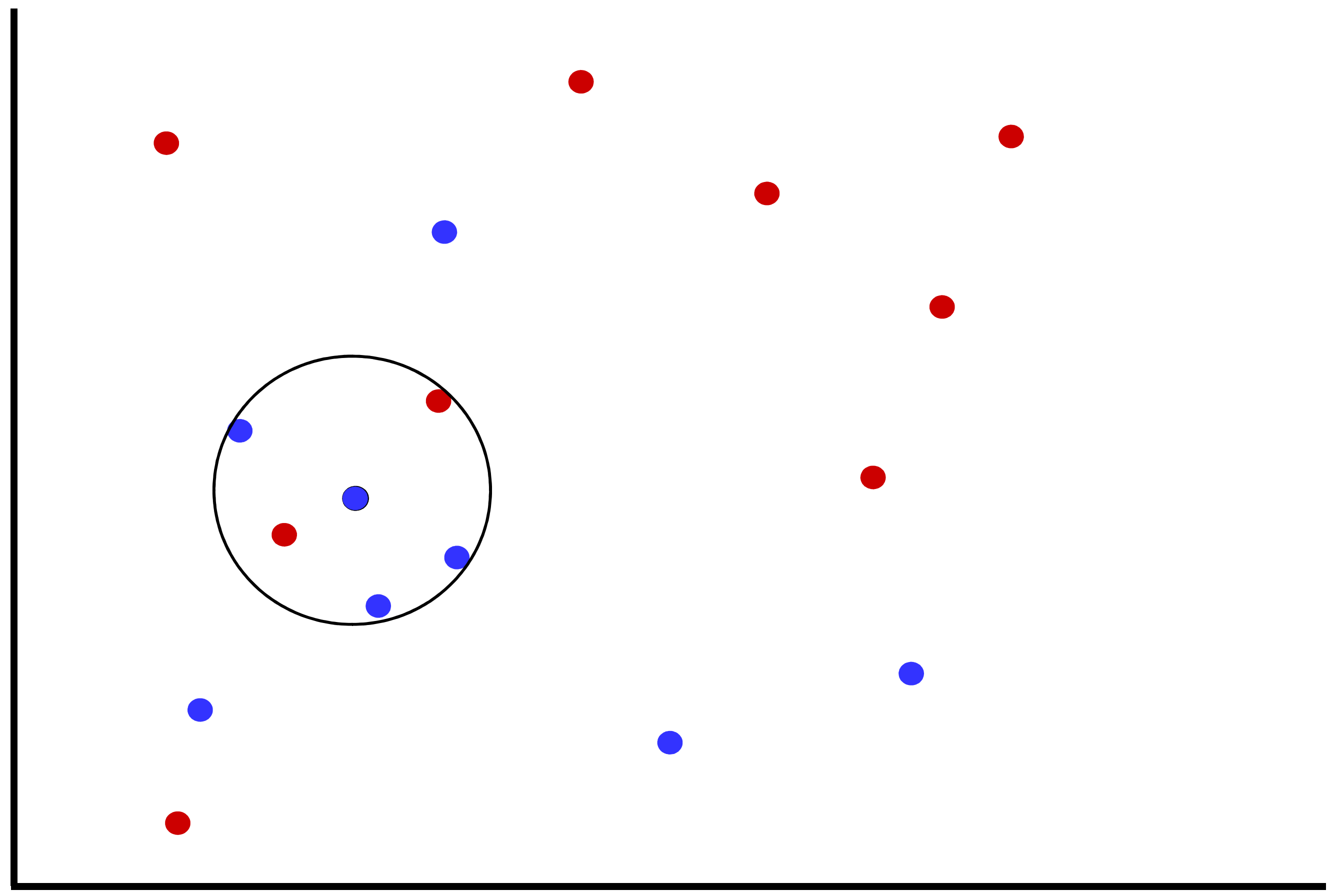
Instance Based Learning

Based on Raymond J. Mooney's slides

University of Texas at Austin

Example



Instance-Based Learning

- Unlike other learning algorithms, does not involve construction of an explicit abstract generalization but classifies new instances based on direct comparison and similarity to known training instances.
- Training can be very easy, just memorizing training instances.
- Testing can be very expensive, requiring detailed comparison to all past training instances.
- Also known as:
  - Case-based
  - Exemplar-based
  - Nearest Neighbor
  - Memory-based
  - Lazy Learning

Similarity/Distance Metrics

- Instance-based methods assume a function for determining the similarity or distance between any two instances.
- For continuous feature vectors, Euclidian distance is the generic choice:

$$d(x_i, x_j) = \sqrt{\sum_{p=1}^{n} (a_p(x_i) - a_p(x_j))^2}$$

Where $a_p(x)$ is the value of the p th feature of instance x.

- For discrete features, assume distance between two values is 0 if they are the same and 1 if they are different (e.g. Hamming distance for bit vectors).
- To compensate for difference in units across features, scale all continuous values to the interval [0,1].

Other Distance Metrics

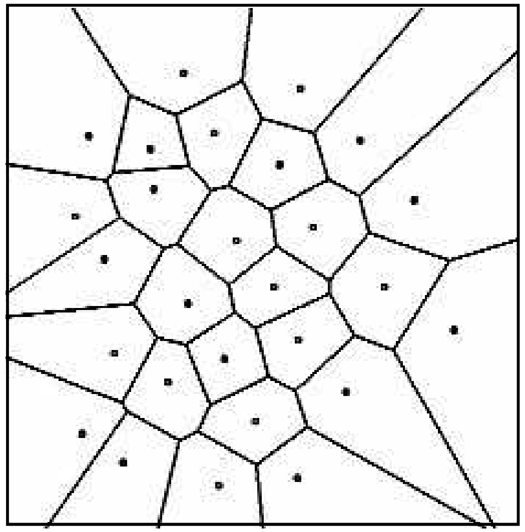
- Mahalanobis distance (→)
  - Scale-invariant metric that normalizes for variance.
- Cosine Similarity
  - Cosine of the angle between the two vectors.
  - Used in text and other high-dimensional data.
- Pearson correlation (→)
  - Standard statistical correlation coefficient.
- Edit distance
  - Used to measure distance between unbounded length strings.

K-Nearest Neighbor

- Calculate the distance between a test point and every training instance.
- Pick the *k* closest training examples and assign the test instance to the most common category amongst these nearest neighbors.
- Voting multiple neighbors helps decrease susceptibility to noise.
- Usually use odd value for *k* to avoid ties.

Implicit Classification Function

- Although it is not necessary to explicitly calculate it, the learned classification rule is based on regions of the feature space closest to each training example.
- For 1-nearest neighbor with Euclidian distance, the **Voronoi diagram** gives the complex polyhedra segmenting the space into the regions closest to each point.



Efficient Indexing

- Linear search to find the nearest neighbors is not efficient for large training sets.
- Indexing structures can be built to speed testing.
- For Euclidian distance, a **kd-tree** can be built that reduces the expected time to find the nearest neighbor to O(log *n*) in the number of training examples.
  - Nodes branch on threshold tests on individual features and leaves terminate at nearest neighbors.
- Other indexing structures possible for other metrics or string data.
  - Inverted index for text retrieval.

kd-tree

- The kd-tree is a binary tree in which every node is a k-dimensional point.
- Every non-leaf node generates a splitting hyperplane that divides the space into two subspaces.
- Points left to the hyperplane represent the left sub-tree of that node and the points right to the hyperplane by the right sub-tree.
- The hyperplane direction is chosen in the following way: every node split to sub-trees is associated with one of the k-dimensions, such that the hyperplane is perpendicular to that dimension vector.

Nearest Neighbor Variations

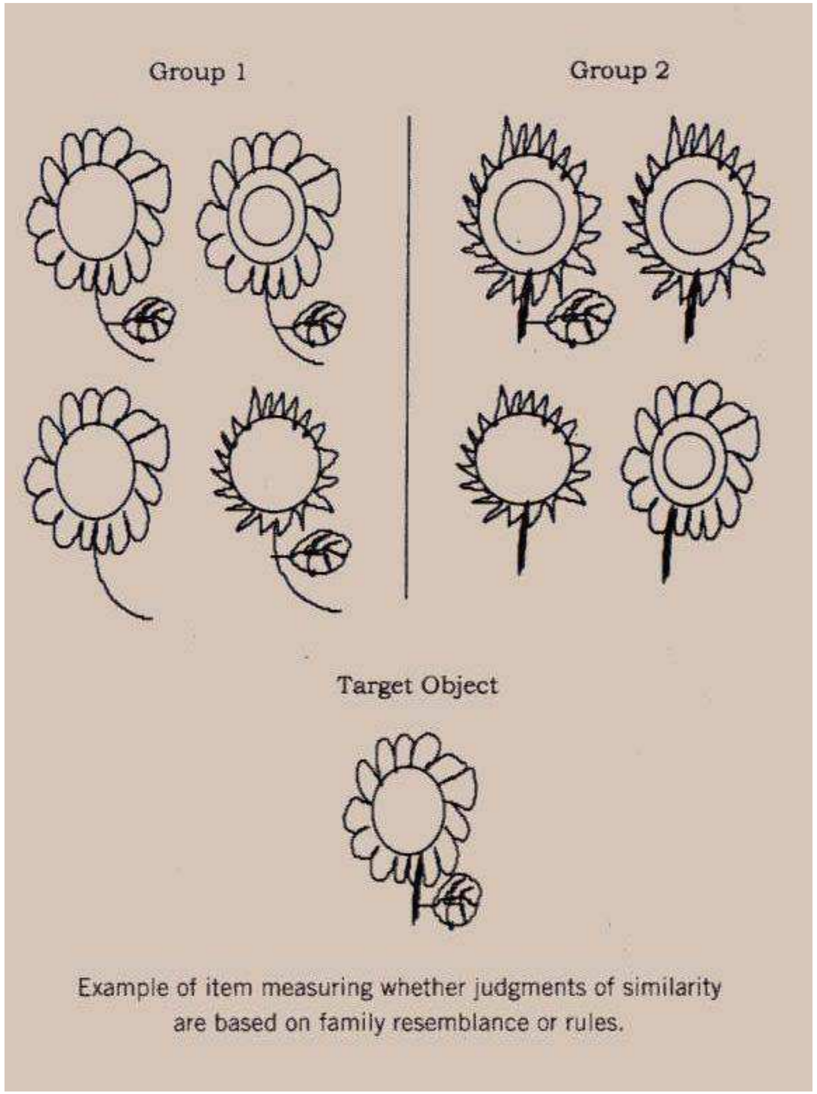
- Can be used to estimate the value of a real-valued function – regression - by taking the average function value of the *k* nearest neighbors to an input point.
- All training examples can be used to help classify a test instance by giving every training example a vote that is weighted by the inverse square of its distance from the test instance.

Feature Relevance and Weighting

- Standard distance metrics weight each feature equally when determining similarity.
  - Problematic if many features are irrelevant, since similarity along many irrelevant examples could mislead the classification.
- Features can be weighted by some measure that indicates their ability to discriminate the category of an example, such as information gain.
- Overall, instance-based methods favor global similarity over concept simplicity.

Rules and Instances in Human Learning Biases

- Psychological experiments show that people from different cultures exhibit distinct categorization biases.
- "Western" subjects favor simple rules (straight stem) and classify the target object in group 2.
- "Asian" subjects favor global similarity and classify the target object in group 1.



Example of item measuring whether judgments of similarity are based on family resemblance or rules.

Other Issues

- Can reduce storage of training instances to a small set of representative examples.
  - Support vectors in an SVM are somewhat analogous.
- Can hybridize with rule-based methods or neural-net methods.
  - Radial basis functions in neural nets and Gaussian kernels in SVMs are similar.
- Can be used for more complex relational or graph data.
  - Similarity computation is complex since it involves some sort of graph isomorphism.
- Can be used in problems other than classification.
  - Case-based planning
  - Case-based reasoning in law and business.

Conclusions

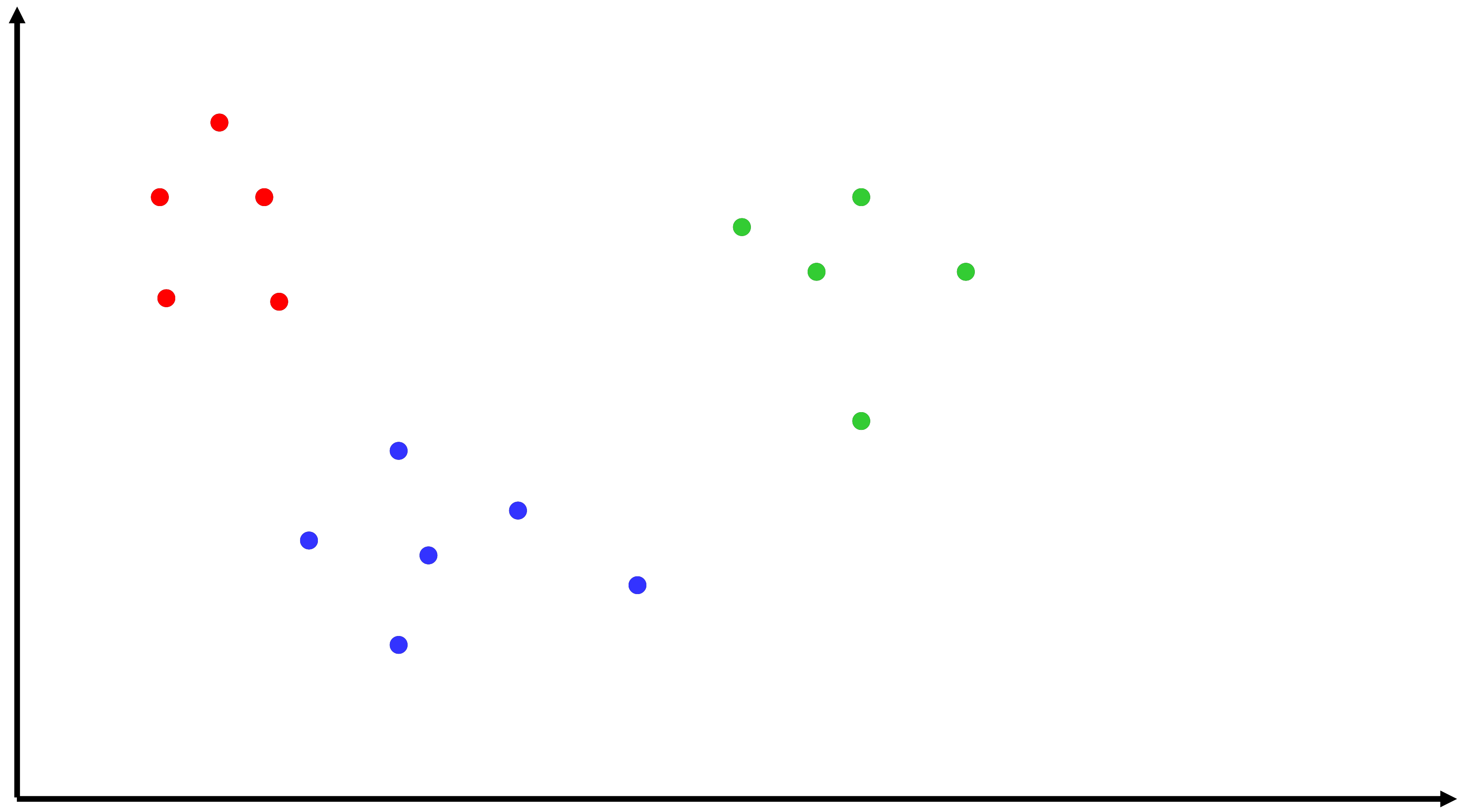
- IBL methods classify test instances based on similarity to specific training instances rather than forming explicit generalizations.
- Typically trade decreased training time for increased testing time.

Unsupervised learning. Clustering

Clustering

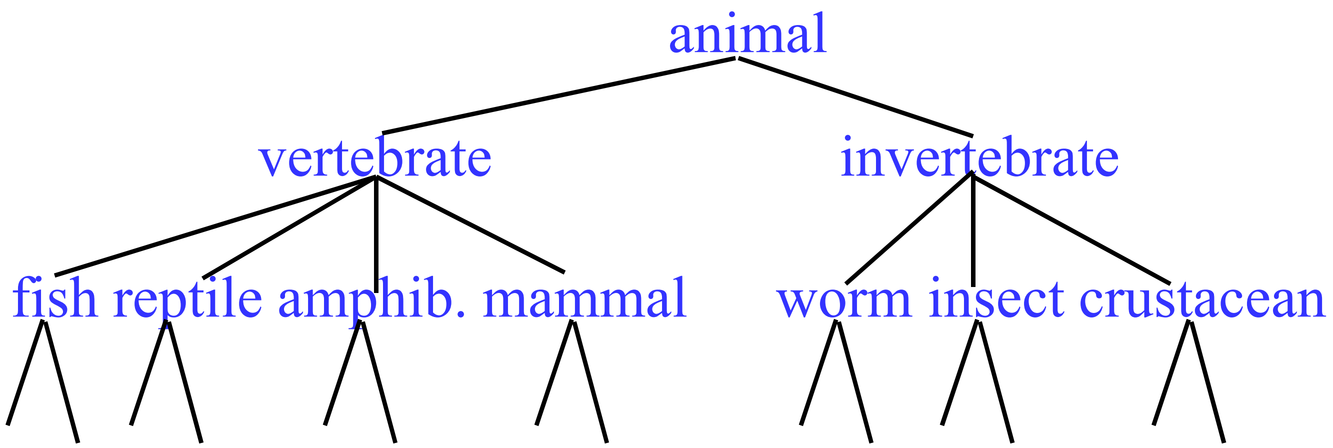
- Partition unlabeled examples into disjoint subsets of *clusters*, such that:
  - Examples within a cluster are very similar
  - Examples in different clusters are very different
- Discover new categories in an *unsupervised* manner (no sample category labels provided).

Clustering Example



Hierarchical Clustering

- Build a tree-based hierarchical taxonomy (*dendrogram*) from a set of unlabeled examples.



- Recursive application of a standard clustering algorithm can produce a hierarchical clustering.

Aglommerative vs. Divisive Clustering

- *Aglommerative* (*bottom-up*) methods start with each example in its own cluster and iteratively combine them to form larger and larger clusters.
- *Divisive* (*partitional, top-down*) separate all examples immediately into clusters.

Direct Clustering Method

- *Direct clustering* methods require a specification of the number of clusters, *k*, desired.
- A *clustering evaluation function* assigns a real-value quality measure to a clustering.
- The number of clusters can be determined automatically by explicitly generating clusterings for multiple values of *k* and choosing the best result according to a clustering evaluation function.

Hierarchical Agglomerative Clustering (HAC)

- Assumes a *similarity function* for determining the similarity of two instances.
- Starts with all instances in a separate cluster 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 $c_i \cup c_j$
```

Cluster Similarity

- Assume a similarity function that determines the similarity of two instances: *sim*(*x*,*y*).
  - Euclidean /Mahalanobis, Hamming, Cosine similarity, Pearson r etc.
- 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.

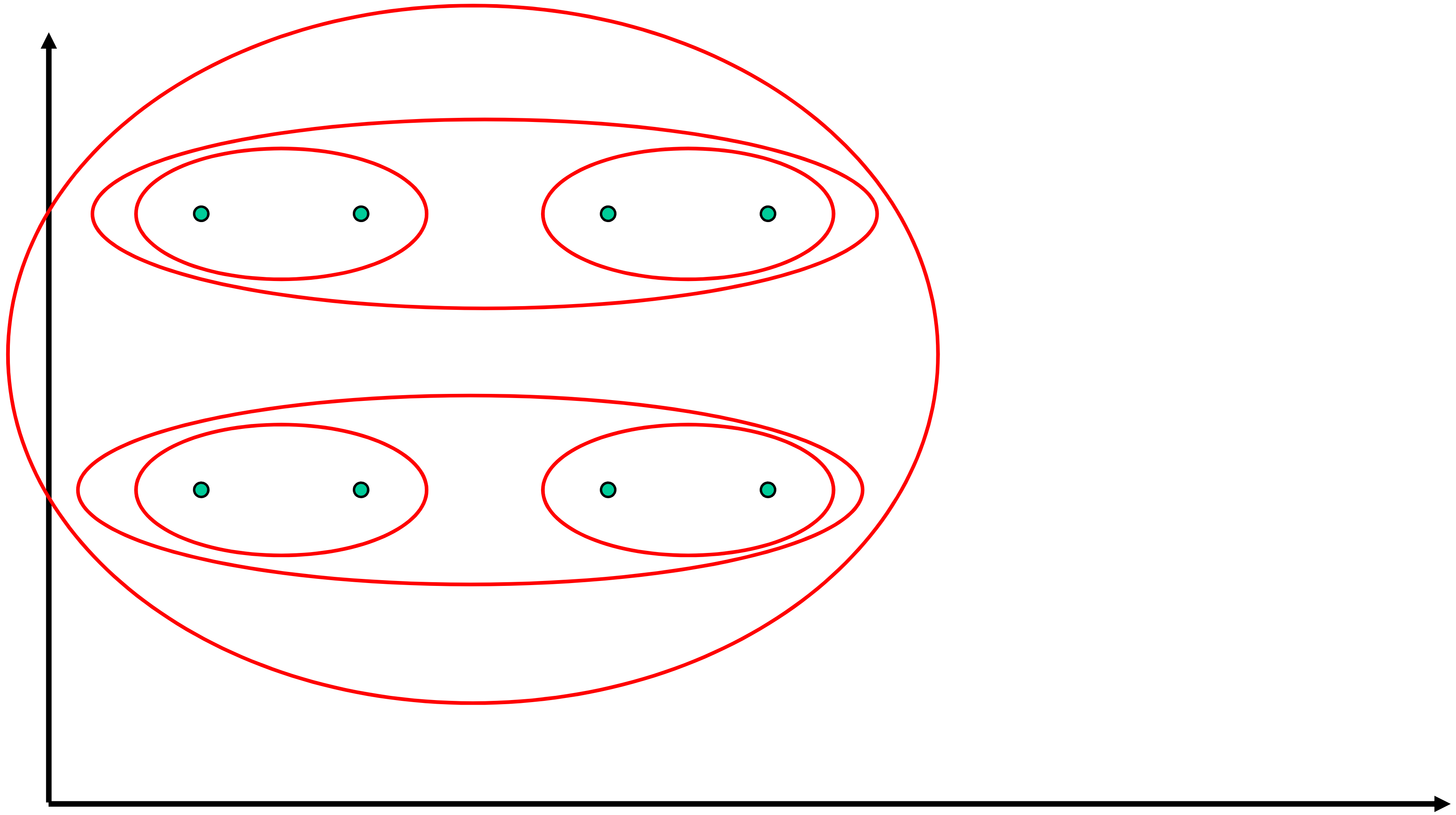
Single Link Agglomerative Clustering

- Use maximum similarity of pairs:

$$sim(c_i, c_j) = \max_{x \in c_i, y \in c_j} sim(x, y)$$

- Can result in "straggly" (long and thin) clusters due to *chaining effect*.
  - Appropriate in some domains, such as clustering islands.

Single Link Example



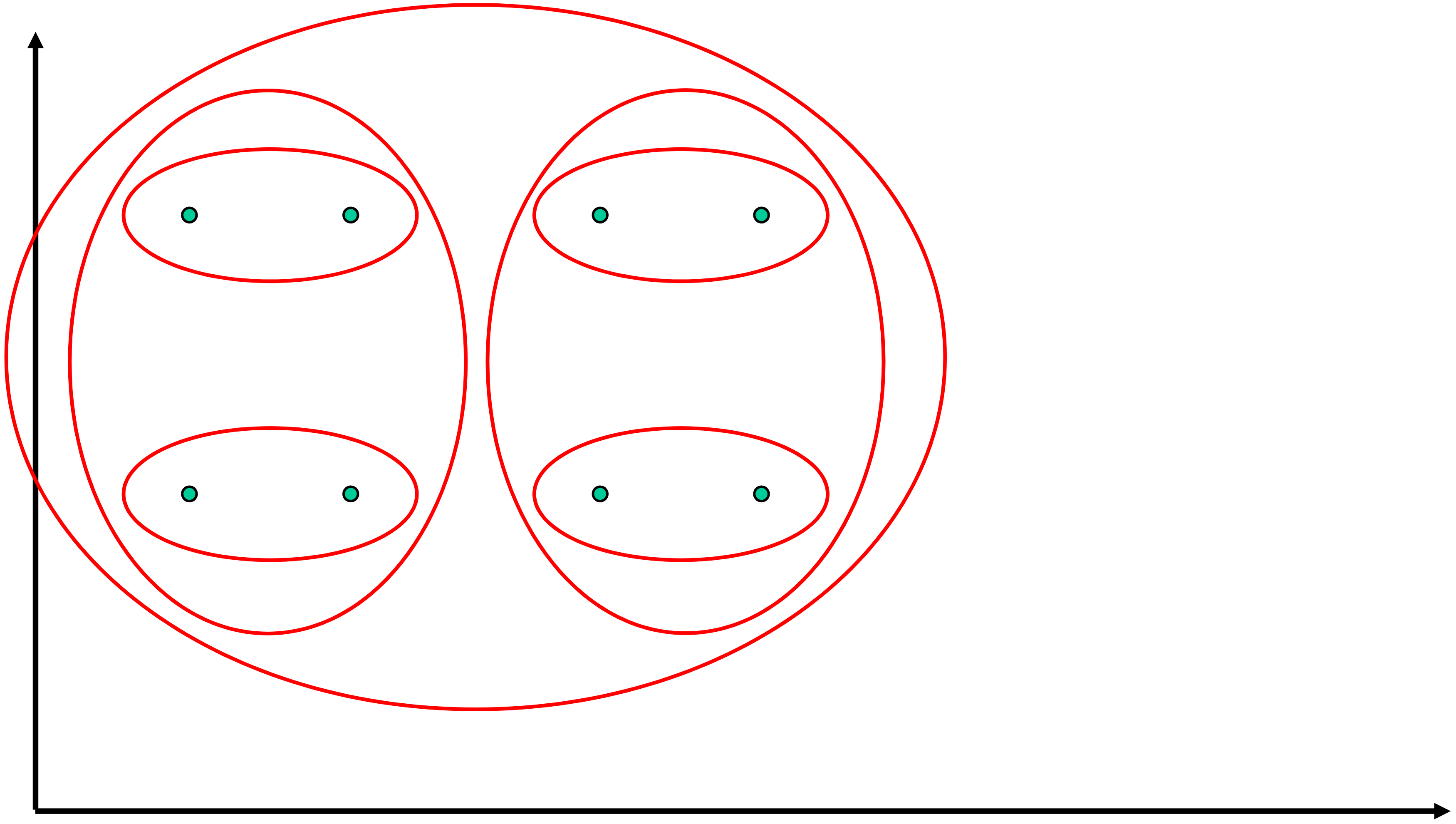
Complete Link Agglomerative Clustering

- Use minimum similarity of pairs:

$$sim(c_i,c_j) = \min_{x \in c_i, y \in c_j} sim(x, y)$$

- Makes more "tight," spherical clusters that are typically preferable.

Complete Link Example



Computational Complexity

- 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 n–2 merging iterations, it must compute the distance between the most recently created cluster and all other existing clusters.
- In order to maintain an overall O(n^2) performance, computing similarity to each other cluster must be done in constant time.

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:

$$sim((c_i \cup c_j), c_k) = \max(sim(c_i, c_k), sim(c_j, c_k))$$

  - Complete Link:

$$sim((c_i \cup c_j), c_k) = \min(sim(c_i, c_k), sim(c_j, c_k))$$

Group Average Agglomerative Clustering

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

$$sim(c_i, c_j) = \frac{1}{|c_i \cup c_j|(|c_i \cup c_j| - 1)} \sum_{\vec{x} \in (c_i \cup c_j)} \sum_{\vec{y} \in (c_i \cup c_j) : \vec{y} \neq \vec{x}} sim(\vec{x}, \vec{y})$$

- Compromise between single and complete link.
- Averaged across all ordered pairs in the merged cluster instead of unordered pairs *between* the two clusters to encourage tight clusters.

Computing Group Average Similarity

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

$$\vec{s}(c_j) = \sum_{\vec{x} \in c_j} \vec{x}$$

- Compute similarity of clusters in constant time:

$$sim(c_i, c_j) = \frac{(\vec{s}(c_i) + \vec{s}(c_j)) \bullet (\vec{s}(c_i) + \vec{s}(c_j)) - (|c_i| + |c_j|)}{(|c_i| + |c_j|)(|c_i| + |c_j| - 1)}$$

Non-Hierarchical Clustering

- Typically must provide the number of desired clusters, k.
- Randomly choose k instances as *seeds*, one per cluster.
- Form initial clusters based on these seeds.
- Iterate, repeatedly reallocating instances to different clusters to improve the overall clustering.
- Stop when clustering converges or after a fixed number of iterations.

K-Means

- Assumes instances are real-valued vectors.
- Clusters based on *centroids*, *center of gravity*, or mean of points in a cluster, *c*:

$$\vec{\mu}(\mathrm{c}) = \frac{1}{|c|} \sum_{\vec{x} \in c} \vec{x}$$

- Reassignment of instances to clusters is based on distance to the current cluster centroids.

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:

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

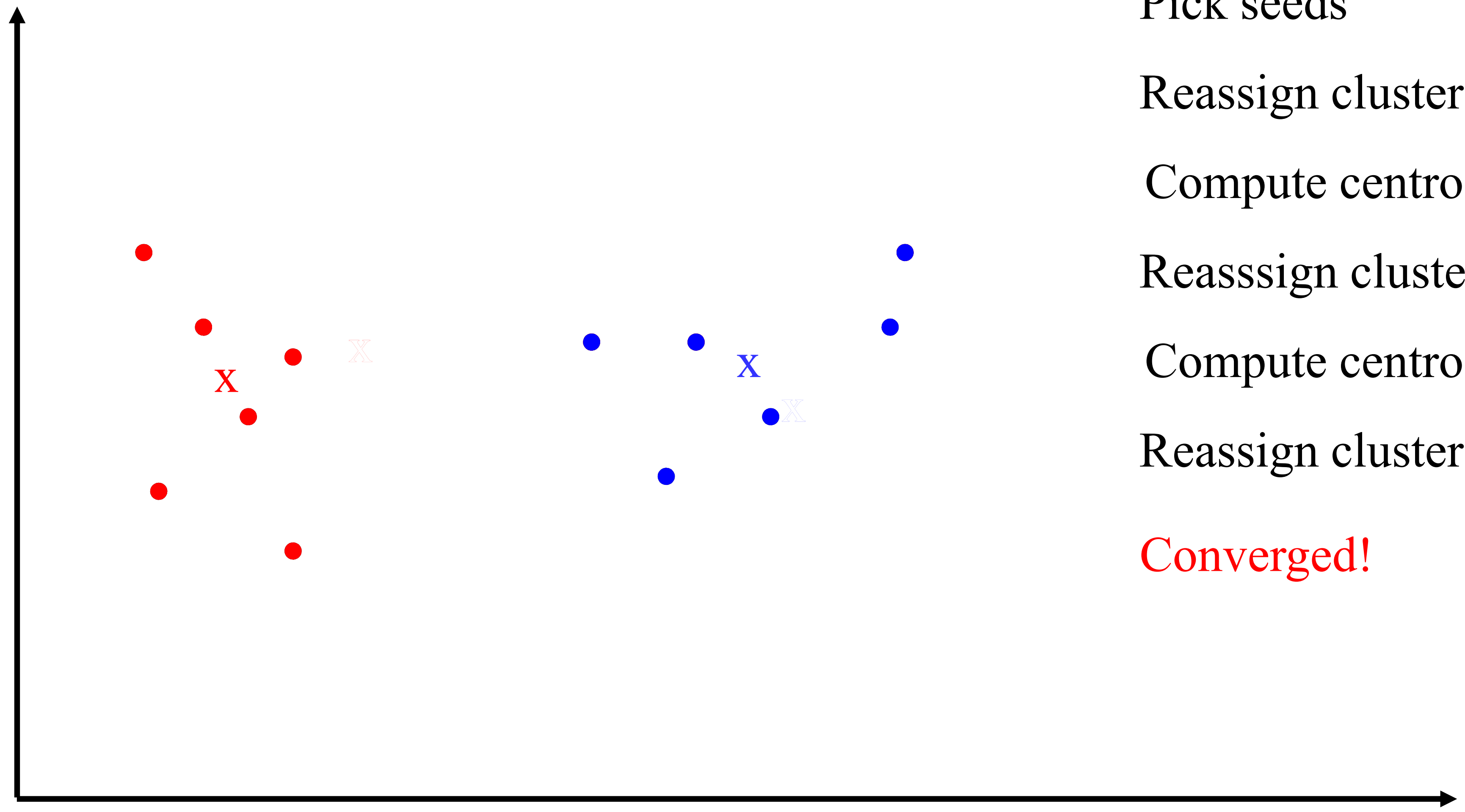
- Cosine Similarity (transform to a distance by subtracting from 1):

$$1 - \frac{\vec{x} \bullet \vec{y}}{|\vec{x}| \cdot |\vec{y}|}$$

K-Means Algorithm

```
Let $d$ be the distance measure between instances.
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 minimal.
    (Update the seeds to the centroid of each cluster)
    For each cluster $c_j$
        $s_j = \mu(c_j)$
```

K Means Example
(K=2)



Pick seeds

Reassign clusters

Compute centroids

Reasssign clusters

Compute centroids

Reassign clusters

Converged!

Time Complexity

- Assume computing distance between two instances is O(m) where m is the dimensionality of the vectors.
- Reassigning clusters: O(kn) distance computations, or O(knm).
- Computing centroids: Each instance vector gets added once to some centroid: O(nm).
- Assume these two steps are each done once for I iterations: O($Iknm$).
- Linear in all relevant factors, assuming a fixed number of iterations, more efficient than O(n^2) HAC.

K-Means Objective

- The objective of k-means is to minimize the total sum of the squared distance of every point to its corresponding cluster centroid.

$$\sum_{l=1}^{K} \sum_{x_i \in X_l} \| x_i - \mu_l \|^2$$

- Finding the global optimum is NP-hard.
- The k-means algorithm is guaranteed to converge a local optimum.

Seed Choice

- Results can vary based on random seed selection.
- Some seeds can result in poor convergence rate, or convergence to sub-optimal clusterings.
- Select good seeds using a heuristic or the results of another method.

Buckshot Algorithm

- Combines HAC and K-Means clustering.
- First randomly take a sample of instances of size $\sqrt{n}$
- Run group-average HAC on this sample, which takes only O(n) time.
- Use the results of HAC as initial seeds for K-means.
- Overall algorithm is O(n) and avoids problems of bad seed selection.

Text Clustering

- HAC and K-Means have been applied to text in a straightforward way.
- Typically use ***normalized***, TF/IDF-weighted vectors and cosine similarity.
- Optimize computations for sparse vectors.
- Applications:
  - During retrieval, add other documents in the same cluster as the initial retrieved documents to improve recall.
  - Clustering of results of retrieval to present more organized results to the user à la Northernlight folders (→).
  - Automated production of hierarchical taxonomies of documents for browsing purposes (à la Yahoo & DMOZ).

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 (probabilities of all categories must sum to 1).

Expectation Maximumization (EM)

- Probabilistic method for soft clustering.
- Direct method that assumes *k* clusters: $\{c_1, c_2, \ldots c_k\}$
- Soft version of *k*-means.
- Assumes a probabilistic model of categories that allows computing $P(c_i \mid E)$ for each category, c_i, for a given example, *E*.
- For text, typically assume a naïve-Bayes category model.
  - Parameters $\theta = \{P(c_i), P(w_j \mid c_i): i \in \{1,\ldots k\}, j \in \{1,\ldots,|V|\}\}$

EM Algorithm

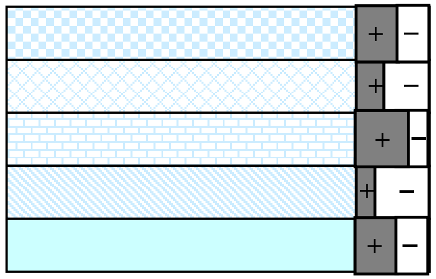
- Iterative method for learning probabilistic categorization model from unsupervised data.
- Initially assume random assignment of examples to categories.
- Learn an initial probabilistic model by estimating model parameters θ from this randomly labeled data.
- Iterate following two steps until convergence:
  - Expectation (E-step): Compute $P(c_i \mid E)$ 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, θ, from the probabilistically re-labeled data.

EM

Initialize:

Assign random probabilistic labels to unlabeled data

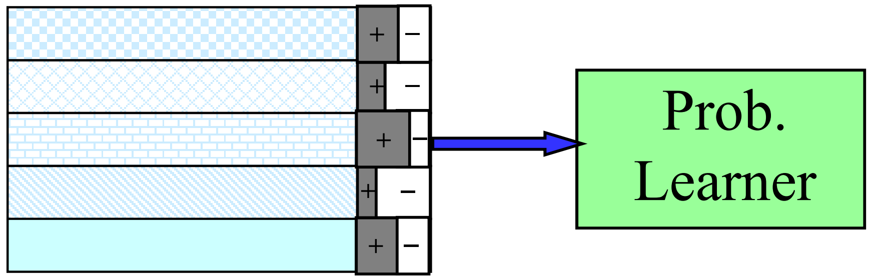
Unlabeled Examples



EM

Initialize:

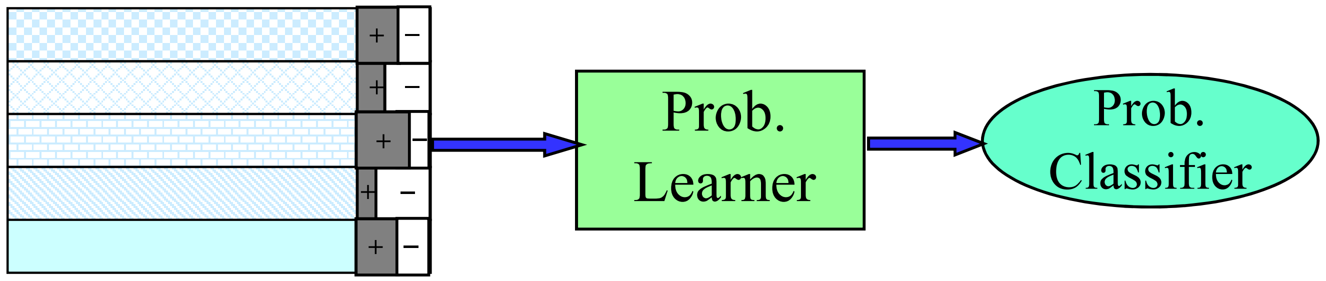
Give soft-labeled training data to a probabilistic learner



EM

Initialize:

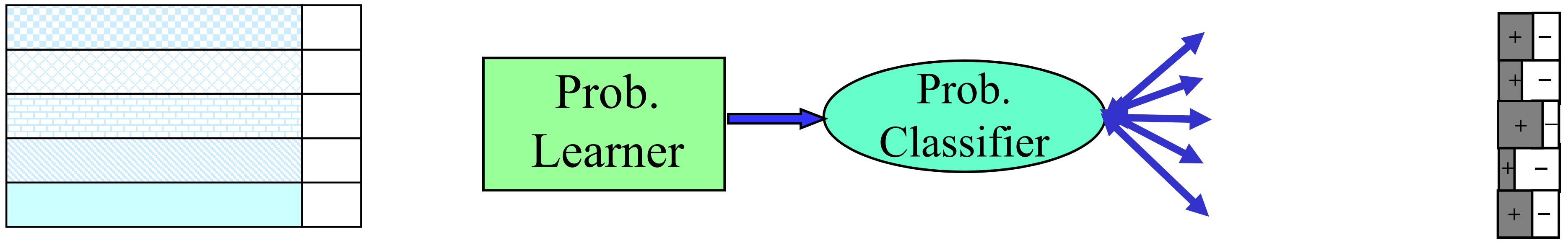
Produce a probabilistic classifier



EM

E Step:

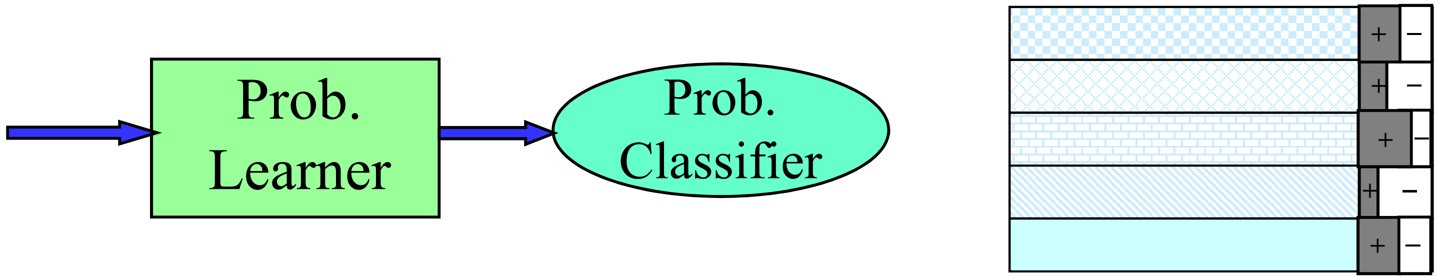
Relabel unlabled data using the trained classifier



EM

M step:

Retrain classifier on relabeled data



Continue EM iterations until probabilistic labels on unlabeled data converge.

Learning from Probabilistically Labeled Data

- Instead of training data labeled with “hard” category labels, training data is labeled with “soft” probabilistic category labels.
- When estimating model parameters θ from training data, weight counts by the corresponding probability of the given category label.
- For example, if $\mathrm{P}(c_1 \mid E) = 0.8$ and $\mathrm{P}(c_2 \mid E) = 0.2$, each word w_j in E 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

Randomly assign examples probabilistic category labels.
Use standard naïve-Bayes training to learn a probabilistic model with parameters θ from the labeled data.
Until convergence or until maximum number of iterations reached:
- E-Step: Use the naïve Bayes model θ to compute $P(c_i \mid E)$ 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 θ using these new probabilistic category labels.

Conclusions

- Unsupervised learning induces categories from unlabeled data.
- Agglomerative vs. Divisive. Hard vs. soft
- There are a variety of approaches, including:
  - HAC
  - k-means
  - EM