Supervised Learning vs. Unsupervised Learning

- **Supervised Learning**
  - Classification: partition examples into groups according to predefined categories
  - Regression: assign value to feature vectors
  - Requires labeled data for training
- **Unsupervised Learning**
  - Clustering: partition examples into groups when no pre-defined categories/classes are available
  - Signal separation: recover components of a mixed signal
  - Embeddings: find low dimensional representation of high dimensional data
  - Outlier detection: find unusual events (e.g. hackers)
  - Novelty detection: find changes in data
  - Only instances required, but no labels

Clustering

- Partition unlabeled examples into disjoint subsets of clusters, such that:
  - Examples within a cluster are similar
  - Examples in different clusters are different
- Discover new categories in an unsupervised manner (no sample category labels provided).

Applications of Clustering

- Exploratory data analysis
- Cluster retrieved documents in search engine
- Detecting near duplicates
  - Entity resolution
    - E.g. "Thorsten Joachims" == "Thorsten B Joachims"
  - Cheating detection
- Automated (or semi-automated) creation of taxonomies
  - E.g. phylogenetic tree
- Compression
Clustering Example

Similarity (Distance) Measures

- Euclidian distance ($L_2$ norm):
  $$L_2(\vec{x}, \vec{x}') = \sqrt{\sum_{i=1}^{N} (x_i - x_i')^2}$$
- $L_1$ norm:
  $$L_1(\vec{x}, \vec{x}') = \sum_{i=1}^{N} |x_i - x_i'|$$
- Cosine similarity:
  $$\cos(\vec{x}, \vec{x}') = \frac{\vec{x} \cdot \vec{x}'}{\|\vec{x}\| \|\vec{x}'\|}$$
- Kernels

Hierarchical Clustering

- Build a tree-based hierarchical taxonomy from a set of unlabeled examples.
- Recursive application of a standard clustering algorithm can produce a hierarchical clustering.

Agglomerative vs. Divisive Clustering

- Agglomerative (bottom-up) methods start with each example in its own cluster and iteratively combine them to form larger and larger clusters.
- Divisive (top-down) separate all examples immediately into clusters.
Hierarchical Agglomerative Clustering (HAC)

• Assumes a similarity function for determining the similarity of two clusters.

• Basic 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 \( c_i \cup c_j \).
  - The history of merging forms a binary tree or hierarchy.

Cluster Similarity

• How to compute similarity of two clusters each possibly containing multiple instances?
  - Single link: Similarity of two most similar members.
  - Complete link: Similarity of two least similar members.
  - Group average: Average similarity between members.

Computational Complexity of HAC

• In the first iteration, all HAC methods need to compute similarity of all pairs of \( n \) individual instances which is \( O(n^2) \).

• In each of the subsequent \( O(n) \) merging iterations,
  - must find smallest distance pair of clusters \( \rightarrow \)
    - Maintain heap \( O(n \log n) \)
  - it must compute the distance between the most recently created cluster and each other existing cluster. Can this be done in constant time?
  \( \rightarrow O(n^2 \log n) \) overall.

Computing Cluster Similarity

• After merging \( c_i \) and \( c_j \), the similarity of the resulting cluster to any other cluster, \( c_k \), can be computed by:
  - Single Link:
    \[
    \text{sim}(c_i \cup c_j, c_k) = \max \left( \text{sim}(c_i, c_k), \text{sim}(c_j, c_k) \right)
    \]
  - Complete Link:
    \[
    \text{sim}(c_i \cup c_j, c_k) = \min \left( \text{sim}(c_i, c_k), \text{sim}(c_j, c_k) \right)
    \]
Single-Link Example

Group Average Agglomerative Clustering

- Use average similarity across all pairs within the merged cluster to measure the similarity of two clusters.

\[
sim(c_i, c_j) = \frac{1}{|c_i \cup c_j| - |c_i \cap c_j|} \sum_{x \in c_i \cup c_j} \sum_{y \in c_i \cup c_j} \text{sim}(x, y)
\]

- Compromise between single and complete link.

Computing Group Average Similarity

- Assume cosine similarity and normalized vectors with unit length.
- Always maintain sum of vectors in each cluster.

\[
s(i) = \sum_{x \in c_i} \tilde{x}
\]

- Compute similarity of clusters in constant time:

\[
sim(c_i, c_j) = (\tilde{s}(c_i) \cdot \tilde{s}(c_j)) - \frac{(|c_i| + |c_j| - |c_i \cap c_j|)}{|c_i| \cdot |c_j|}
\]

Non-Hierarchical Clustering

- K-means clustering (“hard”)
- Mixtures of Gaussians and training via Expectation maximization Algorithm (“soft”)

Clustering Criterion

- Evaluation function that assigns a (usually real-valued) value to a clustering
  - Clustering criterion typically function of
    - within-cluster similarity and
    - between-cluster dissimilarity
- Optimization
  - Find clustering that maximizes the criterion
    - Global optimization (often intractable)
    - Greedy search
    - Approximation algorithms

K-Means Algorithm

- Input: \( k \) = number of clusters, Euclidian distance \( d \)
- Select \( k \) random instances \( \{s_1, s_2, \ldots, s_k\} \) as seeds.
- Until clustering converges or other stopping criterion:
  - For each instance \( x_i \):
    - Assign \( x_i \) to the cluster \( c_j \) such that \( d(x_i, s_j) \) is min.
  - For each cluster \( c_j \) / Update the centroid of each cluster
    - \( s_j = \mu(c_j) \)

Note: Clusters represented via centroids

\[
\mu(c) = \frac{1}{|c|} \sum_{x \in c} \tilde{x}
\]
K-means Example
(k=2)

Pick seeds
Reassign clusters
Compute centroids
Reassign clusters
Compute centroids
Reassign clusters
Converged!

Time Complexity

• Assume computing distance between two instances is \(O(N)\) where \(N\) is the dimensionality of the vectors.
• Reassigning clusters for \(n\) points: \(O(kn)\) distance computations, or \(O(knN)\).
• Computing centroids: Each instance gets added once to some centroid: \(O(nN)\).
• Assume these two steps are each done once for \(i\) iterations: \(O(iknN)\).
• Linear in all relevant factors, assuming a fixed number of iterations.

Buckshot Algorithm

Problem
- Results can vary based on random seed selection, especially for high-dimensional data.
- Some seeds can result in poor convergence rate, or convergence to sub-optimal clusterings.

- First randomly take a sample of instances of size \(n^{1/2}\)
- Run group-average HAC on this sample
- Use the results of HAC as initial seeds for K-means.
- Overall algorithm is efficient and avoids problems of bad seed selection.