Unsupervised Learning: Clustering

- Partition unlabeled examples into disjoint clusters such that:
  - Examples in the same cluster are very similar.
  - Examples in different clusters are very different.
Unsupervised Learning: Clustering

- Partition unlabeled examples into disjoint clusters such that:
  - Examples in the same cluster are very similar.
  - Examples in different clusters are very different.
Hierarchical Agglomerative Clustering (HAC)

- Start out with $n$ clusters, one example per cluster.
- At each step merge the *nearest* two clusters.
- Stop when there is only one cluster left, or:
  - there are only $k$ clusters left.
  - distance is above a threshold $\tau$.

- History of clustering decision can be represented as a binary tree.
The HAC Algorithm

1. let $C_i = \{x_i\}$, for $i \in 1\ldots n$
2. let $C = \{C_i\}$, for $i \in 1\ldots n$
3. while $|C| > 1$:
4. set $\langle C_i, C_j \rangle = \arg \min_{C_k \neq C_l} d(C_k, C_l)$
5. replace $C_i, C_j$ in $C$ with $C_i \cup C_j$

Q: How do we compute the distance $d$ between two clusters?
Distance Measures

- Assume a distance function between any two instances:
  - Euclidean distance \( \|x-y\| \)

- **Single Link:**
  \[ d(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\| \]

- **Complete Link:**
  \[ d(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\| \]

- **Group Average:**
  \[ d(C_i, C_j) = \frac{1}{|C_i| \times |C_j|} \sum_{x \in C_i, y \in C_j} \|x - y\| \]

- **Centroid Distance:**
  \[ d(C_i, C_j) = \|m_i - m_j\| \]
Single Link (Nearest Neighbor)

- Distance function $d(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|$.
- It favors elongated clusters.
- Equivalent with Kruskal’s MST algorithm.
Single Link
Complete Link (Farthest Neighbor)

- Distance function \( d(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\| \)
- It favors tight, spherical clusters.
- \( d(C_i, C_j) \) is the diameter of the cluster \( C_i \cup C_j \).
Complete Link
Divisive Clustering with $k$-Means

- The goal is to produce $k$ clusters such that instances are close to the cluster centroids:
  - The cluster centroid is the mean of all instances in the cluster.

- Optimization problem:

$$\hat{C} = \arg\min_C J(C)$$

$$J(C) = \sum_{i=1}^{k} \sum_{x \in C_i} \| x - m_i \|^2$$
The $k$-Means Algorithm

1. start with some seed centroids $m_1(0), m_2(0), \ldots, m_k(0)$
2. set $t \leftarrow 0$.
3. while not converged:
   4. for each $x$:
      5. set $m^{(t)}(x) \leftarrow \arg \min_{m_i^{(t)}} \| x - m_i^{(t)} \|$ ← $[E]$ step
   6. set $C_i^{(t+1)} \leftarrow \{ x \mid m^{(t)}(x) = m_i^{(t)} \}$
   7. set $m_i^{(t+1)} \leftarrow \frac{1}{|C_i^{(t+1)}|} \sum_{x \in C_i^{(t+1)}} x$ ← $[M]$ step
8. set $t \leftarrow t + 1$
The \( k \)-Means Algorithm (\( k = 2 \))

1. Pick seeds
2. Reassign clusters
3. Compute centroids
4. Reassign clusters
5. Compute centroids
6. Reassign clusters

Converged!
The $k$-Means Algorithm ($k = 2$)
The \( k \)-Means Algorithm (\( k = 2 \))
The $k$-Means Algorithm ($k = 2$)
The $k$-Means Algorithm ($k = 2$)
The $k$-Means Algorithm ($k = 2$)
The $k$-Means Algorithm ($k = 2$)
The $k$-Means Algorithm ($k = 2$)
The $k$-Means Algorithm ($k = 2$)
The $k$-Means Algorithm ($k = 2$)
The $k$-Means Algorithm

- The objective function monotonically decreases at every iteration:

\[ J^{(t)} \geq J^{(t+1)} \]
The $k$-Means Algorithm

- Optimization problem is NP-hard:
  - results depend on seed selection.
  - improve performance by providing *must-link* and/or *cannot-link* constraints ⇒ *semi-supervised clustering*.

- Time complexity for each iteration is $O(knm)$:
  - number of clusters is $k$.
  - feature vectors have dimensionality $m$.
  - total number of instances is $n$. 
Soft Clustering

- Clustering typically assumes that each instance is given a “hard” assignment to exactly one cluster.
- Does not allow uncertainty in class membership or for an instance to belong to more than one cluster.
- **Soft clustering** gives probabilities that an instance belongs to each of a set of clusters.
- Each instance is assigned a probability distribution across a set of discovered categories.
Soft Clustering with EM

• Soft version of $k$-means.
• Assumes a probabilistic model of categories that allows computing $P(c_i \mid x)$ for each category, $c_i$, for a given example $x$.
• For text, typically assume a naïve-Bayes category model.
  – Parameters $\theta = \{P(c_i), P(w_j \mid c_i) \mid i \in \{1, \ldots k\}, j \in \{1, \ldots, |V|\}\}$
Soft Clustering with EM

- Iterative method for learning probabilistic categorization model from unsupervised data.
- Initially assume random assignment of examples to categories.
- Learn an initial probabilistic model by estimating model parameters \( \theta \) from this randomly labeled data.
- Iterate following two steps until convergence:
  - **Expectation (E-step):** Compute \( P(c_i | x) \) for each example given the current model, and probabilistically re-label the examples based on these posterior probability estimates.
  - **Maximization (M-step):** Re-estimate the model parameters, \( \theta \), from the probabilistically re-labeled data.
Learning with Probabilistic Labels

- Instead of training data labeled with “hard” category labels, training data is labeled with “soft” probabilistic category labels.
- When estimating model parameters $\theta$ from training data, weight counts by the corresponding probability of the given category label.
- For example, if $P(c_1 \mid x) = 0.8$ and $P(c_2 \mid x) = 0.2$, each word $w_j$ in $x$ contributes only $0.8$ towards the counts $n_1$ and $n_{1j}$, and $0.2$ towards the counts $n_2$ and $n_{2j}$.
Naïve Bayes EM

1. Randomly assign examples probabilistic category labels.
2. Use standard naïve-Bayes training to learn a probabilistic model with parameters $\theta$ from the labeled data.
3. Until convergence or until maximum number of iterations reached:
   - **E-Step**: Use the naïve Bayes model $\theta$ to compute $P(c_i \mid x)$ for each category and example, and re-label each example using these probability values as soft category labels.
   - **M-Step**: Use standard naïve-Bayes training to re-estimate the parameters $\theta$ using these new probabilistic category labels.
Principal Component Analysis (PCA)

• A technique widely used for:
  – dimensionality reduction.
  – data compression.
  – feature extraction.
  – data visualization.

• Two equivalent definitions of PCA:
  1) Project the data onto a lower dimensional space such that the variance of the projected data is maximized.
  2) Project the data onto a lower dimensional space such that the mean squared distance between data points and their projections (average projection cost) is minimized.
Principal Component Analysis (PCA)
PCA (Maximum Variance)

- Let $X = \{x_n\}_{1 \leq n \leq N}$ be a set of observations:
  - Each $x_n \in \mathbb{R}^D$ ($D$ is the dimensionality of $x_n$).

- Project $X$ onto an $M$ dimensional space ($M < D$) such that the variance of the projected $X$ is maximized.

- Work out solution for $M = 1$, then generalize to any $M < D$. 

Lecture 10
PCA (Maximum Variance, $M = 1$)

- The lower dimensional space is defined by a vector $u_1 \in \mathbb{R}^D$.
  - Show that only direction is important $\Rightarrow$ choose $\|u_1\| = 1$.
- Each $x_n$ is projected onto a scalar $u_1^T x_n$
- The (sample) mean of the data is:
  \[
  \bar{x} = \frac{1}{N} \sum_{n=1}^{N} x_n
  \]
- The (sample) mean of the projected data is $u_1^T \bar{x}$
PCA (Maximum Variance, $M = 1$)

- The (sample) variance of the projected data:

$$\frac{1}{N} \sum_{n=1}^{N} (u_1^T x_n - u_1^T \bar{x})^2 = u_1^T S u_1$$

where $S$ is the data covariance matrix:

$$S = \frac{1}{N} \sum_{n=1}^{N} (x_n - \bar{x})(x_n - \bar{x})^T$$

- Optimization problem is:

minimize: 

$$u_1^T S u_1$$

subject to:

$$u_1^T u_1 = 1$$
PCA (Maximum Variance, $M = 1$)

- Lagrangian function:

$$L_P(u_1, \lambda_1) = u_1^T Su_1 + \lambda_1 (1 - u_1^T u_1)$$

where $\lambda_1$ is the Lagrangian multiplier for constraint $u_1^T u_1 = 1$

- Solve:

$$\frac{\partial L_P}{\partial u_1} = 0 \Rightarrow Su_1 = \lambda_1 u_1 \Rightarrow \begin{cases} u_1 \text{ is an eigenvector of } S \\ \lambda_1 \text{ is an eigenvalue of } S \end{cases}$$

$$\Rightarrow u_1^T Su_1 = \lambda_1 u_1^T u_1 = \lambda_1$$

$$\Rightarrow \lambda_1 \text{ is the largest eigenvalue of } S.$$
PCA (Maximum Variance, $M = 1$)

- $\lambda_1$ is the largest eigenvalue of $S$.
- $u_1$ is the eigenvector corresponding to $\lambda_1$: also called the first principal component.

- For $M < D$ dimensions:
  - $u_1, u_2 \ldots u_M$ are the eigenvectors corresponding to the largest eigenvalues $\lambda_1, \lambda_2 \ldots \lambda_M$ of $S$.
  - proof by induction.
Principal Component Analysis vs. Fisher Linear Discriminant

• Both methods can be used for linear dimensionality reduction.

• PCA is unsupervised:
  – it depends only on the values $x_n$.

• Fisher linear discriminant is supervised:
  – it depends on both the observations and the labels $(x_n, t_n)$.
Principal Component Analysis vs. Fisher Linear Discriminant
PCA for High-Dimensional Data

• If $N < D$, it does not make sense to use PCA for $M > N-1$:
  – The set of $N$ points define a linear subspace with dimensionality at most $N-1$.
  – PCA will find at least $D-N+1$ eigenvalues that are 0.
  – Typical algorithms for finding eigenvalues are $O(D^3)$.

• Solution:
  – Let $X$ by the $N \times D$ matrix with nth row given by $(\mathbf{x}_n - \bar{\mathbf{x}})^T$
  – Then the sample covariance matrix $S$ can be written as:

\[
S = \frac{1}{N} X^T X
\]
PCA for High-Dimensional Data

\[
Su_i = \lambda_i u_i \Rightarrow \frac{1}{N} X^T X u_i = \lambda_i u_i
\]

\[
\Rightarrow \frac{1}{N} X X^T (X u_i) = \lambda_i (X u_i)
\]

Define \( v_i = X u_i \)

\[
\Rightarrow \left( \frac{1}{N} X X^T \right) v_i = \lambda_i v_i
\]

an \( N \times N \) matrix \( \Rightarrow O(N^3) \) instead of \( O(D^3) \) cost.

- Same eigenvalues as original problem, but what are the original, principal eigenvectors?
PCA for High-Dimensional Data

\[
\left( \frac{1}{N} X X^T \right) v_i = \lambda_i v_i \Rightarrow \left( \frac{1}{N} X^T X \right) (X^T v_i) = \lambda_i (X^T v_i)
\]

\[\Rightarrow X^T v_i \text{ is an eigenvector of } S \text{ with eigenvalue } \lambda_i.\]

\[\Rightarrow u_i = \frac{X^T v_i}{\|X^T v_i\|}\]

• Summary of solution:
  1. evaluate \(X^T X\).
  2. find its eigenvectors and eigenvalues.
  3. compute the eigenvectors in the original dataspace.
PCA, Fisher & Kernels

• **Minimum error** formulation leads to the same solution [12.1.2].
  – shows how PCA can be used for compression.

• **Kernel PCA** [12.3].

• **Kernel Fisher linear discriminant** [Mika et al., 1999]