UNIT V

CHAPTER 5

Big Data Mining Algorithms

University Prescribed Syllabus

Frequent Pattern Mining : Handling Larger Datasets in Main Memory Basic Algorithm of Park, Chen, and Yu. The SON Algorithm and MapReduce. Clustering Algorithms : CURE Algorithm. Canopy Clustering, Clustering with MapReduce Classification Algorithms: Overview SVM classifiers, Parallel SVM, KNearest Neighbor classifications for Big Data, One Nearest Neighbour.

Self-learning Topics : Standard libraries included with spark like graphX, MLlib

5.1 FREQUENT PATTERN MINING

5.1.1 Handling Larger Datasets in Main Memory Basic Algorithm of Park, Chen, and Yu

PCY (Park-Chen-Yu) Algorithm

In pass 1 of A-Priori, most memory is idle. We store only individual item counts. We use the idle memory to reduce memory required in pass 2.

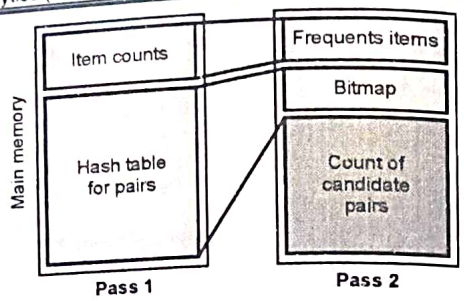

Fig. 5.1.1 : Organization of main memory for the first two passes of the PCY Algorithm

☞ Pass 1 of PCY

In addition to item counts, maintain a hash table with as many buckets as fit in memory. Keep a count for each bucket into which pairs of items are hashed v For each bucket just keep the count, not the actual pairs that hash to the bucket.

FOR (each basket)

FOR (each item in the basket)

add 1 to item's count;

FOR (each pair of items)

hash the pair to a bucket;

add 1 to the count for that bucket;

- Pairs of items need to be generated from the input file; they are not present in the file.
- We are not just interested in the presence of a pair, but we need to see whether it is present at least s (support) times.
- If a bucket contains a frequent pair, then the bucket is surely frequent v However, even without any frequent pair, a bucket can still be frequent. So, we cannot use the hash to eliminate any member (pair) of a "frequent" bucket.

But, for a bucket with total count less than s, none of its pairs can be frequent. Pairs that hash to this bucket can be eliminated as candidates (even if the pair consists of 2 frequent items).

E.g., even though {A}, {B} are frequent, count of the bucket containing {A,B} might be < s

Pass 2 : Only count pairs that hash to frequent buckets

Replace the buckets by a bit-vector: 1 means the bucket count exceeded the support s (call it a frequent bucket); 0 means it did not.

4-byte integer counts are replaced by bits, so the bit-vector requires 1/32 of memory. Also, decide which items are frequent and list them for the second pass.

Count all pairs {i, j} that meet the conditions for being a candidate pair :

1. Both i and j are frequent items
2. The pair {i, j} hashes to a bucket whose bit in the bit vector is 1 (i.e., a frequent bucket).

Both conditions are necessary for the pair to have a chance of being frequent.

5.1.2 The SON Algorithm and MapReduce

The SON algorithm lends itself well to a parallel-computing environment. Each of the chunks can be processed in parallel, and the frequent itemsets from each chunk combined to form the candidates. We can distribute the candidates to many processors, have each processor count the support for each candidate in a subset of the baskets, and finally sum those supports to get the support for each candidate itemset in the whole dataset. This process does not have to

be implemented in MapReduce, but there is a natural way of expressing each of the two passes as a MapReduce operation.

- **First Map Function :** Take the assigned subset of the baskets and find the itemsets frequent in the subset using the algorithm. As described there, lower the support threshold from s to ps if each Map task gets fraction p of the total input file. The output is a set of key-value pairs (F, 1), where F is a frequent itemset from the sample. The value is always 1 and is irrelevant.
- **First Reduce Function :** Each Reduce task is assigned a set of keys, which are itemsets. The value is ignored, and the Reduce task simply produces those keys (itemsets) that appear one or more times. Thus, the output of the first Reduce function is the candidate itemsets.
- **Second Map Function :** The Map tasks for the second Map function take all the output from the first Reduce Function (the candidate itemsets) and a portion of the input data file. Each Map task counts the number of occurrences of each of the candidate itemsets among the baskets in the portion of the dataset that it was assigned. The output is a set of key-value pairs (C, v), where C is one of the candidate sets and v is the support for that itemset among the baskets that were input to this Map task.
- **Second Reduce Function :** The Reduce tasks take the itemsets they are given as keys and sum the associated values. The result is the total support for each of the itemsets that the Reduce task was assigned to handle. Those itemsets whose sum of values is at least s are frequent in the whole dataset, so the Reduce task outputs these itemsets with their counts. Itemsets that do not have total support at least s are not transmitted to the output of the Reduce task.

5.1.3 Clustering Algorithms

5.1.3(A) CURE Algorithm

CURE (Clustering Using Representatives), assumes a Euclidean space. However, it does not assume anything about the shape of clusters; they need not be normally distributed, and can even have strange bends, S-shapes, or even rings. Instead of representing clusters by their centroid, it uses a collection of representative points, as the name implies.

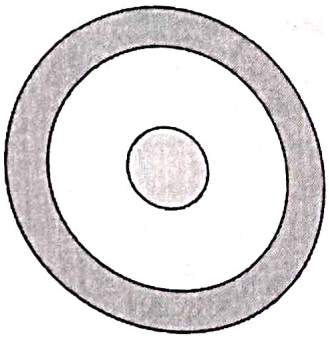

Fig. 5.1.2 : Two clusters, one surrounding the other

Example 5.1.1 : Fig. 5.1.2 is an illustration of two clusters. The inner cluster is an ordinary circle, while the second is a ring around the circle. This arrangement is not completely pathological. A creature from another galaxy might look at our solar system and observe that the objects cluster into an inner circle (the planets) and an outer ring (the Kuyper belt), with little in between.

5.1.3(B) Initialization in CURE

1. Take a small sample of the data and cluster it in main memory. In principle, any clustering method could be used, but as CURE is designed to handle oddly shaped clusters, it is often advisable to

use a hierarchical method in which clusters are merged when they have a close pair of points.

2. Select a small set of points from each cluster to be representative points. These points should be chosen to be as far from one another as possible, using the method.

3. Move each of the representative points a fixed fraction of the distance between its location and the centroid of its cluster. Perhaps 20% is a good fraction to choose. Note that this step requires a Euclidean space, since otherwise, there might not be any notion of a line between two points.

Example 5.1.2 : We could use a hierarchical clustering algorithm on a sample of the data from Fig. 5.1.2. If we took as the distance between clusters the shortest distance between any pair of points, one from each cluster, then we would correctly find the two clusters. That is, pieces of the ring would stick together, and pieces of the inner circle would stick together, but pieces of ring would always be far away from the pieces of the circle. Note that if we used the rule that the distance between clusters was the distance between their centroids, then we might not get the intuitively correct result. The reason is that the centroids of both clusters are in the center of the diagram.

For the second step, we pick the representative points. If the sample from which the clusters are constructed is large enough, we can count on a cluster's sample points at greatest distance from one another lying on the boundary of the cluster. Fig. 5.1.3 suggests what our initial selection of sample points might look like.

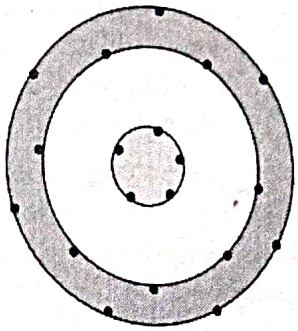

Fig. 5.1.3 : Select representative points from each cluster, as far from one another as possible

Finally, we move the representative points a fixed fraction of the distance from their true location toward the centroid of the cluster. In Fig. 5.1.3 both clusters have their centroid in the same place: the center of the inner circle. Thus, the representative points from the circle move inside the cluster, as was intended. Points on the outer edge of the ring also move into their cluster, but points on the ring's inner edge move outside the cluster.

5.1.3(C) Completion of the CURE Algorithm

The next phase of CURE is to merge two clusters if they have a pair of representative points, one from each cluster, that are sufficiently close. The user may pick the distance that defines "close." This merging step can repeat, until there are no more sufficiently close clusters.

Example 5.1.3 : The situation of Fig. 5.1.4 serves as a useful illustration. There is some argument that the ring and circle should really be merged, because their centroids are the same. For instance, if the gap between the ring and circle were much smaller, it might well be argued that combining the points of the ring and circle into a single

cluster reflected the true state of affairs. For instance, the rings of Saturn have narrow gaps between them, but it is reasonable to visualize the rings as a single object, rather than several concentric objects. In the case of Fig. 5.1.4 the choice of :

1. The fraction of the distance to the centroid that we move the representative points and
2. The choice of how far apart representative points of two clusters need to be to avoid merger.

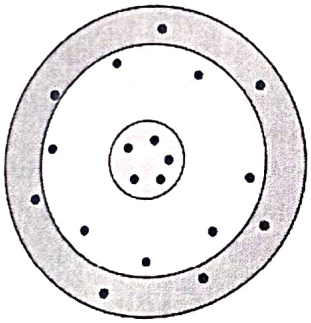

Fig. 5.1.4 : Moving the representative points 20% of the distance to the cluster's centroid

The last step of CURE is point assignment. Each point p is brought from secondary storage and compared with the representative points. We assign p to the cluster of the representative point that is closest to p.

5.1.4 Canopy Clustering

Canopy clustering is a fast and approximate clustering technique. It divides the input data points into overlapping clusters called canopies. Two different distance thresholds are used for the estimation of the cluster centroids. Canopy clustering can provide a quick approximation of the number of clusters and initial cluster centroids of

a given dataset. It is mainly used to understand the data and provide input to algorithms such as k-means.

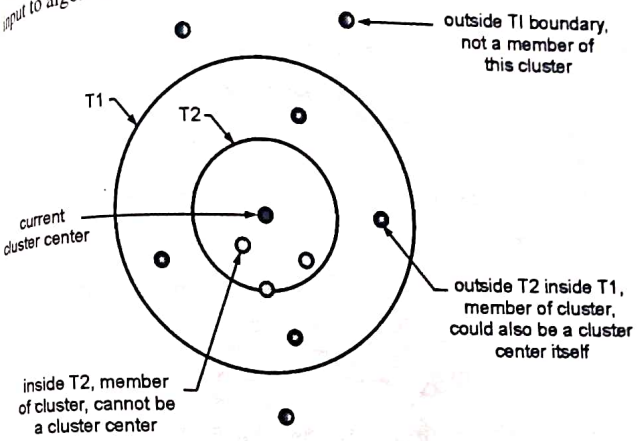

Fig. 5.1.5 : Canopy Clustering

- It is an unsupervised pre-clustering algorithm, often used as pre processing step for the K-means algorithm or the Hierarchical clustering algorithm. It speeds up the clustering operations on large data sets, whereby the other algorithms may be impractical due to the size of the data set. Briefly the algorithm may be stated as :
  - (i) Cheaply partitioning the data into overlapping subsets (called "canopies")
  - (ii) Perform more expensive clustering, but only within these canopies

Algorithm

(i) Two distance thresholds T1 and T2 are decided such that T1 > T2

(ii) A set of points are considered and remove one at random.

(iii) Create a Canopy containing this point and iterate through the remainder of the point set.

(iv) At each point, if its distance from the first point is < T1, then add the point to the cluster and f the distance is < T2, then remove the point from the set.

(v) With the processing in step iv, points close to the original avoids all further processing.

(vi) The algorithm loops until the initial set is empty, accumulating a set of Canopies, each containing one or more points. A given point may occur in more than one Canopy

5.1.5 Clustering with MapReduce Classification Algorithms

5.1.5(A) Overview SVM Classifiers

An SVM selects one particular hyperplane that not only separates the points in the two classes, but does so in a way that maximizes the margin – the distance between the hyperplane and the closest points of the training set.

The goal of an SVM is to select a hyperplane $w.x + b = 0$ that maximizes the distance γ between the hyperplane and any point of the training set. The idea is suggested by Fig. 5.1.6. There, we see the points of two classes and a hyperplane dividing them.

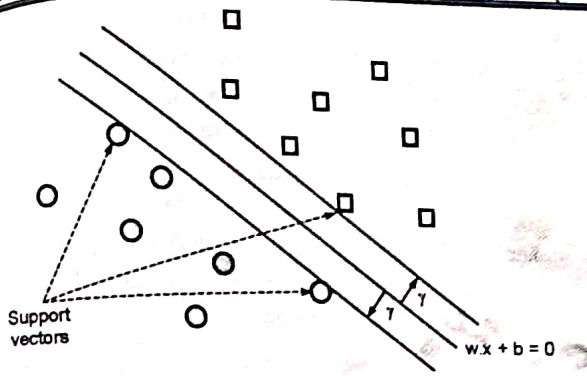

Fig. 5.1.6 : An SVM selects the hyperplane with the greatest possible margin γ between the hyperplane and the training points

- In Fig. 5.1.6 two parallel hyperplanes at distance γ from the central hyperplane $w.x + b = 0$, and these each touch one or more of the support vectors. The latter are the points that actually constrain the dividing hyperplane, in the sense that they are all at distance γ from the hyperplane.
- The SVM improves upon perceptrons by finding a separating hyperplane that not only separates the positive and negative points, but does so in a way that maximizes the margin – the distance perpendicular to the hyperplane to the nearest points. The points that lie exactly at this minimum distance are the support vectors. Alternatively, the SVM can be designed to allow points that are too close to the hyperplane, or even on the wrong side of the hyperplane, but minimize the error due to such misplaced points.

5.1.5(B) Parallel SVM

- Start with the current w and b, and in parallel do several iterations based on each training example. Then average the changes for each of the examples to create a new w and b. If we distribute w

and b to each mapper, then the Map tasks can do as many iterations as we wish to do in one round, and we need use the Reduce tasks only to average the results. One iteration of MapReduce is needed for each round.

- A second approach is to follow the prescription given here, but implement the computation of the second term in below Equation in parallel. The contribution from each training example can then be summed. This approach requires one round of MapReduce for each iteration of gradient descent.

$$f(w, b) = \frac{1}{2}\sum_{j=1}^{d} w_j^2 + C \sum_{i=1}^{n} \max\left\{0, 1 - y_i\left(\sum_{j=1}^{d} w_j x_{ij} + b\right)\right\}$$

The first term encourages small w, while the second term, involving the constant C that must be chosen properly, represents the penalty for bad points.

5.1.6 KNearest Neighbor Classifications for Big Data

- In this approach to machine learning, the entire training set is used as the model. For each ("query") point to be classified, we search for its k nearest neighbors in the training set. The classification of the query point is some function of the labels of these k neighbors.
- The training set is first preprocessed and stored. The decisions take place when a new example, called the query example arrives and must be classified.
- There are several decisions we must make in order to design a nearest neighbor-based algorithm that will classify query examples. Like :

(1) What distance measure do we use?

(2) How many of the nearest neighbors do we look at?

(3) How do we weight the nearest neighbors? Normally, we provide a function (the kernel function) of the distance between the query example and its nearest neighbors in the training set, and use this function to weight the neighbors.

(4) How do we define the label to associate with the query? This label is some function of the labels of the nearest neighbors, perhaps weighted by the kernel function, or perhaps not. If there is no weighting, then the kernel function need not be specified.

5.1.6(A) One Nearest Neighbour

The simplest cases of nearest-neighbor learning are when we choose only the one neighbor that is nearest the query example. In that case, there is no use for weighting the neighbors, so the kernel function is omitted. There is also typically only one possible choice for the labeling function: take the label of the query to be the same as the label of the nearest neighbor.

Example : Fig. 5.1.7 shows some of the examples of dogs that last appeared in Fig. 5.1.7. We have dropped most of the examples for simplicity, leaving only three Chihuahuas, two Dachshunds, and two Beagles. Since the height-weight vectors describing the dogs are two-dimensional, there is a simple and efficient way to construct a Voronoi diagram for the points, in which the perpendicular bisectors of the lines between each pair of points is constructed. Each point gets a region around it, containing all the points to which it is the nearest. These regions are always convex, although they may be open to infinity in one direction. 2 It is also a surprising fact that, even though there are O(n2) perpendicular bisectors for n points, the Voronoi diagram can be found in O(n log n) time.

In Fig. 5.1.7 we see the Voronoi diagram for the seven points. The boundaries that separate dogs of different breeds are shown solid, while the boundaries

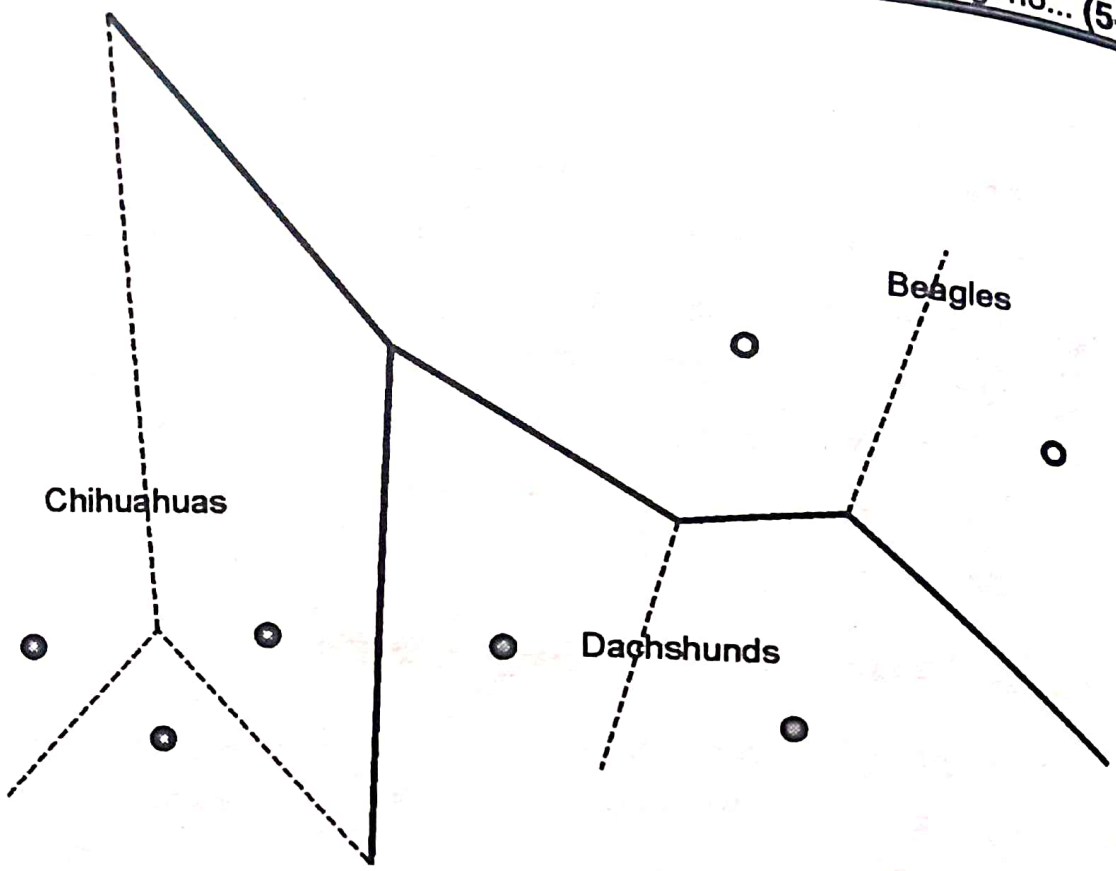

Fig. 5.1.7 : Voronoi diagram for the three breeds of dogs

- Between dogs of the same breed are shown dashed. Suppose a query example q is provided. Note that q is a point in the space of Fig. 5.1.7. We find the region into which q falls, and give q the label of the training example to which that region belongs. It is not too hard to find the region of q. We have to determine to which side of certain lines q falls.
- To compare a vector x with a hyperplane perpendicular to a vector w. In fact, if the lines that actually form parts of the Voronoi diagram are pre-processed properly, we can make the determination in O(log n) comparisons; it is not necessary to compare q with all of the O(n log n) lines that form part of the diagram.

Chapter Ends...

❑❑❑