing.
Clustering Example

Initial data

[Image of a scatter plot with red dots]
Clustering Example

assign into 3 clusters randomly
Clustering Example

compute centroids
Clustering Example

reassign clusters
Clustering Example

recompute centroids
Clustering Example

reassign clusters
Clustering Example

recompute centroids – done!

[Diagram of clustering example]
Clustering Example

What if we don’t know the right number of clusters?
assign into 4 clusters randomly
Clustering Example

compute centroids
Clustering Example

reassign clusters
Clustering Example

recompute centroids
Clustering Example

reassign clusters
Clustering Example

recompute centroids – done!
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. 
   \textit{(e.g. } k=5 \text{)}

2. Randomly guess \( k \) cluster Center locations
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
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!
K-means Start

Advance apologies: in Black and White this example will deteriorate

Example generated by Dan Pelleg’s super-duper fast K-means system:

K-means continues ...
K-means continues...
K-means continues ...
K-means continues ...
K-means continues...

...
K-means continues ...
K-means continues...
K-means continues...

...
K-means terminates
Questions

• What is K-means trying to optimize?
• Will it terminate?
• Will it always find the same answer?
• How should we choose the initial cluster centers?
• Can we automatically choose the number of centers?
Example application: color quantization

- Suppose you have an image stored with 24 bits per pixel.
- You want to compress it so that you use only 8 bits per pixel (256 colors)
- You want the compressed image to look *as similar as possible* to the original image.
- In general, you may want to take continuous data and map it to a discrete space; this is called **vector quantization**
- If we want to send the image over, now we can send the compressed version, plus the color map.
- If you only send the compressed version, you want to minimize the **reconstruction error**
Loss (distortion) function

- In a similar way, suppose we want to send all the instances over a communication channel.
- In order to compress the message, we cluster the data and **encode each instance** as the center of the cluster to which it belongs.
- The **reconstruction error** for real-valued data can be measured as the Euclidian distance between the true value and its encoding.
- Note that if we encoded the instance using the center of a different cluster, the error would be larger!
- If we try to minimize the error, we take its derivative, set it to 0, and see that the center of a cluster should be the mean of its instances.
Distortion

Given...

- an encoder function: $\text{ENCODE} : \mathbb{R}^n \rightarrow [1..k]$
- a decoder function: $\text{DECODE} : [1..k] \rightarrow \mathbb{R}^n$

Define...

$$\text{Distortion} = \sum_{i=1}^{m} \left( x_i - \text{DECODE}[\text{ENCODE}(x_i)] \right)^2$$

We may as well write

$$\text{DECODE}[j] = \mu_{C(i)}$$

so

$$\text{Distortion} = \sum_{i=1}^{m} \left( x_i - \mu_{C(i)} \right)^2$$
The Minimal Distortion

\[ \text{Distortion} = \sum_{i=1}^{m} (x_i - \mu_{C(i)})^2 \]

What properties must centers \( c_1, c_2, \ldots, c_k \) have when distortion is minimized?
The Minimal Distortion (1)

\[ \text{Distortion} = \sum_{i=1}^{m} (x_i - \mu_{C(i)})^2 \]

What properties must centers \( c_1, c_2, \ldots, c_k \) have when distortion is minimized?

(1) \( x_i \) must be encoded by its nearest center.

...why?

\[ \mu_{C(i)} = \text{the nearest center} \]

..at the minimal distortion

Otherwise distortion could be reduced by replacing \( \text{ENCODE}[x_i] \) by the nearest center.
The Minimal Distortion (2)

\[ \text{Distortion} = \sum_{i=1}^{m} \left( x_i - \mu_{C(i)} \right)^2 \]

What properties must centers \( c_1, c_2, \ldots, c_k \) have when distortion is minimized?

(2) The partial derivative of Distortion with respect to each center location must be zero.
(2) The partial derivative of Distortion with respect to each center location must be zero.

$$\text{Distortion} = \sum_{i=1}^{m} (x_i - \mu_{c(i)})^2$$

$$= \sum_{j=1}^{k} \sum_{i \in \text{OwnedBy}(c_j)} (x_i - c_j)^2$$

$$\frac{\partial \text{Distortion}}{\partial c_j} = \frac{\partial}{\partial c_j} \sum_{i \in \text{OwnedBy}(c_j)} (x_i - c_j)^2$$

$$= -2 \sum_{i \in \text{OwnedBy}(c_j)} (x_i - c_j)$$

$$= 0 \text{ (for a minimum)}$$

Thus, at a minimum: 
$$c_j = \frac{1}{|\text{OwnedBy}(c_j)|} \sum_{i \in \text{OwnedBy}(c_j)} x_i$$
Improving a suboptimal configuration...

\[
\text{Distortion} = \sum_{i=1}^{m} (x_i - \mu_{C(i)})^2
\]

What properties can be changed for centers \( c_1, c_2, \ldots, c_k \) have when distortion is not minimized?

(1) Change encoding so that \( x_i \) is encoded by its nearest center

(2) Set each Center to the centroid of points it owns.
Improving a suboptimal configuration...

\[
\text{Distortion} = \sum_{i=1}^{m} (x_i - \mu_{C(i)})^2
\]

What properties can be changed for centers \( c_1, c_2, \ldots, c_k \) have when distortion is not minimized?

(1) Change encoding so that \( x_i \) is encoded by its nearest center

(2) Set each Center to the centroid of points it owns.

There’s no point applying either operation twice in succession.

But it can be profitable to alternate.

...And that’s K-means!

*Easy to prove this procedure will terminate in a state at which neither (1) or (2) change the configuration. Why?*
Termination of $K$-means clustering

- For given data $\{x_1, \ldots, x_m\}$ and a clustering $C$, consider the sum of the squared Euclidean distance between each vector and the center of its cluster:

$$J = \sum_{i=1}^{m} \| x_i - \mu_{C(i)} \|^2$$

where $\mu_{C(i)}$ denotes the centroid of the cluster containing $x_i$.

- There are finitely many possible clusterings: at most $K^m$.

- Each time we reassign a vector to a cluster with a nearer centroid, $J$ decreases.

- Each time we recompute the centroids of each cluster, $J$ decreases (or stays the same).

- Thus, the algorithm must terminate.
Will we find the optimal configuration?

- Not necessarily.
- Can you invent a configuration that has converged, but does not have the minimum distortion? (Hint: try a fiendish $k=3$ configuration here...)
Trying to find good optima

- Idea 1: Be careful about where you start
- Idea 2: Do many runs of k-means, each from a different random start configuration
- Many other ideas floating around.
Trying to find good optima

- Idea 1: Be careful about where you start
- Idea 2: Do many runs of k-means, each from a different random start configuration
- Many other ideas floating around.

Neat trick:
Place first center on top of randomly chosen datapoint.
Place second center on datapoint that’s as far away as possible from first center.

Place j’th center on datapoint that’s as far away as possible from the closest of Centers 1 through j-1.
Choosing the number of clusters

- A difficult problem, ideas are floating around
- Delete clusters that cover too few points
- Split clusters that cover too many points
- Add extra clusters for “outliers”
- Minimum description length: minimize loss + complexity of the clustering
- ...
Choosing the number of clusters (2)

- Most common approach is to try to find the solution that minimizes the Schwarz Criterion (also related to the BIC)

\[ \text{Distortion} + \lambda (\# \text{parameters}) \log R \]

\[ = \text{Distortion} + \lambda nk \log m \]

- $n = \# \text{dimensions}$
- $k = \# \text{Centers}$
- $m = \# \text{Records}$
Why the sum of squared Euclidean distances?

Subjective reason: It produces nice, round clusters.
Why the sum of squared Euclidean distances?

Objective reason: Maximum Likelihood Principle

• Suppose the data really does divide into K clusters
• Suppose the data in each cluster is generated by a multivariate Gaussian distribution, where
  – The mean of the Gaussian is the centroid of the cluster
  – The covariance matrix is of the form $\sigma^2 I$ (It’s a “round” Gaussian.)
• Then the probability of the data is highest when the sum of squared Euclidean distances is smallest.
Why *not* the sum of squared Euclidean distances?

**Reason 1:** It produces nice, round clusters!

**Reason 2:** Differently scaled axes can dramatically affect results.

**Reason 3:** There may be symbolic attributes, which have to be treated differently.
K-means-like clustering in general

Given a set of objects,

• Choose a notion of pairwise distance / similarity between the objects.
• Choose a scoring function for the clustering
• Optimize the scoring function, to find a good clustering.

For most choices, the optimization problem will be intractable. Local optimization is often necessary.
Examples

Distance metrics:
• Euclidean distance
• Hamming distance
• Number of shared transcription factors (between two genes)
• ...

Scoring functions:
• Minimize: Summed distances between all pairs of objects in the same cluster. (Also known as “within-cluster scatter.”)
• Minimize: Maximum distance between any two objects in the same cluster. (Can be hard to optimize.)
• Maximize: Minimum distance between any two objects in different clusters.
Common uses of $K$-means

- Often used in exploratory data analysis
- Often used as a pre-processing step before supervised learning
- In one-dimension, it is a good way to discretize real-valued variables into non-uniform buckets
- Used in speech understanding/recognition to convert wave forms into one of $k$ categories (vector quantization).
A different view of clustering

• Suppose you had a classification data set, with data coming from K classes.
• But someone erased all the class labels!
• You would like to know to what class each example belongs
• This is exactly the problem solved by K-means!
• And as we saw, it gives a maximum likelihood solution, but under a very restrictive assumption.
• Let us try a move general mixture model, using Gaussian distributions
Missing Values

- Suppose we have a model of the probability distribution of the data with parameters $\theta$, and the values $y$ are missing in some (or all) of the instances.

- The likelihood of the data can be written as:

$$\log L(\theta) = \sum_{\text{complete data}} \log P(x_i, y_i \mid \theta) + \sum_{\text{incomplete data}} \log P(x_i \mid \theta)$$

- For the second term, we must consider \textit{all possible values} for $y$:

$$\sum_{\text{incomplete data}} \log P(x_i \mid \theta) = \sum_{\text{incomplete data}} \log \left( \sum_y P(x_i, y \mid \theta) \right)$$

- In our problem, $y$ is \textit{never} observed, so we only have the second term.
Expectation Maximization (EM)

- A general purpose method for learning from incomplete data
- Main idea:
  - If we had complete data we could easily maximize the likelihood
  - But because the data is incomplete, we get a summation inside the log, which makes the optimization much harder
  - So in the case of missing values, we will “fantasize” what they should be, based on the current parameter setting
  - In other words, we fill in the missing values based on our current expectation (the E step in EM)
  - Then we compute new parameters, which maximize the likelihood of the completed data (the M step in EM)
Special Case: k-Means Clustering

1. Assign an initial $y_i$ to each data point $x_i$ at random

2. M Step. For each class $k = 1, ..., K$ compute the mean:

\[ \mu_k = \frac{1}{N_k} \sum_i x_i \cdot I[y_i = k] \]

3. E Step. For each example $x_i$, assign it to the class $k$ with the nearest mean:

\[ y_i = \arg\min_k ||x_i - \mu_k|| \]

4. Repeat steps 2 and 3 to convergence
Example: Gaussian mixture models

- Suppose that all inputs $\mathbf{x}$ where real-valued, and we have some number of classes $c_1, \ldots, c_K$ with $K \geq 2$
- Assume a model similar with naïve Bayes, but with no independence assumptions: class label $c_i$ determines the probability distribution of the instances with that class, and this distribution is a multivariate Gaussian
- The parameters of the model are the probabilities of different classes, $P(c_k)$, and the parameters of the corresponding Gaussian distributions, $\mu_k$, $\Sigma_k$
- The best parameters should maximize the likelihood of the data:

$$\log L = \sum_i \left( \log P(y_i) + \log P(\mathbf{x}_i \mid y_i) \right)$$
Maximum likelihood solution

- Let $\delta_{i,k} = 1$ if $y_i = c_k$ and 0 otherwise
- The class probabilities are determined by the empirical frequency of examples in each class:

$$P(c_k) = \frac{\sum_i \delta_{i,k}}{\sum_l \sum_i \delta_{i,l}}$$

- The mean and covariance matrix for class $k$ are the empirical mean and covariance of the examples in that class:

$$\mu_k = \frac{\sum_i \delta_{i,k} \cdot x_i}{\sum_i \delta_{i,k}}$$

$$\sum_k = \frac{\sum_i \delta_{i,k} (x_i - \mu_k) \cdot (x_i - \mu_k)^T}{\sum_i \delta_{i,k}}$$
The EM algorithm

- The true EM algorithm augments the incomplete data with a probability distribution over the possible \( y \) values.

1. Start with initial naive Bayes hypothesis
2. E step: For each example, compute \( P(y_i) \) and add it to the table
3. M step: Compute updated estimates of the parameters
4. Repeat steps 2-3 to convergence.
Details of the M step

- Each example $\mathbf{x}_i$ is treated as if $y_i = k$ with probability $P(y_i = k | \mathbf{x}_i)$

\[
P(y = k) := \frac{1}{N} \sum_{i=1}^{N} P(y_i = k | \mathbf{x}_i)
\]

\[
P(x_j = v | y = k) := \frac{\sum_i P(y_i = k | \mathbf{x}_i) \cdot I(x_{ij} = v)}{\sum_{i=1}^{N} P(y_i = k | \mathbf{x}_i)}
\]
Gaussian Interpretation of K-means

- Each feature $x_j$ in class $k$ is gaussian distributed with mean $\mu_{kj}$ and constant variance $\sigma^2$

\[
P(x_j|y = k) = \frac{1}{\sqrt{2\pi \sigma}} \exp \left[ -\frac{1}{2} \frac{\|x_j - \mu_{kj}\|^2}{\sigma^2} \right]
\]
\[
\log P(x_j|y = k) = -\frac{1}{2} \frac{\|x_j - \mu_{kj}\|^2}{\sigma^2} + C
\]
\[
\arg\max_y P(x|y = k) = \arg\max_y \log P(x|y) = \arg\min_y \|x - \mu_{kj}\|^2 = \arg\min_y \|x - \mu_{kj}\|
\]

- This could easily be extended to have general covariance matrix $\Sigma$ or class-specific $\Sigma_k$
Example: Mixture of 2 Gaussians

Initial distributions

means at -0.5, +0.5
Example: Mixture of 2 Gaussians

Iteration 1
Example: Mixture of 2 Gaussians

Iteration 2
Example: Mixture of 2 Gaussians

Iteration 3
Example: Mixture of 2 Gaussians

Iteration 10
Example: Mixture of 2 Gaussians

Iteration 20
Evaluation: Test set likelihood

- Overfitting is also a problem in unsupervised learning
Choosing K

- Internal holdout likelihood
Gaussian classifiers in practice

- Note that there are many parameters, and estimating the covariance matrices may be prone to overfitting.
- Independence assumptions can be used to reduce the number of non-zero entries in the covariance matrix.
- These classifiers have a strong assumption; to address this, several Gaussians can be used for every class.
Mixture models more generally

- The term **mixture model** is typically used to describe a generative model in which there are several components, selected with some probability.
- Each component itself is a probability distribution, and is usually described in parametric form.
- For classification, the components usually correspond to the different classes, although we can assume several components for each class as well.
- For regression, the components usually cover different parts of the input space.
- Clustering can be handled using a classification mixture model, *assuming the labels on the instances are erased*.
EM for Mixture of Gaussians

- We start with an initial guess for the parameters $P(c_k), \mu_k, \Sigma_k$
- We will iterate an **expectation step (E-step)**, in which we “complete” the data, and a **maximization step (M-step)**, in which we re-compute the parameters
- In the “hard EM” version, completing the data means that each data point is assumed to be generated by exactly one Gaussian
- This is roughly equivalent to the setting of $K$-means clustering
- In the “soft EM” version (also usually known as EM), we assume that each data point could have been generated from any component
- In this case, each point will contribute to the mean and variance estimate of each component.
Hard EM for Mixture of Gaussians (1)

1. Guess an initial parameter setting

\[ \theta_j = \left< p_j, \mu_j, \Sigma_j \right>, \quad j = 1, \ldots, K \]

2. Repeat until convergence:
   
   a) **E-step:** For each \( i = 1, \ldots, m \) each each \( j = 1, \ldots, K \), compute the probability of \( x_i \) being drawn from the distribution of class \( j \) and assign the instance to the most likely class

\[ k_i = \arg \max_j P(x_i | \theta_j) \quad \text{where} \quad P(x_i | p_j, \mu_j, \Sigma_j) \propto p_j P(x_i | \mu_j, \Sigma_j) \]

   this corresponds to the assignment of instances to clusters in K-means
2. (b) **M-step:** Update the parameters of the model to maximize the likelihood of the data

\[
p_j = \frac{1}{m} \sum_{i=1}^{m} \delta_{i,j}
\]

\[
\mu_j = \frac{\sum_{i=1}^{m} \delta_{i,j} x_i}{\sum_{i=1}^{m} \delta_{i,j}}
\]

\[
\sum_j = \frac{\sum_{i=1}^{m} \delta_{i,j} (x_i - \mu_j) \cdot (x_i - \mu_j)^T}{\sum_{i=1}^{m} \delta_{i,j}}
\]

This corresponds to re-computing the cluster centers in K-means.
Soft EM for Mixture of Gaussians

1. Guess an initial parameter setting
   \[ \theta_j = \langle p_j, \mu_j, \Sigma_j \rangle, \quad j = 1, \ldots, K \]

2. Repeat until convergence:
   a) \textbf{E-step:} For each \( i = 1, \ldots, m \) each each \( j = 1, \ldots, K \), compute the probability of \( x_i \) being drawn from the distribution of class \( j \):
      \[ w_{i,j} = P(x_i \mid p_j, \mu_j, \Sigma_j) \propto p_j P(x_i \mid \mu_j, \Sigma_j) \]
   b) \textbf{M-step:} Update the parameters of the model to maximize the likelihood of the data
      \[ p_j = \frac{1}{m} \sum_{i=1}^{m} w_{i,j} \quad \mu_j = \frac{\sum_{i=1}^{m} w_{i,j} x_i}{\sum_{i=1}^{m} w_{i,j}} \]
      \[ \Sigma_j = \frac{\sum_{i=1}^{m} w_{i,j} (x_i - \mu_j) \cdot (x_i - \mu_j)^T}{\sum_{i=1}^{m} w_{i,j}} \]
EM in general

- Whenever we are trying to model data drawn probabilistically, and we have missing values in the data, EM is an option (both for missing labels and missing attributes)
- We need some structured or parameteric form of the distribution (we saw mixtures of Gaussians as examples)
- We start with a guess for the parameters of the distribution
- You can think of the E-step as trying to “complete” the data, by filling in the missing values
- the M-step will compute new parameters, given the completed data.
Comparison of hard EM and soft EM

- Soft EM does not commit to a particular value of the missing item. Instead, it considers all possible values, with some probability.
- This is a pleasing property, given the uncertainty in the value.
- The complexity of the two versions is the same:
  - Hard EM requires computing most probably values.
  - Soft EM requires computing conditional probabilities for completing the missing values.
- Soft EM is almost always the method of choice (and often when people say “EM”, the mean the soft version).
Theoretical properties of EM

• Each iteration improves the likelihood:

\[ L(\theta_{i+1} \mid D) \geq L(\theta_i \mid D) \]

• If the parameters do not change in one iteration, \( \theta_{i+1} = \theta_i \), then the gradient of the log-likelihood function is 0 at \( \theta_i \):

\[
\frac{\partial L(\theta \mid D)}{\partial \theta}(\theta_i) = 0
\]

This means that \( \theta_i \) is a min, max or saddle point

• In practice, convergence only occurs at local maxima
Variations

- If only some of the data is incomplete, the likelihood will have one component based on the complete instances and other ones based on incomplete instances.
- Sparse EM: Only compute probability at a few data points (most values will be close to 0 anyway).
- EM can be used if you have labeled data but which is possibly unreliable, in order to get better labels.
- Instead of a complete M-step, just improve the likelihood a bit.
- Note that EM can be stuck in local minima, so it has to be restarted!
- It works very well for low-dimensional problems, but can have problems if \( \theta \) is high-dimensional.
Hierarchical Clustering

- Organizes data instances into trees
- For visualization, exploratory data analysis
- **Agglomerative methods** build the tree bottom-up, successively grouping together clustering deemed most similar.
- **Divisive methods** build the tree top-down, recursively partitioning the data.
What is hierarchical clustering?

- Given instances $D=\{x_1,\ldots,x_m\}$
- A hierarchical clustering is a set of subsets (clusters) of $D$, $C=\{C_1,\ldots,C_K\}$, where
  - $D \in C$
  - $C_j$ can be assigned to the nodes of a tree such that the cluster at any node is precisely the union of the clusters, at the node’s children (if any)
Example of a hierarchical clustering

- Suppose \( D = \{1, 2, 3, 4, 5, 6, 7\} \)
- One hierarchical clustering is \( C = \{\{1\}, \{2, 3\}, \{4, 5\}, \{1, 2, 3, 4, 5\}, \{6, 7\}, \{1, 2, 3, 4, 5, 6, 7\}\}
- Leaves of the tree need not correspond to single instances.
- The branching factor of the tree is not limited.
- However, most hierarchical clustering algorithms produce binary trees, and take single instances as the smallest clusters.
Agglomerative Clustering

• Inputs: a set of instances, and pairwise distances $d(x,x')$ between them

• Outputs: A hierarchical clustering (dendrogram)

• Algorithm:
  – Assign each instance as its own cluster on a working list $W$
  – Repeat
    • Find the two clusters in $W$ that are most “similar”
    • Remove them from $W$
    • Add their union to $W$
    Until $W$ contains a single cluster with all the data objects
  – The hierarchical clustering contains all clusters appearing in $W$ at any stage of the algorithm
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
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

- Each cluster is defined only by the points it contains (not by a parameterized model)
- Very fast (using priority queue)
- No objective measure of correctness

- Distance measures
  - distance between nearest pair of points
  - distance between cluster centers
How do we measure dissimilarity between clusters?

- Distance between nearest objects ("Single-linkage" agglomerative clustering, or "nearest neighbor"): 
  \[
  \min_{x \in C, x' \in C'} d(x, x')
  \]

- Distance between farthest objects ("Complete-linkage" agglomerative clustering, or "furthest neighbor"): 
  \[
  \max_{x \in C, x' \in C'} d(x, x')
  \]

- Average distance between objects ("Group-average" agglomerative clustering): 
  \[
  \frac{1}{|C| \cdot |C'|} \sum_{x \in C, x' \in C'} d(x, x')
  \]
Dendrograms and Monotonicity

- Single-linkage, complete-linkage and group-average dissimilarity measures all share a monotonicity property:
  - Let A, B, C be clusters.
  - Let d be one of the dissimilarity measures.
  - If \( d(A, B) < d(A, C) \) and \( d(A, B) < d(B, C) \) then \( D(A, B) < d(A \cup B, C) \)

- Implication: every time agglomerative clustering merges two clusters, the dissimilarity of those clusters is \( \geq \) the dissimilarity of all previous merges.

- Dendrograms (trees depicting hierarchical clusterings) are often drawn so that the height of a node corresponds to the dissimilarity of the merged clusters.
Example: Single-linkage dendrogram
Example: Complete-linkage dendrogram
Example: Average-linkage dendrogram
Remarks

- We can form a flat clustering by cutting the tree at any height.
- Jumps in the height of the dendrogram can suggest natural cutoffs.
Divisive Clustering

- Works by recursively partitioning the instances
- But dividing such as to optimize one of the agglomerative criteria is computationally hard!
- Many heuristics for partitioning the instances have been proposed...but many violate monotonicity, making it hard to draw dendrograms.
Clustering Desiderata

Abstractly, clustering algorithm $f$ maps distance matrix $D$ between objects to partition $P$ of objects.

What is an ideal clustering algorithm?

1. **Richness**: for all $P$, exists $D$ s.t. $f(D) = P$.
2. **Scale invariance**: for all $c > 0$, $f(D) = f(cD)$.
3. **Consistency**: for any $D$, if $f(D) = P$, then for any $D'$ with smaller within-cluster distances and larger between-cluster distances, $f(D') = P$. 
Any Pair of These Possible

Using single-link clustering, we can vary the stopping condition and get:

- **Richness and consistency:**
  Stop when distances are more than $r$.

- **Scale invariance and consistency:**
  Stop when number of clusters is $k$.

- **Richness and scale invariance:**
  Stop when distances exceed $r/\max_{ij} D_{ij}$.
All Three Impossible!

It is impossible to specify a clustering algorithm that satisfies all three conditions simultaneously (Kleinberg 02).

Essentially, the proof shows that consistency and scale invariance severely constrain what partitions can be created.
Why Clustering?

Which brings up the question, why did we want to do clustering anyway? What is the task?

Can we accomplish the same things by assigning *distances* or *similarities* to pairs of objects?
Alternatives to Clustering

An embedding algorithm $e$ maps distances $D$ to locations $L$ in some vector space so that $\text{dist}(L_i, L_j)$ approximates $D_{ij}$.

- multidimensional scaling (MDS)
- singular value decomposition (SVD)
- non-negative matrix factorization (NMF)
- non-linear embedding (LLE, Isomap)
- and more

No hard decision on clusters.
• Often, raw data have very high dimension
  – Example: images of human faces

• Dimensionality Reduction:
  – Construct a lower-dimensional space that preserves information important for the task
  – Examples:
    • preserve distances
    • preserve separation between classes
    • etc.
Principal Component Analysis

Given:

- Data: n-dimensional vectors \(\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}\)
- Desired dimensionality \(m\)

Find an \(m \times n\) orthogonal matrix \(A\) to minimize

\[
\sum_i ||A^{-1}Ax_i - x_i||^2
\]

Explanation:

- \(Ax_i\) maps \(x_i\) into an \(m\)-dimensional matrix \(x'_i\)
- \(A^{-1}Ax_i\) maps \(x'_i\) back to \(n\)-dimensional space
- We minimize the "squared reconstruction error" between the reconstructed vectors and the original vectors
Conceptual Algorithm

• Find a line such that when the data is projected onto that line, it has the maximum variance:
Conceptual Algorithm

- Find a new line, orthogonal to the first, that has maximum projected variance:
Repeat Until m Lines Have Been Found

- The projected position of a point on these lines gives the coordinates in the m-dimensional reduced space
A Better Numerical Method

- Compute the co-variance matrix
  \[ \Sigma = \sum_i (x_i - \mu) \cdot (x_i - \mu)^T \]

- Compute the singular value decomposition
  \[ \Sigma = U \cdot D \cdot V^T \]

  where
  - the columns of U are the eigenvectors of \( \Sigma \)
  - D is a diagonal matrix whose elements are the square roots of the eigenvalues of \( \Sigma \) in descending order
  - \( V^T \) are the projected data points

- Replace all but the m largest elements of D by zeros
Example: Eigenfaces

- Database of 128 carefully-aligned faces
- Here are the first 15 eigenvectors:
Face Classification in Eigenspace is Easier

• Nearest Mean classifier (Turk & Pentland)
  \[ \hat{y} = \arg\min_k \| A\mathbf{x} - A\mu_k \| \]

• Accuracy
  – variation in lighting: 96%
  – variation in orientation: 85%
  – variation in size: 64%
PCA is a useful preprocessing step

- Helps all LTU algorithms by making the features more independent
- Helps decision tree algorithms
- Helps nearest neighbor algorithms by discovering the distance metric

- Fails when data consists of multiple separate clusters
  - mixtures of PCAs can be learned too
Non-Linear Dimensionality Reduction: ISOMAP

- Replace Euclidean distance by geodesic distance
  - Construct a graph where each point is connected to its k nearest neighbors by an edge AND any pair of points are connected if they are less than $\varepsilon$ apart
  - Construct an N x N matrix D in which $D[i,j]$ is the shortest path in the graph connecting $x_i$ to $x_j$
  - Apply SVD to D and keep the m most important dimensions