CSE 4502/5717 - Big Data Analytics
Lecture 25, April 25, 2018

Record Linkage

Record linkage is an important problem in biomedical informatics where individuals may have multiple records. Identifying all the records belonging to each individual could lead to a lot savings in providing healthcare. We can define the record linkage problem as follows:

input: $R_1, R_2, \ldots, R_n$ (records coming from several datasets);
output: clusters where each cluster contains all the records belonging to one and only one individual.

If each record has a unique key and there are no errors in this key, then the above problem is trivial. However, in practice, there may not be any such unique key and there could be many sources of error. For instance, there could be typing mistakes, the first name and the last name could be swapped, nicknames could be used, names could be abbreviated, etc. These errors make the problem of record linkage challenging. The performance of any record linkage algorithm is measured with its run time and accuracy.

A simple algorithm: 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 can be 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.

Let R_i and R_j be any two records. Let $\max\{|R_i|, |R_j|\} = L$. We can compute the distance between R_i and R_j in $O(L^2)$ time using dynamic programming. If τ is an upper bound on the distance between R_i and R_j, then we can compute the distance between them in $O(L\tau$ time or $O(L + \tau^2)$ time.

The run time of the above simple algorithm can be seen to be $O(n^2 L\tau)$. We can improve this run time using the blocking teachnique.

Blocking Technique

The record linkage algorithm using the blocking technique works as follows:

Step 1: Let R be any record. For some small value of k (such as 3 or 4), we generate all possible k-mers of R. For example, let the alphabet be $\{0, 1, 2, \ldots, 9\}$, $R = 2313314211342132$, and $k = 3$. In this case the following 3-mers will be generated: $231, 313, 133, 331, 314, 142, 421, 211, 113, 134, 342, 421, 213, 132$. There will be a bucket corresponding to each possible k-mer. In the above example there will be $1,000$ buckets corresponding to the 3-mers $000, 001, 001, \ldots, 999$.

The record R will be placed in each of the buckets corresponding to its k-mers; In the above example, R will be placed in all the 14 buckets listed above. In particular, if $|R| = l$, then R will be placed in $l - k + 1$ buckets;

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

Step 2: for each bucket B **do**
Cluster the records in B (using the hierarchical clustering algorithm);

Step 3: Construct a graph $G(V, E)$ where each input record is a node. There will be an edge between two records R_i and R_j if these two records have been placed in the same cluster in at least one of the buckets (in Step 2);

Step 4: Find the connected components of $G(V, E)$. Output each connected component as a cluster (i.e., records belonging to one individual);

Analysis: Le the size of the alphabet be 36 (26 English letters and 10 digits). There will be 36^k buckets. Note that each record goes to $\leq (L - k + 1)$ buckets, where L is the maximum length of any record. This means that the expected size of each bucket is $\frac{n(L-k+1)}{36^k} < \frac{nL}{36^k}$.

The expected time spent in clustering each bucket is $O\left(\left(\frac{nL}{36^k}\right)^2 L\tau\right)$. Summing over all the buckets, the total expected time spent (in Step 2) is $O\left(n^2 L\tau \frac{L^2}{36^k}\right)$.

Note that $\frac{L^2}{36^k}$ could be much less than one and hence blocking helps in reducing the run time. For example, consider the case of $L = 36$ and $k = 4$.
$\rightarrow \frac{L^2}{36^k} = \frac{36^2}{36^4} = \frac{1}{1296} \ll 1$.
Steps 1, 3, and 4 take a total of $O(n(L - k + 1)) = O(nL)$ time. $\square$

There exist many other clustering algorithms. One such is the k-means clustering algorithm.
$k - means$ **clustering:**
Input: $p_1, p_2, ..., p_n$
Output: $C_1, C_2, ..., C_k$
Let $q_1, q_2, ..., q_k$ be the centers of the clusters.
We want to minimize: $\sum_{i=1}^{k} \sum_{p_i^j \in C_i} (p_i^j - q_i)$
Lloyd's Algorithm:
Start with k random points as centers. Associate each point with the center that is the closest to it. Now we have new clusters. Compute the center of each cluster. Associate each point with the center that is the closest to it. Repeat the above process until the clusters do not change any more.

The Closest Pair Problem

Given a set of n points in a metric space, the closest pair problem (CPP) is to identify the closest pair of points. This problem has numerous applications such as in Clustering, Time Series Motifs, Genome-wide Association Study (GWAS), etc. A simple algorithm takes $O(n^2)$ time. There exists an $O(n \log n)$ time divide and conquer algorithm. There exist randomized algorithms that take an expected linear time as well. All the above algorithms have an exponential dependence on the dimension. Algorithms that perform the best in practice happen to have $O(n^2)$ run times in the worst case. The run times of these algorithms depend linearly on the dimension. A number of heuristics are used in practice (in conjunction with the simple $O(n^2)$ time algorithm): early abandoning, pruning using the triangular inequality.

Early abondoning: The idea here is to speedup the computation of distances. Let $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$ be any two points in $\Re^d$. To compute the distance between x and y, the algorithm keeps adding $(x_i - y_i)^2$ for $i = 1, 2, \ldots, d$. When the sum exceeds δ^2 in the middle of this computation, this pair is immediately dropped (without completing the rest of the distance computation). Here δ is a known upper bound on the distance between the closest pair of points. This technique is known as *early abandoning*. For example, a value for δ could be otained at the beginning of any algorithm using a random sample of points. From thereon, δ could be dynamically updated to be the smallest distance among all the point pairs that have been processed until that point in the algorithm.

Pruning using the triangular inequality: Let x and y be any two points. At any stage in the algorithm, we have an upper bound δ on the distance between the closest pair of points. If $d(x, y)$ can be inferred to be greater than δ, then we can drop the pair (x, y) from future consideration (since this pair cannot be the closest). Ideally, we would like to calculate $d(x, y)$ exactly for every pair of points x and y. But this will take too much time. We can circumvent this problem by **estimating** the distance between x and y via the triangular inequality. In particular, a random reference point r is chosen and the distance between each input point and r is computed. The points are kept in an ascending order of their distances to r. From thereon, $d(r, y) - d(r, x)$ is used as a lower bound on $d(x, y)$. If this lower bound is $> \delta$, then (x, y) is dropped from future consideration.

Let $p_1, p_2, \ldots, p_n$ be the input points. Choose one of these as reference. Let this point be r. Compute the distance between r and every input point and sort them based on this distance. For every point p_i in this sorted list compute the distance between p_i and every point to the right. We always keep the smallest distance δ seen so far. Let p_j be a point to the right of p_i. Trinagular inequality implies that $d(p_i, p_j) \geq d(r, p_j) - d(r, p_i)$. If the RHS is $> \delta$, we can skip the computation of the distance between p_i and p_j and every point to the right of p_j. See Figure 1.

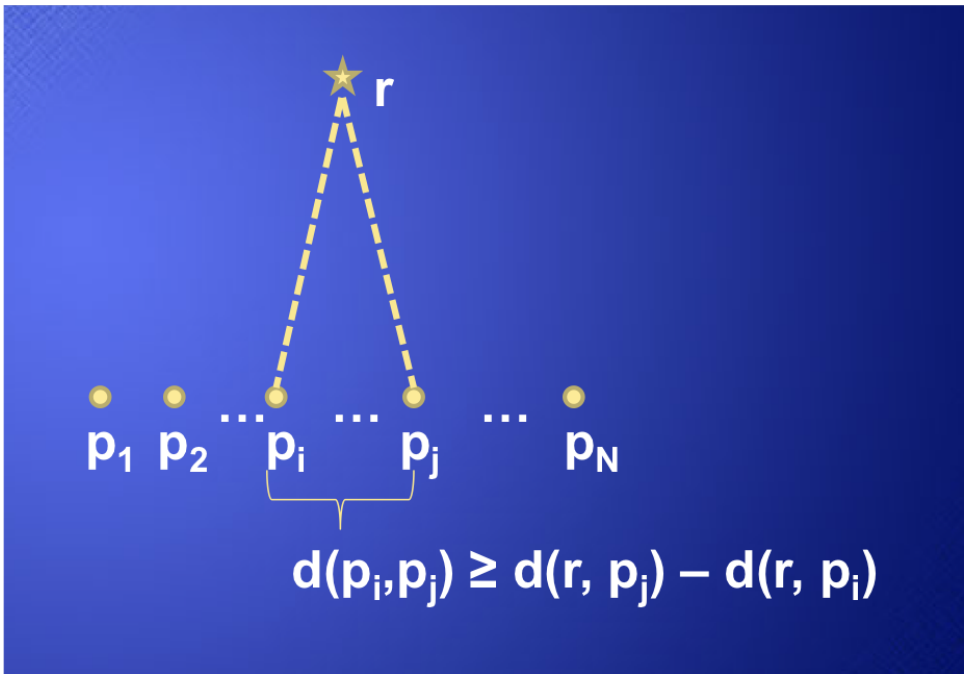

Figure 1: Pruning using the triangular inequality.

Projecting the reference point: This technique exploits the relationship between $d(r, x)$ and $|d(r, x) - d(r, y)|$. In particular, we pick a random reference point and project it out by multiplying each coordinate value of the point by a factor of f. For example, f could be 10. The rest of the algorithm is the same as before (i.e., use early abandoning and pruning using the triangular inequality). To illustrate how projection helps see Figure 2. Consider two points $A = (0, 0)$ and $B = (1, 0)$. When we use r_1 as the reference point (for which $d(A, r_1) = \sqrt{2}$)), $d(r_1, A) - d(r_1, B) \approx 0.41$. When we use r_3 as the reference (for which $d(r_3, A) = 10$), $d(r_3, A) - d(r_3, B)$ increases to around 0.68.

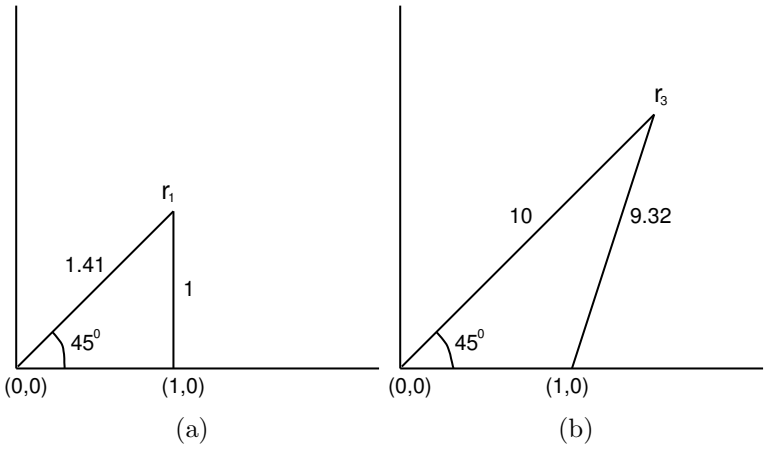

Figure 2: The effect of scaling on reference points.