Unsupervised Learning and Clustering

Graph-Theoretic Clustering
Decision Trees
Graph-Theoretic Clustering

- **Graph: (S,R)**
  - S: Set of nodes
  - R: Set of edges, \( R \subseteq S \times S \)

- **Clique:** Set of nodes that are all connected to each other,
  - i.e., \( \{ P \subseteq S \mid P \times P \subseteq R \} \)

- **Goal:** Find clusters in a graph that are not as dense as cliques but are compact enough within user specified thresholds.
Graph-Theoretic Clustering
Graph-Theoretic Clustering

- $(X, Y) \in R$ means $Y$ is a neighbor of $X$,
  \[ \text{Neighborhood}(X) = \{ Y \mid (X, Y) \in R \} \]

- **Conditional density** $D(Y \mid X)$ is the number of nodes in the neighborhood of $X$ which have $Y$ as a neighbor
  \[
  D(Y|X) = \# \{ N \in S \mid (N, Y) \in R \text{ and } (X, N) \in R \} \\
  = D(X|Y) \\
  = \# \{ \text{Neighborhood}(X) \cap \text{Neighborhood}(Y) \} 
  \]
Graph-Theoretic Clustering

- Given an integer $K$, a dense region $Z$ around a node $X \in S$ is defined as

$$Z(X, K) = \{Y \in S \mid D(Y|X) \geq K\}$$

- $Z(X) = Z(X, J)$ is a dense region candidate around $X$ where

$$J = \max\{K \mid \#Z(X, K) \geq K\}$$
Graph-Theoretic Clustering

**Association** of a node $X$ to a subset $B$ of $S$ is

$$A(X|B) = \frac{\#\{\text{Neighborhood}(X) \cap B\}}{\#B}$$

where $0 \leq A(X|B) \leq 1$.

**Compactness** of a subset $B$ of $S$ is

$$C(B) = \frac{1}{\#B} \sum_{X \in B} A(X|B)$$

where $0 \leq C(B) \leq 1$. 
Graph-Theoretic Clustering

A dense region $B$ of the graph $(S, R)$ should satisfy

1. $B = \{ N \in Z(X) \mid A(N|Z(X)) \geq \tau_a \}$ for some $X \in S,$
2. $C(B) \geq \tau_c,$
3. $\#B \geq \tau_s$

where $\tau_a$, $\tau_c$ and $\tau_s$ are thresholds supplied by the user for minimum association, minimum compactness, and minimum size, respectively.
Graph-Theoretic Clustering

Algorithm for finding a dense region around a node $X$:

1. Compute $D(Y|X)$ for every other node $Y$ in $S$.
2. Find a dense region candidate $Z(X, K')$ where

   $$K' = \max\{K \mid \#\{Y \mid D(Y|X) \geq K\} \geq K\}.$$

3. Remove the nodes with a low association from the candidate set. Iterate until all of the nodes have high enough association.
4. Check whether the remaining nodes have high enough average association.
5. Check whether the candidate set is large enough.
Given the dense regions, the algorithm for graph-theoretic clustering proceeds as follows:

1. Define the **dense-region relation** $F$ as

   $$F = \{(B_1, B_2) \mid B_1, B_2 \text{ are dense regions of } R, \quad \frac{\#B_1 \cap B_2}{\#B_1} \geq \tau_o \text{ or } \frac{\#B_1 \cap B_2}{\#B_2} \geq \tau_o \}$$

   where $\tau_o$ is a threshold supplied by the user for minimum overlap.

2. Merge the regions that have enough overlap if all of the nodes in the set resulting after merging have high enough associations.

3. Iterate until no regions can be merged.
Graph-Theoretic Clustering

Clusters found in the example graph using the thresholds
\[ \tau_a = 0.5, \tau_c = 0.6, \tau_s = 3, \tau_o = 0.9: \{1, 2, 3, 4, 6\} \text{ (compactness}=0.92), \{7, 8, 9, 10\} \text{ (compactness}=1.00), \{2, 5, 6, 7, 10\} \text{ (compactness}=0.68). \]
Graph-Theoretic Clustering
Graph-Theoretic Clustering

Clusters found in the second example graph using the thresholds $\tau_a = 0.5, \tau_c = 0.8, \tau_s = 3, \tau_o = 0.75$: \{1, 2, 3, 4, 6, 8\} (compactness=0.78), \{2, 4, 5, 8\} (compactness=0.88), \{5, 7, 9, 10\} (compactness=1.00).
Decision Trees

Most pattern recognition methods address problems where there is a natural measure of distance between feature vectors.

What happens when the classification problem involves nominal (categorical) data, e.g., descriptions that are discrete and without any natural notion of similarity or even ordering?

A common representation for this kind of data is a list of attributes (instead of a vector of real numbers).
Decision Trees

It is natural and intuitive to classify a pattern through a sequence of questions, in which the next question asked depends on the answer to the current question. Such a sequence of questions can be represented as a decision tree.

In a decision tree, the top node is called the root node, and is connected by directional links (branches) to other nodes. These nodes are similarly connected until terminal (leaf) nodes, which have no further links, are reached.
Decision Trees

The classification of a particular pattern begins at the root node, which checks for the value of a particular attribute. Different branches from the root node correspond to different values of this attribute that are mutually distinct and exhaustive.

Based on the particular value of that attribute for that pattern, the appropriate branch is followed to a subsequent node.

Similar decisions are made in subsequent nodes until a leaf node is reached.

Each leaf node bears a class label and the pattern is assigned the label of the leaf node reached.
Decision Trees

- **Root**: Color?
  - **Green**
    - Size?
      - Big: Watermelon
      - Medium: Apple
      - Small: Grape
  - **Yellow**
    - Shape?
      - Round
  - **Red**
    - Size?
      - Big: Grapefruit
      - Small: Lemon

- **Level 1**:
  - Right branch: Size?
    - Medium: Apple
      - Taste?
        - Sweet: Cherry
        - Sour: Grape
Decision Trees

Learning a decision tree is based on partitioning the set of training examples into smaller and smaller subsets where each subset is as pure as possible.

Purity for a particular subset is measured according to the number of training samples in that subset having the same class label.

Different criteria can grow trees differently.
Decision Trees

Decision tree learning is a recursive process where the following questions arise:

- Are the attributes numerical or nominal?
- Are the nominal attributes binary-valued or multi-valued?
- How many branches will there be at a node?
- Which attribute should be tested at a node?
- When should a node be declared as a leaf?
- If a tree becomes too large, how can it be simplified?
- How can a class label be assigned to a leaf node?
- How should missing data be handled?
Decision Trees

Each decision outcome at a node is called a **split**.

Each tree can be represented using just binary splits (called a **binary tree**).

For numerical attributes, splitting questions have the form “is $x \leq x_0$”?  

For nonnumerical (nominal) attributes, splitting questions have the form “is $x \in A$” where $A$ is a subset of the possible values of $x$. 
Decision Trees

For a given node, the particular splitting attribute and the corresponding question can be chosen using a search according to the purity (or impurity) measures that are based on entropy, variance, Gini or misclassification criteria. Commonly used criteria about deciding when to stop splitting include thresholds on impurity or number of examples remaining at a node, or statistical tests on the significance of reduction in impurity.
Decision Trees

Alternatively, a tree can be grown fully, and then can be pruned by considering the leaf nodes or even subtrees for elimination or merging.

Once the leaf nodes are finalized, they can be labeled by the class that has the most patterns represented in the corresponding nodes.
Decision Trees

An interesting problem that can arise during training, classification or both is *missing data*. There are several possible reasons for a value to be missing, such as:

- it was not measured,
- there was an instrument malfunction,
- the attribute does not apply,
- or the attribute’s value cannot be known.
Decision Trees

Decision trees can handle missing data by using the primary decision attribute at a node whenever possible, and use alternative attributes when a pattern is missing the primary attribute.

These alternative attributes are called *surrogate splits* and are found by maximizing the probability of making the same decision as the primary split.
In summary, tree-based tools offer the following advantages:

- They can operate on both numerical and nominal measurements.
- They do not require any assumptions about neither the distributions nor the independence of attribute values.
- They are often easy to interpret by creating subgroups of data which the user may graphically analyze.
Assignment

Assume four characters A, D, M and Z are given as 8x8 binary matrices. Using k-means algorithm develop a clustering system for identifying a given character.

As the first step, use the whole matrix as a feature vector.

Modify your algorithm and the number of black pixels at each column as your feature vector. Compare your results for both cases