CSE5095: Research Topics in Big Data Analytics **Professor Rajasekaran**
4/3/14 Lecture 19: Hierarchical Clustering note taker: Timothy Becker

Rules Inference – we can use techniques similar to the Apriori (pruning).

Random Sampling – take a random sample *S* (and lower the support threshold). Identify the set *Q* of frequent itemsets in the sample. Hopefully, this set will be a superset of all the frequent itemsets in the database DB.

Q: How can we be sure that we have found all the true frequent itemsets in DB?

In the last lecture we introduced the notion of the negative border. Let *NB(Q)* be the negative border of *Q*. Compute the support in DB of each itemset in *Q* and *NB(Q)*. If none of the itemsets in *NB(Q)* is frequent, then we output all the imtesets in *Q* whose support in DB is greater than or equal to *minSupport* and quit. If there exists at least one itemset in *NB(Q)* whose support in DB is greater than or equal to *minSupport,* then we know that *Q* is not a superset of all the frequent itemsets. In this case we use another algorithm to identify all the frequent itemsets in DB.

Analysis of Randomized Rules Mining
Let $\boldsymbol{X}$ be any itemset whose support in the DB is $f(\boldsymbol{X})$.
Let the support of $\boldsymbol{X}$ in the sample $\boldsymbol{S}$ be $f'(\boldsymbol{X})$ and let $|\boldsymbol{S}| = s$.
$f(\boldsymbol{X})$ and $f'(\boldsymbol{X})$ are expressed as fractions.

Q: what can we say about $f'(X)$ as a function of $f(\boldsymbol{X})$?

Note that $sf'(\boldsymbol{X}) = B(s, f(\boldsymbol{X}))$ ($sf'(\boldsymbol{X})$ is a Binomial Random Variable with $n=s$, and $p=f(\boldsymbol{X})$). In the following analysis δ and δ' are user defined probabilities (usually very small).

Using Chernoff bounds we have:
$Prob\left(sf'(\boldsymbol{X}) < (1-\epsilon)sf(\boldsymbol{X})\right) \leq \exp\left[\frac{-\epsilon^2 sf(\boldsymbol{X})}{2}\right]$. We want this probability to be $\leq \delta$ so:

$$\delta = \exp\left[\frac{-\epsilon^2 sf(\boldsymbol{X})}{2}\right] \Rightarrow \ln(\delta) = \frac{-\epsilon^2 sf(\boldsymbol{X})}{2} \Rightarrow s \geq \frac{2ln\left(\frac{1}{\delta}\right)}{\epsilon^2 f(\boldsymbol{X})}$$

Let $\boldsymbol{N}$ be an upper bound on the number of frequent itemsets in DB and $\boldsymbol{m}$ denote *minSupport*, then:
$Prob(\exists\ a\ frequent\ itemset\ of\ DB\ not\ frequent\ in\ S) \leq \boldsymbol{N}\exp\left[\frac{-\epsilon^2 s\boldsymbol{m}}{2}\right]$ (by the union bound).
We want this probability to be $\leq \delta'$ so:

$$\delta' = \boldsymbol{N}\exp\left[\frac{-\epsilon^2 s\boldsymbol{m}}{2}\right] \Rightarrow \ln\left(\frac{\delta'}{\boldsymbol{N}}\right) = \frac{-\epsilon^2 s\boldsymbol{m}}{2} \Rightarrow s = \frac{2ln\left(\frac{\boldsymbol{N}}{\delta'}\right)}{\epsilon^2 \boldsymbol{m}} \blacksquare$$

(1) With high probability we will take one sampling pass. The value of minimum support to be employed in the sample is determined using the above equations.
(2) The authors propose a deterministic 2-phase algorithm if the sample misses a frequent itemset found by the negative border *NB*(*S*), second sampling pass could be done instead.

End of Rules Mining (Vertical, Horizontal layouts all utilize heuristics like Apriori).

Clustering Algorithms

We can think of these as data-reduction (this would be point reduction by replacing each cluster by a single point (centroid of the cluster, for example)).

Let $\boldsymbol{X}$ be a given point set. Then, clustering is defined as a partitioning of $\boldsymbol{X}$ into k groups:
$\boldsymbol{X} \longmapsto \boldsymbol{X_1}, \boldsymbol{X_2}, \ldots \boldsymbol{X_k}$ *s.t.* points in each cluster are similar to each other.

Let $d(i,j)$ be a distance measure between $p_i, p_j \in \boldsymbol{X}$. We say p_i and p_j are similar if $d(i,j) \leq \epsilon$ (for some small value).

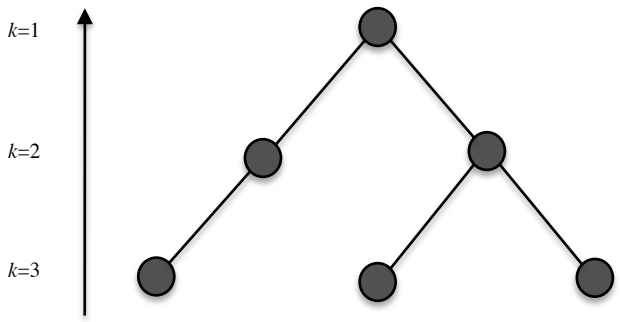
(In general, we can use some objective function to define clustering).

Hierarchical Clustering
input: $\boldsymbol{X}=\{p_1, p_2, \ldots p_n\}$, k (the number of clusters to compute)
output: k clusters

Algorithm
[0] to begin we have n clusters (each point is a cluster): $\{p_1\},\{p_2\},\ldots\{p_n\}$
[1] merge clusters (in many stages):
at any stage we merge the pair of clusters whose distance is minimum.

We could represent the sequence of merges done using a tree called the dendrogram.

Definition: distance between two clusters c_i, c_j can be defined in several ways:
(i) **single-link** - $d(c_i, c_j) = \min_{p \in c_i, q \in c_j}\{d(p,q)\}$
(ii) **complete-link** - $d(c_i, c_j) = \max_{p \in c_i, q \in c_j}\{d(p,q)\}$
(iii) **average-link** - $d(c_i, c_j) = \text{mean}_{p \in c_i, q \in c_j}\{d(p,q)\}$



Theorem: we can do hierarchical Clustering in $O(n^2)$ time.

Construct a pairwise distance matrix $\boldsymbol{D}[i,j] = d(p_i, p_j)$ and minimum array **min**. The **min** array stores the row minima.

D	1	…	*j*	…	*n*	**min**
1	*d(1,1)*	…	*d(1,j)*	…	*d(1,n)*	First row min
…	…		…		…	…
i	*d(i,1)*	…	*d(i,j)*	…	*d(i,n)*	…

…	…		…		…	…
n	*d(n,1)*	…	*d(n,j)*	…	*d(n,n)*	Min of row *n*

[0] to begin with using **min** find the closest points. At any given time we have a single row to represent a cluster.

[1] merging: Let p_i and p_j be the two closest clusters. We merge these two rows into one. For example, the merged row could be row i. Merging of the two rows is done as follows.
$d(i,q) = \min_{1 \le q \le n}\{d(i,q), d(j,q)\}$ and **min**[i] = min{**min**[i], **min**[j]}

[2] do [1] n-2 times

Analysis
It takes $O(n^2)$ time to do all pairwise calculations (assuming that $d(i,j)$ is done in $O(1)$ time for a specific i and j.)
For each merge we spend $O(n)$ time for a total merging time of $O(n^2)$.
Total time then is: $O(n^2) + O(n^2) = O(n^2)$.

Note: Instead of getting the number of clusters as the input, we could also think of a variation where we get a threshold on the distance. We could then eliminate all the edges in the dendrogram whose edge weights are greater than the threshold. Each tree in the resultant forest will be an output cluster.

Parallel Hierarchical Clustering (Rajasekaran 2004)
[1] Construct a minimum spanning tree (MST).
[2] Delete all the edges whose weights are greater than the given distance threshold.
[3] Find the connected components in the resultant forest. Each component is a cluster.

claim: ∃ a Las Vegas implementation of this algorithm on the CRCW PRAM model that runs in $\tilde{O}(\log n)$ time using $\frac{n^2}{\log n}$ processors.

Theorem (Pettie&Ramachandran 2002)
We can construct a MST for any undirected graph $G(V,E)$ in $\tilde{O}(\log|V|)$ using $\frac{|E|+|V|}{\log |V|}$ CRCW PRAM processors.
(use this for Rajasekaran step[1]).

Fact: We can find the connected components in a forest F in $O(\log |V|)$ using $|V|$ processors.

⇒ we can do $||^l$ Hierarchical Clustering in $\tilde{O}(\log n)$ time using $\frac{n^2}{\log n}$ processors.

Application: Records Linkage
Useful in biomedical informatics where individuals may vary services, generating several partial records at disparate medical locations.

input: $D_1, D_2, \ldots D_k$ (data sets of medical records)
output: clusters where each cluster contains records belonging to the same individual exclusively.

Approach: put all the records into one collection C, perform Hierarchical clustering on C. Each cluster is output as records belonging to one individual. Each record is thought of as a string of characters (obtained by concatenating the various fields in the record such as the first name, last name, social security number, etc.) Distance between two records is calculated as the edit distance between the two corresponding strings. Recall that string edit distance is the minimum number of edits needed to transform one string into another using insertion, deletion, and substitution operations).

If $\min\{|R_i|, |R_j|\} = L$ and the distance threshold is set as τ, then we can compute $d(R_i, R_j)$ in $O(L\tau)$ time.

Blocking Technique
Hash a substring of contiguous characters (or k-mer) into buckets and do clustering on the buckets.

[EX] k=3
If R = "abcdefghijklmnopqrstuvwxyz", the 3-mers in this string are
"abc", … , and "xyz".
There will be a bucket corresponding to each possible 3-mer. The record R will be hashed into buckets corresponding to all of the above 3-mers.

The intuition behind blocking is the observation that if two records are very similar, then they should share at least one l-mer (for some suitably small value of l).

End of lecture, continued in lecture 20.