TDT4300 Datavarehus og datagruvedrift - Spring 2013
Assignment 4: Clustering

Magnus L Kirø

April 14, 2013

1, apriori algorithm

minSupport First we find all elements, {a,b,c,d,e,f}. Then we count the occurrences of the elements:

element	count
a	3
b	2
c	2
d	1
e	1
f	1

Then we remove the elements that occurs fewer then 0.5% of the time. This fives us the frequent itemset of {a,b,c}

Reapeating step one, which is finding all itemsets from the frequent itemset, we get: {ab,ac,bc}.

Again we count:

element	count
ab	1
ac	2
bc	1

And prune away the elements that doesn't meet the minimum support og 0.5. This result in the minimum frequent itemset of {ac}

Repeating we craete all itemsets that combines ac with {a,b,c,d,e,f}, and are within the minimum suppoert range. Giving the count:

element	count
acb	1
acd	0
ace	0
acf	0

which results in an empty minFreqSet.

Then we have all the permutations of our shopping basket elements that meets the minimum support of 0.5. minFreqSet = {a,b,c,ac}.

Confidence Calculating the confidence with:

confidence = (permutation containing a also contains b) / (permutations only containing a)

permutation	in x and y	in y	confidence
a *gives* b	1	3	0.33
a *gives* c	2	3	0.66
a *gives* ac	2	3	0.66
b *gives* a	1	2	0.5
b *gives* c	1	2	0.5
b *gives* ac	1	2	0.5
c *gives* a	2	2	1
c *gives* b	1	2	0.5
c *gives* ac	2	2	1
ac *gives* a	2	2	1
ac *gives* b	1	2	0.5
ac *gives* c	2	2	1

Given the confidence limit we end up with strong rules of { c *gives* a , c *gives* ac, ac *gives* a, ac *gives* c }

Clustering

2 k-Means Clustering

Calculating the distance with: —centroid.value - point.value—

Clustering on p2 and p5 Given the table with starting values we split the dataset in two, ventering by the two centroids, p2 and p5. p2:{p1,p3,p4} and p5:{p6,p7,p8,p9,p10'}

Then we look at all p's that ain't a centroid, and compare the value of the given p with the value of each centroid.

p	distance from p2	distance from p5	closes centroid
p1	2	4	p2
p3	1	6	p2
p4	3	3	p2
p6	8	2	p5
p7	5	1	p5
p8	8	2	p5
p9	5	1	p5
p10	7	1	p5

Given the distances calculated we reassign p's that's not assigned to the closest centroid. In this case we were lucky with the initial assignment. It's only one p that maybe should be reassigned. But while it dosn't matter which centroid it belongs to it's more work to move it then to let it be, so we don't reassign it.

This results in the same cluastering as the initial one, and we have convergence. Clustering complete.

Clustering on centroids p2, p6 and p8 Initial cluster: p2:{p1, p2, p4}, p6:{p5, p7}, p8:{p9, p10}

calculating distances:

p	distance from p2	distance from p6	distance from p8	closes centroid
p1	2	6	6	p2
p3	1	7	7	p2
p4	3	5	5	p2
p5	8	2	2	p6
p7	5	3	3	p6
p9	5	3	3	p8
p10	7	1	1	p8

Again given the initial clustering we get no reasignments.

The initial clustering used for the example with three centroids is mainly by id and the amount of elements.

Which is something like: *for numCentroids assign (amountOfUnassignedElements/numCentroidsRemaining) number of elements to the current centroid.*

eksample:

- 1.iteration: 7 unasigned elements, divedid by 3(centroid without attached elements) is 2.33 elementsi(rounding up gives us 3). Assigning three elements to the first centroid.
- 2.iteration: 4 unassigned elements divided by 2 is 2. Assigning 2 elements to the second unasigned centroid.

- 3.iteration: Last centroid, assigning the rest of the unassigned elements to this centroid.

2.2 Hierarchical Agglomerative Clustering HAC

1 min/max - linking

min-linking is "The minimum distance between elements of each cluster". Given two clusters a and b. We get the minimum distance by calculating the distance between elements from a and b and then finding the minvalue of the calculated distances. = min{d(x,y): x is an element of a, y is an element of b }

max-linking is the opposite of the minlinking. We do mostly the same thing. Calculate the distances. {d(x,y): x is an element of a, y is an element of b } and taking the maximum value from the set of calculated distances. = max{d(x,y): x is an element of a, y is an element of b }

The main difference is that we define the distance between two clusters as either the maximal distance between two nodes in each cluster, or the minimum distance between two nodes in each cluster. Rather the two(one from each cluster) nodes furthes apart defines the distance(maximal) or the two(one from each cluster) closest nodes defines the distance(minimal)

Performed hierarchical agglomerative clustering

Given the data in table three we have: (x,y):{(1,11), (1,9), (1,5), (1,2), (6,7), (11,7)}

This gives us these distances between node pairs. (the length of the vector given by the two nodes) sqrt($(x_2 - x_1)^2 + (y_2 - y_1)^2$)

With python correct syntax for calculation we get the distance between two nodes:

node pair	distance calculation	distance
1.2	sqrt((1-1)**2 + (9-11)**2)	2
1.3	sqrt((1-1)**2 + (5-11)**2)	6
1.4	sqrt((1-1)**2 + (2-11)**2)	9
1.5	sqrt((6-1)**2 + (7-11)**2)	6.40
1.6	sqrt((11-1)**2 + (7-11)**2)	10.77
2.3	sqrt((1-1)**2 + (5-9)**2)	4
2.4	sqrt((1-1)**2 + (2-9)**2)	7
2.5	sqrt((6-1)**2 + (7-9)**2)	5.38
2.6	sqrt((11-1)**2 + (7-9)**2)	10.19
3.4	sqrt((1-1)**2 + (2-5)**2)	3
3.5	sqrt((6-1)**2 + (7-5)**2)	5.38
3.6	sqrt((11-1)**2 + (7-5)**2)	10.19
4.5	sqrt((6-1)**2 + (7-2)**2)	7.07
4.6	sqrt((11-1)**2 + (7-2)**2)	11.18
5.6	sqrt((11-6)**2 + (7-7)**2)	5

Assuming that the more clustering on each level the better we get:

min-linking Given the height of the tree the dendrogram vould be like this: Were {x,y} is a node set, containing the nodes with the shortest distance between them.

- 1: {1}, {2}, {3}, {4}, {5}, {6}
- 2: {1,2}, {3,4}, {5,6}
- 3: {1,2,3,4}, {5,6}
- 4: {1,2,3,4,5,6}

max-linking Given the height of the tree the dendrogram vould be like this: Were {x,y} is a node set, containing the nodes with the longest distance between them.

- 1: {1}, {2}, {3}, {4}, {5}, {6}
- 2: {4,6}, {1,5}, {3,2}
- 3: {4,6,1,5}, {3,2}
- 4: {1,2,3,4,5,6}

Clustering Methods

Cases:

- 1: text collection
  Density based clustering would be a good choice here. We don't need to run the algorith multiple times to get a good result and the space and time requirements are over the tope. the overall runtime complexity is O(n*log n)

- 2: noisy data
  Connectivity based clustering, HAC would be a good choise here while HAC doesn't have any notion of noise.

- 3: colletion with little noise
  With the quite small dataset and little noise k-means clustering can be usefull by running it multiple times. With little data this would be quick, and therefore we can obtain a better result by running multiple fast clusterings.

Plots:

- a, three pictures with different data clustering

  left img: k-means

  center img: Density-based clustering, DBSCAN, detects clusters ased on proximity.

  right img: HAC, would gather the data into multiple clusters. While the data is quite spread the single-linkage would work as wel as any other proximity measure.

- b, HAC-min-linkage, would craete three clusters by proximity. Would also be quite quick. With low complexity.