Decision Trees (New Topic)

Game of Twenty Questions
Apply a series of tests to the input pattern

Test 1 \( \begin{array}{c} \text{Yes} \\ \text{No} \end{array} \)

Tree \( \text{Root} \)

Notation:
Set of Classified Data \( \{ (x_a, w_a) : a \in \Lambda \} \)
Set of Tests \( \{ T_j : j \in \Phi \} \)
Each test has response "T" or "F"

\( T_j (x_a) \in \{ T, F \} \)

Tree nodes \( \{ \mu \} \)
Root node \( \mu_0 \) at top of tree.
Each node either has two child nodes, or is a leaf node.

Each node \( \mu_i \) has a test \( T_{\mu_i} \). Its child node \( \mu_{i1} \) is for data \( x_a \) st. \( T_{\mu_i} (x_a) = T \)
\( \mu_{i2} \) is for data \( x_a \) st. \( T_{\mu_i} (x_a) = F \).
Decision Tree Notation

Define the data that gets to node \( \mu \) recursively.

\[ \Lambda \mu_1 = \{ x_a \in \Lambda \mu \text{ s.t. } T_\mu(x_a) = T \} \]
\[ \Lambda \mu_2 = \{ x_a \in \Lambda \mu \text{ s.t. } T_\mu(x_a) = F \} \]

The root node contains all data \( \Lambda \mu_0 = \Lambda \).

\* Distribution of data at node \( \mu \).

\[ P_\mu(\omega_j) = \frac{1}{|\Lambda \mu|} \sum_{a \in \Lambda \mu} S_{\omega_a} \omega_j \]

\[ \sum_{j=1}^{M} P_\mu(\omega_j) = 1 \quad \text{M is no. classes} \]

\* Define an impurity measure (Entropy) for node \( \mu \)

\[ I(\mu) = \sum_{j} P_\mu(\omega_j) \log P_\mu(\omega_j) \]

Note: if a node is pure, then all data in it belongs to one class.

Intuition: Design a tree so that the leaf nodes are pure — yield good classification.
Heritage Design.

- Initialize tree with the root node only. (so it is a leaf node).
- For all leaf nodes, calculate the maximal decrease in impurity by searching over all the tests.
- Expand the leaf node with maximal decrease and add its child nodes to the tree.

Decrease in impurity at node $\mu$ due to test $T_j$:

$$\Delta I_{\mu_j} = \langle x \in \Lambda_\mu : s.t. \ T_\mu(x) = T \rangle \setminus \langle x \in \Lambda_\mu : s.t. \ T_\mu(x) = F \rangle.$$ 

Decrease in entropy $\Delta I(\mu) = I(\mu) - I(\mu_1, \mu_2)$.

where $I(\mu_1, \mu_2) = \frac{|\Lambda_1|}{|\Lambda_\mu|} I(\mu_1) + \frac{|\Lambda_2|}{|\Lambda_\mu|} I(\mu_2)$.

Hence, for all leaf node $\mu$ calculate $\max_j I_{\mu_1, \mu_2}$, select the leaf node $\mu$ and test $T_j$ which achieve this maximum.
Greedy Strategy

- Start at root node $x_0$.
- Expand root node with test that maximizes the decrease in impurity — or maximizes the gain in purity.
- Repeat with leaf nodes until each leaf node is pure.

Time Complexity: Learning algorithm is $O (1 |\phi| |Y| (|Y|/\log |Y|) \log |Y|) = \text{no. of tests}$.

Run Time $O(\log |Y|)$: Very Rapid

Notes: The design strategy is very greedy. There may be a shorter tree if you learn the tree by searching over a sequence of tests.

The number of children (2) is arbitrary. You can extend the approach to having three or more children.
There are alternative impurity measures (e.g., the Gini index):\[
I(\mu) = \sum_{i,j \in \{1, \ldots, m\}} P(\omega_i|\omega_j) P(\omega_j) = 1 - \sum_{j} P^2(\omega_j).
\]

Expanding the tree until all nodes are pure risks overgeneralizing. It will give perfect performance on the training dataset, but will usually cause errors on the test dataset.

Better to stop splitting the data when the impurity reaches a positive threshold, i.e. set a node to be a leaf if \( I(\mu) \leq \beta \) - threshold.

Then at each leaf, classify data by majority vote.

**Cross Validation Strategy:** learn the decision tree with different impurity thresholds \( \beta \). Select the tree, and hence the \( \beta \), which has best validation (consistency between training \& test datasets).
Spectral Clustering

Undirected graph \( G = (V, E) \) \( V = \{v_1, \ldots, v_n\} \)
weighted edges \( w_{ij} = w_{ji} \geq 0 \) big \( w_{ij} \) means big similarly
Degree of a vertex \( v_i \): \( d_i = \sum_{j=1}^{n} w_{ij} \)
\( D = \text{diag}(d_i) \) degree matrix

For two disjoint subsets \( A \) \( B \)
\[ W(A, B) = \sum_{i \in A, j \in B} w_{ij} \]

Size of a subset \( A \) \( C \) \( U \):
\[ |A| = \text{number of vertices in } A \text{ (un-weighted)} \]
\[ \text{Vol}(A) = \sum_{i \in A} d_i \text{ (weighted volume)} \]

Similarities to Graphs \((x_1, \ldots, x_n)\) data points with similarities \( s_{ij} \geq 0 \)
\( \epsilon \)-neighborhood graph: \( w_{ij} = \begin{cases} 1, & \text{if } s_{ij} > \epsilon \text{ threshold} \\ 0, & \text{otherwise} \end{cases} \)
\( k \)-nn: \( w_{ij} = \begin{cases} s_{ij}, & \text{if } v_i \text{ is a } k\text{-nn of } v_j \text{ or vice versa} \\ 0, & \text{otherwise} \end{cases} \)
fully connected graph: \( w_{ij} = s_{ij} \)

Example (Shi & Malik): \( \mathbb{V} \) set of pixels in an image
\[ W_{ij} = \begin{cases} e^{-\frac{||x_i - x_j||^2}{\sigma^2}}, & \text{if } ||x_i - x_j|| < \sigma \\ 0, & \text{otherwise} \end{cases} \]

Graph Laplacian matrix

\[ L = D - W \]
why “laplacian” \[ \frac{1}{4} \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \]

Laplacian operator -\( \nabla^2 u = -(u_{xx} + u_{yy}) \)

Properties of \( L \) \((\text{for } A)\):
(1) \( f^\top L f = \frac{1}{2} \sum_{i=1}^{n} w_{ii} f_i^2 + \sum_{i < j} w_{ij} f_i f_j \)
(2) \( L \) is symmetric, positive semi-definite
(3) Smallest eigenvalue is 0, eigenvalue \( 1 \)
(4) \( 0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \)
(5) If \( 0 = \lambda_1 = \cdots = \lambda_k \leq \ldots \leq \lambda_n \), then \( G \) has \( k \)-connected components and \( V_1, \ldots, V_k \) are eigenvectors

\[ \lambda_i = \left( \begin{array}{c} a_1 \\ \vdots \\ a_k \end{array} \right) \]

Spring 2013

Note Title
11/23/2011
Normalized Graph Laplacian

\[ L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2} \]
\[ L_{rw} = D^{-1} L = I - D^{-1} W. \]

Properties:
1. \( f^T L_{sym} f = \frac{1}{2} \sum_{i<j} w_{ij} \left( \frac{f_i - f_j}{\sqrt{d_i} \sqrt{d_j}} \right)^2 \)
2. \( (\lambda, u) \) eigenpair of \( L_{rw} \) \( (\lambda, w) \) eigenpair of \( L_{sym} \)
3. \( (\lambda, u) \) eigenpair of \( L_{rw} \): \( L u = \lambda D u \) (generalized eigenvalue)
4. \( (0, u) \) eigenpair of \( L_{rw} \): \( (0, D^{1/2} u) \) eigenpair of \( L_{sym} \)
5. \( L_{sym}, L_{rw} \) are positive semi-def and \( 0 \leq \lambda \leq \ldots \leq \lambda_n \)
6. If \( 0 = \lambda_1 < \lambda_2 < \ldots < \lambda_n \), then \( G \) has \( k \)-connected components.

The eigenspace of \( \lambda = 0 \) is spanned by
\[ \Lambda \]
\[ \Lambda_k \]

Input: Similarly matrix \( S \in \mathbb{R}^{n \times n} \), number of clusters \( k \)

1. Construct \( G = (V, E) \) and \( W \) from \( S \).
2. Compute the un-normalized laplacian \( L = D - W \)
3. Compute the first \( k \) eigenvectors of \( L \\)
4. Let \( V = [v_1, \ldots, v_n] \in \mathbb{R}^{n \times k} \)
5. Cluster the points \( \{y_i\}, i=1,m \) using eg kmeans

Output: Cluster \( A_i, \ldots, A_k \) with \( A_i = \{j : y_j \in G \} \)

Data point: In the new representation \( \{y_i\} \), clustering is much easier.

Main point: In the new representation \( \{y_i\} \) clustering is much easier.
Graph Cut point of view

\[ \text{Cut} (A_1, \ldots, A_k) = \frac{k}{2} \sum_{i=1}^{k} w(A_i, \overline{A_i}) \]

\[ \text{Ratio-cut} (A_1, \ldots, A_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{w(A_i, \overline{A_i})}{\text{Vol}(A_i)} = \frac{1}{2} \sum_{i=1}^{k} \frac{\text{cut}(A_i, \overline{A_i})}{\text{Vol}(A_i)} \]

\[ \text{Ncut} (A_1, \ldots, A_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{w(A_i, \overline{A_i})}{\text{Vol}(A_i)} = \frac{1}{2} \sum_{i=1}^{k} \frac{\text{cut}(A_i, \overline{A_i})}{\text{Vol}(A_i)} \]

Relaxations:
- Ratio-cut - unnormalized spectral clustering
- Ncut - normalized spectral clustering

Random Walks:
- Unique stationary distribution of graph connected and non-bipartite

\[ \pi = (\pi_1, \ldots, \pi_n)^T \text{ with } \pi_i = \frac{d_i}{\text{Vol}(v)} = \frac{d_i}{\sum_{j=1}^{n} d_j} \]

Transition matrix:
- \[ P = D^{-1}W \quad (\text{note } L_{W} = I - P) \]
- \( (x, W) \) corresponds to \( L_{W} \) eigenpair of \( P \)
- Smallest eigenvalue of \( L_{W} \) \( \geq \) largest eigenvalue of \( P \)

\[ \text{Ncut} (A, \overline{A}) = P(A|A) + P(A|\overline{A}) \]

Commute distance (resistance distance)

\[ C_{ij} = \text{expected time it takes the random walk to travel from vertex } v_i \text{ to vertex } v_j \text{ and back} \]

\[ \Rightarrow \text{instead of looking for the single shortest path it takes into account several reasonable paths} \]

\[ C_{ij} = \text{vol}(v) \left( L_{ii}^{-2} L_{ij}^{+} + L_{jj}^{+} \right) = \text{vol}(v) (e_i - e_j)^T L^+ (e_i - e_j) \]

where \( L^+ = \text{principal submatrix} \)

\[ \sqrt{C_{ij}} \] can be considered as Euclidean distance on the vertices of the graph.

Construct an embedding which maps the vertices \( v_i \) of the graph on points \( z_i \in \mathbb{R}^n \) such that the Euclidean distaces between the points \( z_i \) coincide with the commute distances on the graph.