Chapter 5
Unsupervised learning
Introduction

• Unsupervised learning
  – Training samples contain only input patterns
    • No desired output is given (teacher-less)
  – Learn to form classes/clusters of sample patterns according to similarities among them
    • Patterns in a cluster would have similar features
    • No prior knowledge as what features are important for classification, and how many classes are there.
K-means Clustering

• Given a data point $v$ and a set of points $X$, define the distance from $v$ to $X$

$$d(v, X)$$

as the (Euclidian) distance from $v$ to the closest point from $X$.

• Given a set of $n$ data points $V = \{v_1...v_n\}$ and a set of $k$ points $X$, define the Squared Error Distortion

$$d(V, X) = \sum d(v_i, X)^2 / n \quad 1 \leq i \leq n$$
**K-Means Clustering**

- **Input**: A set, \( V \), consisting of \( n \) points and a parameter \( k \)
- **Output**: A set \( X \) consisting of \( k \) points (*cluster centers*) that minimizes the squared error distortion \( d(V,X) \) over all possible choices of \( X \)

- **K-means Clustering algorithm**
  1) Pick a number (K) of cluster centers
  2) Assign every data point (e.g., gene) to its nearest cluster center
  3) Move each cluster center to the mean of its assigned data points (e.g., genes)
  4) Repeat 2-3 until convergence
1-Means Clustering Problem: an Easy Case

- **Input**: A set, \( V \), consisting of \( n \) points

- **Output**: A single points \( x \) (cluster center) that minimizes the squared error distortion \( d(V, x) \) over all possible choices of \( x \)
1-Means Clustering Problem: an Easy Case

- **Input:** A set, $V$, consisting of $n$ points

- **Output:** A single points $x$ (cluster center) that minimizes the squared error distortion $d(V,x)$ over all possible choices of $x$

1-Means Clustering problem is easy.

However, it becomes very difficult (NP-complete) for more than one center.

An efficient *heuristic* method for K-Means clustering is the Lloyd algorithm
K-Means Clustering

- Arbitrarily assign the $k$ cluster centers
- while the cluster centers keep changing
- Assign each data point to the cluster $C_i$, corresponding to the closest cluster representative (center) $(1 \leq i \leq k)$
- After the assignment of all data points, compute new cluster representatives according to the center of gravity of each cluster, that is, the new cluster representative is
  \[ \sum v \backslash |C| \] for all $v$ in $C$ for every cluster $C$

This clustering approach may lead to merely a locally optimal clustering.
Finding a Centroid

We use the following equation to find the n dimensional centroid point amid k n dimensional points:

\[
CP(x_1,x_2,\ldots,x_k) = \left( \frac{\sum_{i=1}^{k} x_{1st_i}}{k}, \frac{\sum_{i=1}^{k} x_{2nd_i}}{k}, \ldots, \frac{\sum_{i=1}^{k} x_{nth_i}}{k} \right)
\]

Let’s find the midpoint between 3 2D points, say: (2,4) (5,2) (8,9)

\[
CP = \left( \frac{2 + 5 + 8}{3}, \frac{4 + 2 + 9}{3} \right) = (5,5)
\]
expression in condition 1

expression in condition 2

$v$: Data points

$X$: Cluster centre

Iteration 0
Iteration 2

- Expression in condition 1
- Expression in condition 2

- $x_1$
- $x_2$
- $x_3$
Iteration 3

Expression in condition 1

Expression in condition 2

0
1
2
3
4
5

0
1
2
3
4
5
Example: 4-cluster data and 4 iterations
K-means clustering summary

Advantages
• Genes automatically assigned to clusters

Disadvantages
• Must determine number of clusters before hand (solution: objective function to select optimum number of clusters)
• All genes forced into a cluster (solution: fuzzy c-means clustering and its versions)
• Clustering models can depend on starting locations of cluster centers (solution: multiple clusterings)
K-means clustering:
Optimum number of clusters

\[ \text{obj1} = \frac{1}{C} \sum_{c=1}^{C} \frac{d_{ic}(x_i, v_c)\mu_{ic}}{\sum_{i=1}^{K} \mu_{ic}} \]

\[ \text{obj2} = \frac{1}{C^2} \sum_{c=1}^{C} \sum_{k=1}^{C} d(v_c, v_k) \]

\[ \text{obj3} = \frac{\text{obj1}}{\text{obj2}} \]

\text{obj1} deals with inner space of a cluster, and try to make each cluster as compact as possible.

\text{obj2} is defined to look at relationship between clusters, and tries to make clusters as far from each other as possible. Therefore, higher the value of \text{obj2} the better the separation of the clusters.

\text{obj3} is combination of both \text{obj1} and \text{obj2}
K-means clustering: Optimum number of clusters

- obj1
- obj2
- obj3
K-means clustering in MATLAB

- \texttt{IDX = KMEANS(X, K)} partitions the points in the N-by-P data matrix \( X \) into \( K \) clusters. This partition minimizes the sum, over all clusters, of the within-cluster sums of point-to-cluster-centroid distances. Rows of \( X \) correspond to points, columns correspond to variables. \texttt{KMEANS} returns an N-by-1 vector \( \text{IDX} \) containing the cluster indices of each point. By default, \texttt{KMEANS} uses squared Euclidean distances.

- \texttt{KMEANS} treats NaNs as missing data, and removes any rows of \( X \) that contain NaNs.

- \texttt{[IDX, C] = KMEANS(X, K)} returns the \( K \) cluster centroid locations in the \( K \)-by-\( P \) matrix \( C \).

- \texttt{[IDX, C, SUMD] = KMEANS(X, K)} returns the within-cluster sums of point-to-centroid distances in the 1-by-\( K \) vector \( \text{SUMD} \).

- \texttt{[IDX, C, SUMD, D] = KMEANS(X, K)} returns distances from each point to every centroid in the N-by-\( K \) matrix \( D \).
K-means clustering in MATLAB

- \texttt{[...]} = \texttt{KMEANS(...,'PARAM1',val1,'PARAM2',val2,...)} allows you to specify optional parameter name/value pairs to control the iterative algorithm used by \texttt{KMEANS}.

- **Parameters**: ‘Distance’, 'Start', 'Replicates', 'EmptyAction', 'Display'

- 'Distance' - Distance measure, in P-dimensional space, that \texttt{KMEANS} should minimize with respect to. Choices are:
  - \{'sqEuclidean\}' - Squared Euclidean distance
  - 'cityblock' - Sum of absolute differences, a.k.a. L1
  - 'cosine' - One minus the cosine of the included angle between points (treated as vectors)
  - 'correlation' - One minus the sample correlation between points (treated as sequences of values)
  - 'Hamming' - Percentage of bits that differ (only suitable for binary data)
K-means clustering in MATLAB

- 'Start' - Method used to choose initial cluster centroid positions, sometimes known as "seeds". Choices are:
  - {'sample'} - Select K observations from X at random
  - 'uniform' - Select K points uniformly at random from the range of X. Not valid for Hamming distance.
  - 'cluster' - Perform preliminary clustering phase on random 10% subsample of X. This preliminary phase is itself initialized using 'sample'.
  - matrix - A K-by-P matrix of starting locations. In this case, you can pass in [] for K, and KMEANS infers K from the first dimension of the matrix. You can also supply a 3D array, implying a value for 'Replicates'
  - from the array's third dimension
K-means clustering in MATLAB

- 'Replicates' - Number of times to repeat the clustering, each with a new set of initial centroids [positive integer | {1}]

- 'Maxiter' - The maximum number of iterations [positive integer | {100}]

- 'EmptyAction' - Action to take if a cluster loses all of its member observations. Choices are:
  - {'error'} - Treat an empty cluster as an error
  - 'drop' - Remove any clusters that become empty, and set corresponding values in C and D to NaN.
  - 'singleton' - Create a new cluster consisting of the one observation furthest from its centroid.
K-means clustering in MATLAB

Practical applications - 1
Investigate the followings from MATLAB’s index menu
- “cluster analysis – K-means clustering”
- “clusters – gene expression data”

Note that, in both applications, the K-means was run 5 times with the same number of clusters and resulted in different results due to random initialisation of the cluster centres. The outputs show that, for both problems (simple or complex), non-global minima do exist. Each of these five replicates began from a different randomly selected set of initial centroids. K-means then found two different local minima for the first application whereas five different local minima was identified for the yeast genes. However, the final solution that K-means returns is the one with the lowest total sum of distances, over all replicates.
K-means clustering in MATLAB

Practical applications – 2

• (a) Using K-means clustering tool, cluster the genes into 16 and 25 sub-groups

• (b) Identify a particular gene that can represent each cluster
  • A gene that is the closest to the centre of a particular cluster can be treated as the gene that characterises the cluster and represent the genes that belong to the cluster

• (c) As the K-means clustering method attains initial centre values randomly and therefore is likely to produce different sets of profiles, try Task-(a) with two different initial centre sets and compare the profiles

• Find out if there is any difference between genes that are found in Task-(b) to represent each cluster
You may start with the following small gene set and then work on the yeast data set

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>g1</td>
<td>10</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>g2</td>
<td>10</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>g3</td>
<td>4</td>
<td>8.5</td>
<td>3</td>
</tr>
<tr>
<td>g4</td>
<td>9.5</td>
<td>0.5</td>
<td>8.5</td>
</tr>
<tr>
<td>g5</td>
<td>4.5</td>
<td>8.5</td>
<td>2.5</td>
</tr>
<tr>
<td>g6</td>
<td>10.5</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>g7</td>
<td>5</td>
<td>8.5</td>
<td>11</td>
</tr>
<tr>
<td>g8</td>
<td>2.7</td>
<td>8.7</td>
<td>2</td>
</tr>
<tr>
<td>g9</td>
<td>9.7</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>g10</td>
<td>10.2</td>
<td>1</td>
<td>9.2</td>
</tr>
</tbody>
</table>
Simple Competitive Learning

- Unsupervised learning
- Goal:
  - Learn to form classes/clusters of examplers/sample patterns according to similarities of these examplers.
  - Patterns in a cluster would have similar features
  - No prior knowledge as what features are important for classification, and how many classes are there.
- Architecture:
  - Output nodes: $Y_1, \ldots, Y_m$, representing the m classes
  - They are competitors
  (WTA realized either by an external procedure or by lateral inhibition as in Maxnet)
• **Training:**
  – Train the network such that the weight vector $w_j$ associated with jth output node becomes the representative vector of a class of similar input patterns.
  – Initially all weights are randomly assigned
  – Two phase unsupervised learning

• **competing phase:**
  – apply an input vector $i_l$ randomly chosen from sample set.
  – compute output for all output nodes: $o_j = i_l \cdot w_j$
  – determine the winner $j^*$ among all output nodes (winner is not given in training samples so this is unsupervised)

• **rewarding phase:**
  – the winner is reworded by updating its weights $w_{j^*}$ to be closer to $i_l$ (weights associated with all other output nodes are not updated: kind of WTA)

• **repeat** the two phases many times (and gradually reduce the learning rate) until all weights are stabilized.
• Weight update: $w_j = w_j + \Delta w_j$
  – Method 1:
    $$\Delta w_j = \eta (i_l - w_j)$$
  – Method 2
    $$\Delta w_j = \eta i_l$$

In each method, $w_j$ is moved closer to $i_l$

– Normalize the weight vector to unit length after it is updated
  $$w_j = \frac{w_j}{\|w_j\|}$$

– Sample input vectors are also normalized
  $$i_l = \frac{i_l}{\|i_l\|}$$

– Distance
  $$i_l - w_j = \|i_l - w_j\|_2 = \sqrt{\sum_i (i_{l,i} - w_{j,i})^2}$$
• $w_j$ is moving to the center of a cluster of sample vectors after repeated weight updates
  – Node $j$ wins for three training samples: $i_1$, $i_2$ and $i_3$
  – Initial weight vector $w_j(0)$
  – After successively trained by $i_1$, $i_2$ and $i_3$, the weight vector changes to $w_j(1)$, $w_j(2)$, and $w_j(3)$,
Examples

• A simple example of competitive learning (pp. 168-170)
  – 6 vectors of dimension 3 in 3 classes (6 input nodes, 3 output nodes)
    \begin{align*}
    i_1 &= (1.1, 1.7, 1.8) & i_2 &= (0, 0, 0) \\
    i_3 &= (0, 0.5, 1.5) & i_4 &= (1, 0, 0) \\
    i_5 &= (0.5, 0.5, 0.5) & i_6 &= (1, 1, 1)
  \end{align*}
  – Weight matrices:

  \[
  W(0) = \begin{pmatrix}
  w_A : & 0.2 & 0.7 & 0.3 \\
  w_B : & 0.1 & 0.1 & 0.9 \\
  w_C : & 1 & 1 & 1
  \end{pmatrix}
  \]

  \[
  W(1) = \begin{pmatrix}
  w_A : & 0.2 & 0.7 & 0.3 \\
  w_B : & 0.1 & 0.1 & 0.9 \\
  w_C : & 1.05 & 1.35 & 1.4
  \end{pmatrix}
  \]

  \[
  W(12) = \begin{pmatrix}
  w_A : & 0.55 & 0.3 & 0.3 \\
  w_B : & 0 & 0.4 & 1.35 \\
  w_C : & 1 & 1.2 & 1.25
  \end{pmatrix}
  \]

  Node A: for class \{i_2, i_4, i_5\}
  Node B: for class \{i_3\}
  Node C: for class \{i_1, i_6\}
Comments

1. Ideally, when learning stops, each $w_j$ is close to the centroid of a group/cluster of sample input vectors.

2. To stabilize $w_j$, the learning rate $\eta$ may be reduced slowly toward zero during learning, e.g., $\eta(t + 1) \leq \eta(t)$

3. # of output nodes:
   - too few: several clusters may be combined into one class
   - too many: over classification
   - ART model (later) allows dynamic add/remove output nodes

4. Initial $w_j$:
   - learning results depend on initial weights (node positions)
   - training samples known to be in distinct classes, provided such info is available
   - random (bad choices may cause anomaly)

5. Results also depend on sequence of sample presentation
Example

$w_1$ will always win no matter the sample is from which class

$w_2$ is stuck and will not participate in learning

unstuck:

let output nodes have some *conscience*

temporarily shot off nodes which have had very high winning rate (hard to determine what rate should be considered as “very high”)

Example

Results depend on the sequence of sample presentation

Solution:
Initialize $w_j$ to randomly selected input vectors that are far away from each other
Self-Organizing Maps (SOM) (§ 5.5)

- Competitive learning (Kohonen 1982) is a special case of SOM (Kohonen 1989)
- In competitive learning,
  - the network is trained to organize input vector space into subspaces/classes/clusters
  - each output node corresponds to one class
  - the output nodes are not ordered: random map

- The topological order of the three clusters is 1, 2, 3
- The order of their maps at output nodes are 2, 3, 1
- The map does not preserve the topological order of the training vectors
• **Topographic map**
  
  – a mapping that preserves neighborhood relations between input vectors, (topology preserving or feature preserving).

  – if $i_1$ and $i_2$ are two neighboring input vectors (by some distance metrics),

  • their corresponding winning output nodes (classes), $i$ and $j$ must also be close to each other in some fashion

  – one dimensional: line or ring, node $i$ has neighbors $i \pm 1$ or $i \pm 1 \mod n$

  – two dimensional: grid.

      rectangular: node $(i, j)$ has neighbors:

      $(i, j \pm 1), (i \pm 1, j)$, (or additional $(i \pm 1, j \pm 1)$)

      hexagonal: 6 neighbors
• Biological motivation
  – Mapping two dimensional continuous inputs from sensory organ (eyes, ears, skin, etc) to two dimensional discrete outputs in the nerve system.
    • Retinotopic map: from eye (retina) to the visual cortex.
    • Tonotopic map: from the ear to the auditory cortex
  – These maps preserve topographic orders of input.
  – Biological evidence shows that the connections in these maps are not entirely “pre-programmed” or “pre-wired” at birth. **Learning must occur** after the birth to create the necessary connections for appropriate topographic mapping.
SOM Architecture

• Two layer network:
  – Output layer:
    • Each node represents a class (of inputs)
    • Node function: \( o_j = i_l \cdot w_j = \sum_k w_{j,k} \cdot i_{l,k} \)
    • Neighborhood relation is defined over these nodes
      – \( N_j(t) \): set of nodes within distance \( D(t) \) to node \( j \).
    • Each node cooperates with all its neighbors and competes with all other output nodes.
    • Cooperation and competition of these nodes can be realized by Mexican Hat model
      
      \( D = 0 \): all nodes are competitors (no cooperative)
      \( \rightarrow \) random map
      
      \( D > 0 \): \( \rightarrow \) topology preserving map
Algorithm SelfOrganize;

- Select network topology (neighborhood relation);
- Initialize weights randomly, and select $D(0) > 0$;
- While computational bounds are not exceeded, do
  1. Select an input sample $i_\ell$;
  2. Find the output node $j^\ast$ with minimum
     \[ \sum_{k=1}^{n}(i_{\ell,k}(t) - w_{j,k}(t))^2; \]
  3. Update weights to all nodes within a topological distance of $D(t)$ from $j^\ast$, using
     \[ w_j(t + 1) = w_j(t) + \eta(t)(i_\ell(t) - w_j(t)), \]
     where $0 < \eta(t) \leq \eta(t - 1) \leq 1$;
  4. Increment $t$;
end-while.
Notes

1. Initial weights: small random value from (-e, e)
2. Reduction of $\eta$:
   - Linear: $\eta(t + 1) = \eta(t) - \Delta\eta$
   - Geometric: $\eta(t + 1) = \eta(t) \cdot \beta$ where $0 < \beta < 1$
3. Reduction of $D$: $D(t + \Delta t) = D(t) - 1$ while $D(t) > 0$
   should be much slower than $\eta$ reduction.
   $D$ can be a constant through out the learning.
4. Effect of learning
   For each input $i$, not only the weight vector of winner $j^*$
   is pulled closer to $i$, but also the weights of $j^*$’s close
   neighbors (within the radius of $D$).
5. Eventually, $w_j$ becomes close (similar) to $w_{j \pm 1}$. The classes
   they represent are also similar.
6. May need large initial $D$ in order to establish topological
   order of all nodes
Notes

7. Find \( j^* \) for a given input \( i_l \):
   - With minimum distance between \( w_j \) and \( i_l \).
   - Distance: \( \text{dist}(w_j, i_l) = \|w_j - i_l\|_2 = \sum_{k=1}^{n} (i_{l,k} - w_{j,k})^2 \)
   - Minimizing \( \text{dist}(w_j, i_l) \) can be realized by maximizing
     \[
     o_j = i_l \cdot w_j = \sum_k w_{j,k} \cdot i_{l,k}
     \]
     - \( \text{dist}(w_j, i_l) = -\sum_{k=1}^{n} (i_{l,k}^2 + w_{j,k}^2 - 2i_{l,k} \cdot w_{j,k}) \)
     \[
     = -\sum_{k=1}^{n} i_{l,k}^2 - \sum_{k=1}^{n} w_{j,k}^2 + 2\sum_{k=1}^{n} i_{l,k} \cdot w_{j,k}
     \]
     \[
     = 2\sum_{k=1}^{n} i_{l,k} \cdot w_{j,k} - 2
     \]
     \[
     = 2i_l \cdot w_j - 2
     \]
Figure 5.16: Emergence of an “ordered” map. The dark circles represent data points in one-dimensional space, and nodes are identified as $A$, $B$, $C$, and $D$, with connections indicating linear topology. Figure (a) depicts initial positions of nodes before applying SOM learning algorithm, and Figure (b) shows final positions of nodes, with topologically adjacent nodes having similar weights.
Examples

- A simple example of competitive learning (pp. 172-175)
  - 6 vectors of dimension 3 in 3 classes, node ordering: B – A – C
    \[ i_1 = (1.1, 1.7, 1.8) \quad i_2 = (0, 0, 0) \]
    \[ i_3 = (0, 0.5, 1.5) \quad i_4 = (1, 0, 0) \]
    \[ i_5 = (0.5, 0.5, 0.5) \quad i_6 = (1, 1, 1) \]
  - Initialization: \( \eta = 0.5 \), weight matrix: \( W(0) = \begin{bmatrix} w_A : 0.2 & 0.7 & 0.3 \\ w_B : 0.1 & 0.1 & 0.9 \\ w_C : 1 & 1 & 1 \end{bmatrix} \)
  - \( D(t) = 1 \) for the first epoch, = 0 afterwards
  - Training with \( i_1 = (1.1, 1.7, 1.8) \)
    determine winner: squared Euclidean distance between \( i_1 \) and \( w_j \)
    \[
    d^2_{A,1} = (1.1 - 0.2)^2 + (1.7 - 0.7)^2 + (1.8 - 0.3)^2 = 4.1
    \]
    \[
    d^2_{B,1} = 4.4, \quad d^2_{C,1} = 1.1
    \]
  - C wins, since \( D(t) = 1 \), weights of node C and its neighbor A are updated, but not \( w_B \)
Examples

\[ W(0) = \begin{bmatrix} w_A : 0.2 & 0.7 & 0.3 \\ w_B : 0.1 & 0.1 & 0.9 \\ w_C : 1 & 1 & 1 \end{bmatrix} \]

\[ W(1) = \begin{bmatrix} w_A : 0.65 & 1.2 & 1.05 \\ w_B : 0.1 & 0.1 & 0.9 \\ w_C : 1.05 & 1.35 & 1.4 \end{bmatrix} \]

\[ W(15) = \begin{bmatrix} w_A : 0.83 & 0.77 & 0.81 \\ w_B : 0.47 & 0.23 & 0.30 \\ w_C : 0.61 & 0.95 & 1.34 \end{bmatrix} \]

- Observations:
  - Distance between weights of non-neighboring nodes (B, C) increase
  - Input vectors switch allegiance between nodes, especially in the early stage of training

<table>
<thead>
<tr>
<th>( t )</th>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 6</td>
<td>(3, 5)</td>
<td>(2, 4)</td>
<td>(1, 6)</td>
</tr>
<tr>
<td>7 – 12</td>
<td>(6)</td>
<td>(2, 4, 5)</td>
<td>(1, 3)</td>
</tr>
<tr>
<td>13 – 16</td>
<td>(6)</td>
<td>(2, 4, 5)</td>
<td>(1, 3)</td>
</tr>
</tbody>
</table>
How to illustrate Kohonen map (for 2 dimensional patterns)

- Input vector: 2 dimensional
  Output vector: 1 dimensional line/ring or 2 dimensional grid.
  Weight vector is also 2 dimensional
- Represent the topology of output nodes by points on a 2 dimensional plane. Plotting each output node on the plane with its weight vector as its coordinates.
- Connecting neighboring output nodes by a line
  output nodes: (1, 1)  (2, 1)  (1, 2)
  weight vectors: (0.5, 0.5) (0.7, 0.2) (0.9, 0.9)
Illustration examples

- Input vectors are uniformly distributed in the region, and randomly drawn from the region.
- Weight vectors are initially drawn from the same region randomly (not necessarily uniformly).
- Weight vectors become ordered according to the given topology (neighborhood), at the end of training.
Similarity Measures in Data Analysis

• **General Assumptions**
  – Each data item is a tuple (vector)
  – Values of tuples are nominal, ordinal or numerical
  – Similarity = (Distance)$^{-1}$

• **Pure Numerical Tuples**
  – $\text{Sim}(d_i,d_j) = \sum d_{i,k}d_{j,k}$
  – $\text{sim } (d_i,d_j) = \cos(d_i,d_j)$
  – …and many more (slide after next)
Similarity Measures in Data Analysis

• For Ordinal Values
  – E.g. "small," "medium," "large," "X-large"
  – Convert to numerical assuming constant Δ... on a normalized [0,1] scale, where: max(v)=1, min(v)=0, others interpolate
  – E.g. "small"=0, "medium"=0.33, etc.
  – Then, use numerical similarity measures
  – Or, use similarity matrix (see next slide)
## Similarity Matrix

<table>
<thead>
<tr>
<th></th>
<th>tiny</th>
<th>little</th>
<th>small</th>
<th>medium</th>
<th>large</th>
<th>huge</th>
</tr>
</thead>
<tbody>
<tr>
<td>tiny</td>
<td>1.0</td>
<td>0.8</td>
<td>0.7</td>
<td>0.5</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>little</td>
<td>1.0</td>
<td>1.0</td>
<td>0.9</td>
<td>0.7</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>small</td>
<td>1.0</td>
<td>1.0</td>
<td>0.7</td>
<td>0.3</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>medium</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.5</td>
<td></td>
<td>0.3</td>
</tr>
<tr>
<td>large</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>huge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
</tbody>
</table>

- Diagonal must be 1.0
- Monotonicity property must hold
- No linearity (value interpolation) assumed
- Qualitative Transitive property must hold
Document Clustering Techniques

- Similarity or Distance Measure: Alternative Choices
  - Cosine similarity
    \[ \cos(x, y) = \frac{\sum_i x_i y_i}{|x||y|} \]
  - Euclidean distance
    \[ d(x, y) = \sqrt{\sum_i (x_i - y_i)^2} \]
  - Kernel functions, e.g.,
    \[ K(d(x, y)) = e^{-d(x, y)^2/2h^2} \]
  - Language Modeling \( P(y|\text{model}_x) \) where \( x \) and \( y \) are documents
Document Clustering Techniques

- Kullback Leibler distance ("relative entropy")

\[
D(p||q) \overset{\text{def}}{=} \sum_{i} p_i \log \frac{p_i}{q_i} = \sum_{i} p_i \log p_i - \sum_{i} p_i \log q_i = -H(p) + H(p,q)
\]

- $\vec{p}$ is the probability distribution of terms in one document;
- $\vec{q}$ is the probability distributions of terms in the other document;
- $p_i$ or $q_i$ is the within-document relative frequency of the $i$th term in the vocabulary.
Principal Component Analysis (PCA) Networks (§ 5.8)

- PCA: a statistical procedure
  - Reduce dimensionality of input vectors
    - Too many features, some of them are dependent of others
    - Extract important (new) features of data which are functions of original features
    - Minimize information loss in the process
  - This is done by forming new interesting features
    - As linear combinations of original features (first order of approximation)
    - New features are required to be linearly independent (to avoid redundancy)
    - New features are desired to be different from each other as much as possible (maximum variability)
Linear Algebra

- Two vectors $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$ are said to be **orthogonal** to each other if
  $$x \cdot y = \sum_{i=1}^{n} x_i y_i = 0.$$ 

- A set of vectors $x^{(1)}, ..., x^{(k)}$ of dimension $n$ are said to be **linearly independent** of each other if there does not exist a set of real numbers $a_1, ..., a_k$ which are not all zero such that
  $$a_1 x^{(1)} + \cdots + a_k x^{(k)} = 0$$
  otherwise, these vectors are linearly dependent and each one can be expressed as a **linear combination** of the others

$$x^{(i)} = -\frac{a_1}{a_i} x^{(1)} - \cdots - \frac{a_k}{a_i} x^{(k)} = \sum_{j \neq i} \frac{a_j}{a_i} x^{(j)}$$
• Vector $x$ is an **eigenvector** of matrix $A$ if there exists a constant $\gamma \neq 0$ such that $Ax = \gamma x$
  - $\gamma$ is called a **eigenvalue** of $A$ (wrt $x$)
  - A matrix $A$ may have more than one eigenvectors, each with its own eigenvalue
  - Eigenvectors of a matrix corresponding to distinct eigenvalues are linearly independent of each other

• **Matrix $B$** is called the **inverse** matrix of matrix $A$ if $AB = 1$
  - $1$ is the identity matrix
  - Denote $B$ as $A^{-1}$
  - Not every matrix has inverse (e.g., when one of the row/column can be expressed as a linear combination of other rows/columns)

• Every matrix $A$ has a unique pseudo-inverse $A^*$, which satisfies the following properties
  $$AA^*A = A; \quad A^*AA^* = A^*; \quad A^*A = (A^*A)^T; \quad AA^* = (AA^*)^T$$
Example of PCA: 3-dim $x$ is transformed to 2-dim $y$

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a & b & c \\ p & q & r \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} ax_1 + bx_2 + cx_3 \\ px_1 + qx_2 + rx_3 \end{bmatrix}$$

If rows of $W$ have unit length and are orthogonal (e.g., $w_1 \cdot w_2 = ap + bq + cr = 0$), then $W^T$ is a pseudo-inverse of $W$
• Generalization
  – Transform \( n \)-dim \( x \) to \( m \)-dem \( y \) \((m < n)\), the pseudo-inverse matrix \( W \) is a \( m \times n \) matrix
  – Transformation: \( y = Wx \)
  – Opposite transformation: \( x' = W^Ty = W^TWx \)
  – If \( W \) minimizes “information loss” in the transformation, then
    \[ ||x - x'|| = ||x - W^TWx|| \] should also be minimized
  – If \( W^T \) is the pseudo-inverse of \( W \), then \( x' = x \): perfect transformation (no information loss)

• How to find such a \( W \) for a given set of input vectors
  – Let \( T = \{x_1, \ldots, x_k\} \) be a set of input vectors
  – Making them zero-mean vectors by subtracting the mean vector \((\sum x_i) / k\) from each \( x_i \).
  – Compute the correlation matrix \( S(T) \) of these zero-mean vectors, which is a \( n \times n \) matrix (book calls covariance-variance matrix)
- Find the $m$ eigenvectors of $S(T)$: $w_1, \ldots, w_m$ corresponding to $m$ largest eigenvalues $\gamma_1, \ldots, \gamma_m$
- $w_1, \ldots, w_m$ are the first $m$ principal components of $T$
- $W = (w_1, \ldots, w_m)$ is the transformation matrix we are looking for
- $m$ new features extract from transformation with $W$ would be linearly independent and have maximum variability
- This is based on the following mathematical result:

Let $b$ be any vector such that $\|b\| = 1$. Then the variance of $b \cdot x$, where $x \in T$, is maximized when $b$ is chosen to be the eigen-vector of $S(T)$ that corresponds to the largest magnitude eigen-value of $S(T)$. 
Example

$$T = \{(1.3, 3.2, 3.7), (1.4, 2.8, 4.1),
(1.5, 3.1, 4.6), (1.2, 2.9, 4.8), (1.1, 3.0, 4.8)\}.$$  
mean vector $(1.3, 3.0, 4.4)$

$$x_1 = (0.0, 0.2, -0.7), x_2 = (0.1, -0.2, -0.3), \ldots,$$

covariance matrix  
$$S = \frac{1}{5} \begin{pmatrix} 0.10 & 0.01 & -0.11 \\
0.01 & 0.10 & -0.10 \\
-0.11 & -0.10 & 0.94 \end{pmatrix}$$
eigen-values eigen-vectors

$$\gamma_1 = 0.965, (-0.823, -0.542, -0.169)$$
$$\gamma_2 = 0.090, (0.553, -0.832, -0.026)$$
$$\gamma_3 = 0.084, (-0.126, -0.115, 0.985)$$
For $m = 1$, we consider the largest magnitude eigenvalue $0.965$ and this gives $W = W_1$ where

$$W_1 = (-0.823 \ - 0.542 \ - 0.169).$$

Original 3 dimensional vectors transformed into 1-dimensional

$$y_1 = W_1 x_1 = (-0.823, -0.541, -0.169)^T (0, 0.2, -0.7) = 0.101$$

$$y_2 = W_1 x_2 = 0.0677$$

For $m = 2$, we use both $\gamma_1$ and $\gamma_2$:

$$W = W_2 = \begin{pmatrix}
-0.823 & -0.542 & -0.169 \\
0.553 & -0.832 & -0.026
\end{pmatrix}.$$  

Original 3 dimensional vectors transformed into 2-dimensional

$$y_1 = W_2 x_1 = \begin{pmatrix} 0.1099 \\ -0.1462 \end{pmatrix}$$

$$y_2 = W_2 x_2 = \begin{pmatrix} 0.0677 \\ 0.2295 \end{pmatrix}$$
Note that $\gamma_1/(\gamma_1 + \gamma_2 + \gamma_3) = 0.965/1.139 = 0.84$, implying that about 84% of the variation in the training set can be explained by a single eigen-vector, using $W_1$ alone. To capture more than 84% of the input characteristics, we may map input vectors into a two-dimensional space, with $y = W_2 x$. 
\begin{itemize}

\item PCA network architecture

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{PCA_network_diagram.png}
\end{figure}

\end{itemize}

- Train \( W \) so that it can transform sample input vector \( x_l \) from n-dim to m-dim output vector \( y_l \).

- Transformation should minimize information loss:

Find \( W \) which minimizes

\[
\sum_l ||x_l - x_l'|| = \sum_l ||x_l - W^T W x_l|| = \sum_l ||x_l - W^T y_l||
\]

where \( x_l' \) is the “opposite” transformation of \( y_l = W x_l \) via \( W^T \)
• Training $W$ for PCA net

  – Unsupervised learning:
    only depends on input samples $x_l$
  – Error driven: $\Delta W$ depends on $||x_l - x_l'|| = ||x_l - W^T W x_l||$
  – Start with randomly selected weight, change $W$ according to

\[
\Delta W = \eta \left( y_l x_l^T - K_\ell W \right) \text{where } K_\ell = y_\ell y_\ell^T
\]

  – This is only one of a number of suggestions for $K_b$ (Williams)
  – Weight update rule becomes

\[
\Delta W = \eta_l (y_l x_l^T - y_l y_l^T W) = \eta_l y_l (x_l^T - y_l^T W) = \eta_l y_l (x_l^T - W^T y_l)
\]
• Example (sample sample inputs as in previous example)

\[ \eta = 1.0, \text{ and initial weights } (0.3, 0.4, 0.5) \]

For the first input, \[ y = (0.3, 0.4, 0.5) \cdot (0, 0.2, -0.7) = -0.27 \]

\[ \Delta W = (-0.27(0.00, 0.20, -0.70) - (-0.27)^2 (0.30, 0.40, 0.50)) \]

\[ W = (0.30, 0.40, 0.50) + \Delta W = (0.28, 0.32, 0.65) \]

For the next input \((x_2), y = -0.23, \text{ and} \]

\[ W = (0.278, 0.316, 0.652) + \Delta W = (0.240, 0.346, 0.687) \]

Subsequent presentations of \(x_3, x_4, \text{ and } x_5\) change \(W\) to

\[
(0.272, 0.351, 0.697) \quad \text{After } x_3 \\
(0.238, 0.313, 0.751) \quad \text{After } x_4 \\
(0.172, 0.293, 0.804) \quad \text{After } x_5 \\
(-0.008, 0.105, 0.989) \quad \text{After second epoch} \\
(-0.111, -0.028, 1.004) \quad \text{After second epoch}
\]

Eventually converging to 1st PC \((-0.823 \ -0.542 \ -0.169)\)
• Notes
  – PCA net **approximates** principal components (error may exist)
  – It obtains PC by learning, without using statistical methods
  – Forced stabilization by gradually reducing $\eta$
  – Some suggestions to improve learning results.
    • instead of using identity function for output $y = Wx$, using non-linear function $S$, then try to minimize
      \[ \mathcal{E} \{ \| x - W^T S(Wx) \|^2 \} \]
    • If $S$ is differentiable, use gradient descent approach
    • For example: $S$ be monotonically increasing odd function $S(-x) = -S(x)$ (e.g., $S(x) = x^3$)