CS 533 Spring 2014
Assignment 3
Devrim Şahin

1.

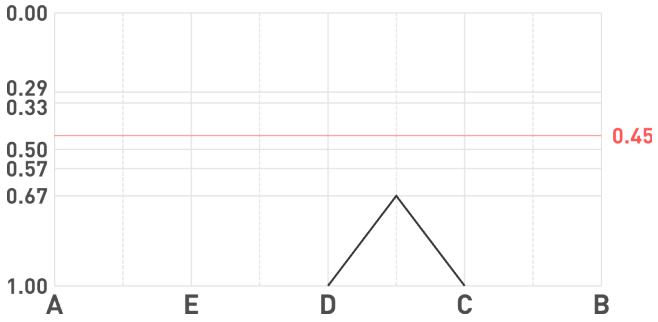
Using Dice coefficient

$$f(X, Y) = 2\,|X \cap Y| \,/\, (|X| + |Y|)$$

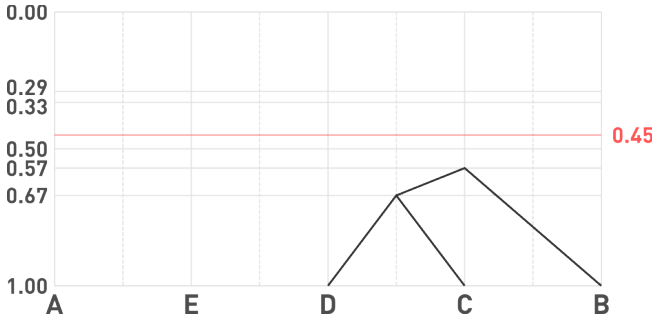
we can calculate and sort pairwise similarities as below:

CD	BD	AE	DE	BC	AB	BE	AC	AD	CE
0.67	0.57	0.50	0.50	0.33	0.29	0.29	0.00	0.00	0.00

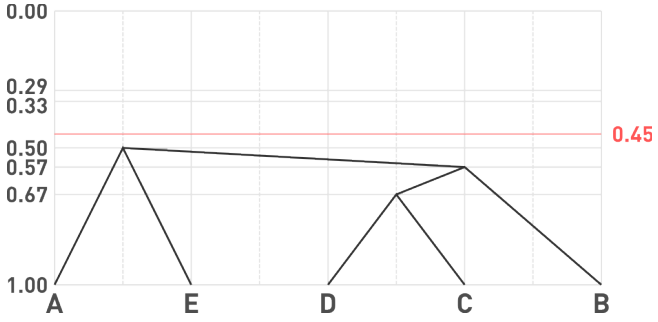
Then, starting from CD, we incrementally add similarities to the dendogram:



Then we add BD. Since this is the single-link dendogram; we do not wait to add B to CD cluster until there is a BC similarity (unlike complete-link):



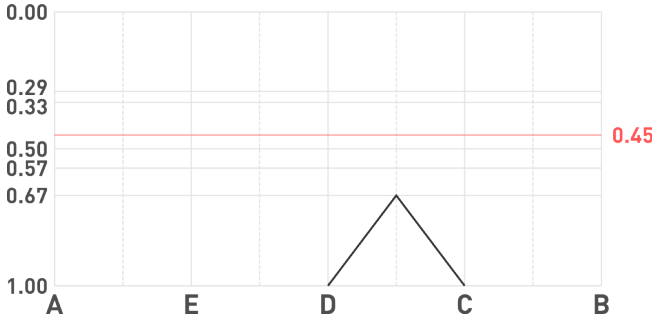
Similarly, we keep adding; and we have a complete dendogram at the 4th iteration already:



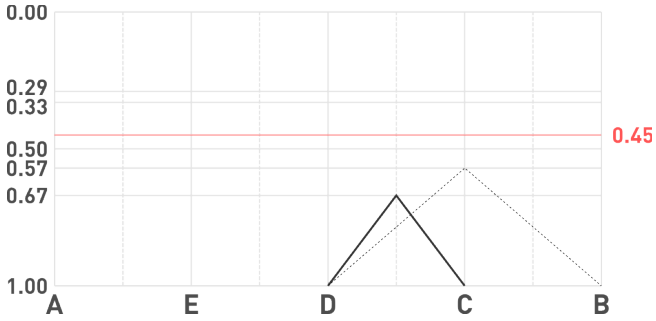
Note that none of the connections is above the threshold; therefore all points are connected into one cluster: **ABCDE**.

2.

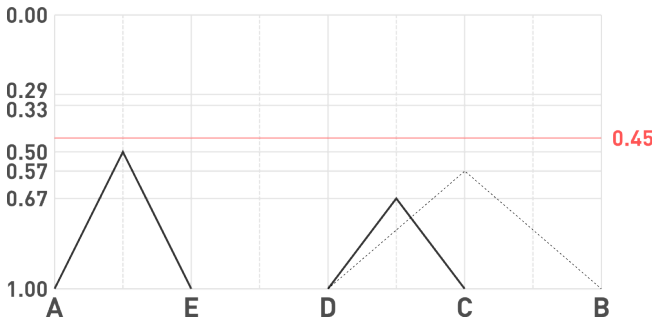
Similarly, add CD:



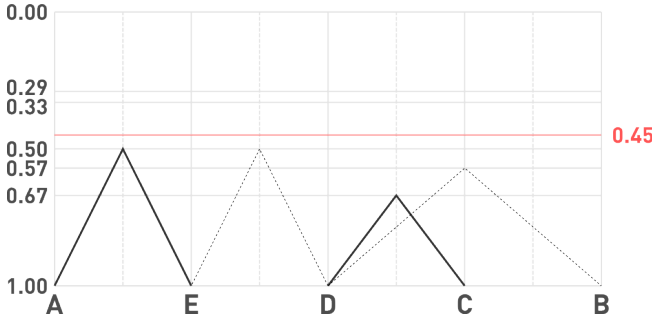
Next is BD, but we cannot add it to the CD cluster because we do not know the BC relation yet. Since this is the complete-link dendogram, we should know every pairwise similarity between two clusters in order to combine them in a similarity value, unlike the single-link case where we immediately merge. Just keep the BD relation in mind (shown with dashed lines):



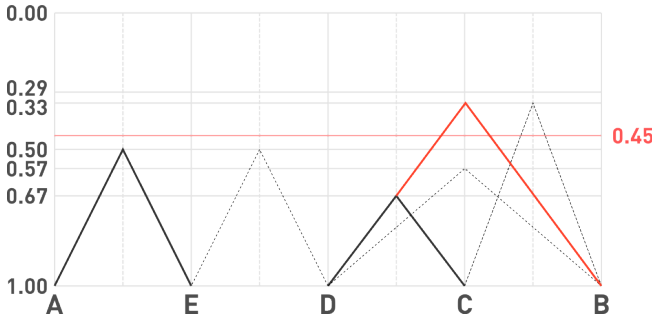
AE:



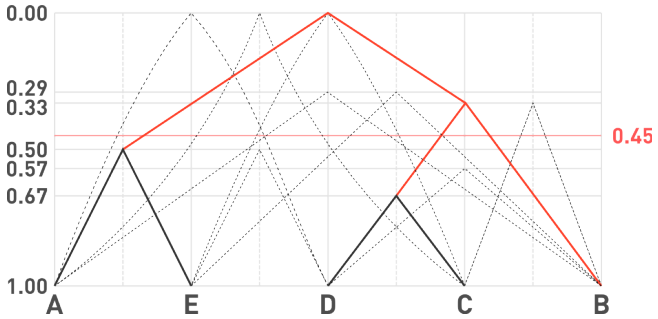
DE (again, no connection because we don't know AD, AC, CE):



Adding BC, now we have a complete list of relations between B and CD clusters (we know both BC and CD). Therefore we can use the minimum of these (BC = 0.33) to combine the two clusters. Note that we have already passed the threshold, and any similarity after this will be smaller than the threshold, so we can actually stop here for the clustering purposes:



Below is the full dendogram (observe that we should visit every pairwise similarity to construct this, as shown by the dashed lines):



C

Note that in this case we have 3 clusters instead of 1: **AE, CD** and **B**.

3. **a.**

$$C = \begin{bmatrix} 0.50 & 0.25 & 0.00 & 0.00 & 0.25 \\ 0.10 & 0.63 & 0.07 & 0.13 & 0.07 \\ 0.00 & 0.33 & 0.33 & 0.33 & 0.00 \\ 0.00 & 0.33 & 0.17 & 0.33 & 0.17 \\ 0.25 & 0.17 & 0.00 & 0.17 & 0.42 \end{bmatrix}$$

b. Sum up the diagonal entries -> 0.50 + 0.63 + 0.33 + 0.33 + 0.42 = 2.22 -> $\mathbf{n_c = 2}$.

c. Seed powers are calculated via $p_i = c_{ii}\,(1 - c_{ii}) / \alpha_i$. Then p_i values are 0.50, 1.16, 0.22, 0.44, 0.49 respectively for documents 1 to 5.

d. Since the seed power formula is a multiplication of coupling factor, decoupling factor and depth of indexing; we want to pick documents with highest p_i, so that out cluster seeds can be both similar to other items and dissimilar from each other. We are to pick the two documents with maximum p_i, because $n_c = 2$. Then our cluster seeds should be d_2 (p_2=1.16) and d_1 (p_1=0.50).

e. Below is the IISD:

t_1 => <1, 1>, <2, 1> t_2 => <2, 1> t_3 => <2, 1>

t_4 => <1, 1> t_5 => <2, 1> t_6 => <2, 1>

f. $d_5 = \{t_3, t_4\}$.
Start with $c_{51} = 0$, $c_{52} = 0$

Take t_3 from IISD: <2,1>

$c_{52} = c_{52} + \alpha_5 (d_{53}\ \beta_3\ d_{23}) = 0.17$

Take t_4 from IISD: <1,1>

$c_{51} = c_{51} + \alpha_5 (d_{54}\ \beta_4\ d_{14}) = 0.25$

Then, $c_{51} = 0.25 > 0.17 = c_{52}$; therefore d_5 belongs to cluster of d_1.

g. Since we already have the C matrix, I avoid the calculations in part f:
d_3 -> 2nd cluster
d_4 -> 2nd cluster
d_5 -> 1st cluster

Clusters, using the terminology in Q1, are **AE** and **BCD**. When we cut the complete-link dendogram from a threshold between 0.00 and 0.33, we obtain the same clustering.

h. (1) We have to find all c_{ii} values: m
assign all non-seeds to seeds: $(m-n_c)\ n_c$
in total: $\mathbf{m + m\ n_c - n_c^2}$
(2) $m = 5$, $n_c = 2$; $m+(m-n_c)n_c = \mathbf{11}$.

4. **a.** If all documents are unique, then two different documents will never have a common term. Thus, either d_{ik} or d_{jk} will be zero; meaning that $d_{ik}*d_{jk} = 0$ only when i = j. Therefore

$$c_{ij} = \alpha_i \sum_{k=1}^{n} \beta_k (d_{jk}\, d_{ik}) = \alpha_i \sum_{k=1}^{n} \beta_k 0 = 0 \quad for\ i \neq j.$$

For c_{ii}, though;

$$c_{ii} = \alpha_i \sum_{k=1}^{n} \beta_k\, d_{ik}^2$$

Now, if no documents have common terms, every term will occur in only one document. Therefore all the column sums are 1, meaning, all the β_k are 1. Also notice that d_{ik} being 1, $d_{ik}^2 = d_{ik}$:

$$C_{ii} = \alpha_i \sum_{k=1}^{n} \beta_k d_{ik}^2 = \alpha_i \sum_{k=1}^{n} 1\, d_{ik} = \frac{1}{\sum_{k=1}^{n} d_{ik}} \sum_{k=1}^{n} d_{ik} = 1$$

Therefore, if all documents are unique, we get an identity matrix.

b. If all documents are identical; then $d_{ik} = d_{jk}$ for all i, j. That is $d_{ik}d_{jk} = d_{ik}^2$:. Also note that all documents will have m terms, and all the β_k are 1/m:

$$C_{ij} = \alpha_i \sum_{k=1}^{n} \beta_k d_{ik}^2 = \alpha_i \sum_{k=1}^{n} \frac{1}{m} d_{ik} = \frac{1}{m} \frac{1}{\sum_{k=1}^{n} d_{ik}} \sum_{k=1}^{n} d_{ik} = \frac{1}{m}$$

5. **a.** m=6, n=5, t = 12; n_c = mn/t = 30/12 = 2.5 -> **2**. The two formulae are consistent. Assume all items are unique, m>>n, and (m-n) of terms never appear in any document.

Then c_{ii} = 1; therefore $n_c = n*c_{ii} = n$. However t = n; then mn/t = m. Since m>>n, they are inconsistent.

b.

Here n_c is clearly 3; but mn/t = 3*9/3 = 9. Obviously this is not a practical issue, because if some of the terms did not appear in any document, we would have discarded them already.

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

6. The incremental clustering algorithm C^2ICM is an extension of C^3M. First, we start with m1 documents, and cluster them using C^3M. Then, as new documents arrive, we follow these steps:

> [a] *Compute n_c and the cluster seed powers of the documents in the updated document database, $D_m = D_m \cup D_{m'} - D_{m''}$ and pick the cluster seeds. (In general $m' \gg m''$.)*
>
> [b] *Determine D_r, the set of documents to be clustered. Cluster these documents by assigning them to the cluster of the seed that covers them most.*
>
> [c] *If there were documents not covered by any seed, then group those together into a ragbag cluster.*
>
> [d] *Apply the above steps for each database update.*
>
> ACM Transactions on Information Systems, Vol. 11, No. 2, April 1993.

Extending the single-link clustering algorithm is simple. We can find the similarities between the new document to every other document and add the document to the cluster in which the item providing maximum similarity is.

7. The given paper does not cover the clustering tendency concept and refers to Dubes [1987] and Cheng [1995] for further information. Clustering tendency is the inclination of a dataset to be separated into clusters; that is, a dataset with a low clustering tendency might better be *not* represented as separate clusters. It is important because we don't want to cluster things where there is no clustered structure. We can use the sum of c_{ii} values, that is, nc; to have an idea about the clustering tendency. After all, c_{ii} are also known as the decoupling coefficients, and if they are low, then the points are not decoupled, meaning that the clustering tendency is low.

8. Since m = 150, n_c = 10; and all clusters are of the same size, then each cluster will have 15 documents. m_j = 150 – 15 = 135.
For all clusters, p_j would be equal, since all $|C_i|$ are the same; meaning that the expected target clusters will be:

$$n_{tr} = n_c p_j = n_c\left(1 - \prod_{i=1}^{k} \frac{m_j - i + 1}{m - i + 1}\right) = 10 - 10\prod_{i=1}^{5} \frac{136 - i}{151 - i} = 10 - 10\,\frac{135}{150}\,\frac{134}{149}\,\frac{133}{148}\,\frac{132}{147}\,\frac{131}{146} = 4.14$$

Then n_{tr} = 4.

9.

The similarity matrix implied by the dendogram:

$$S = \begin{bmatrix} 1.00 & 0.50 & 0.50 & 0.50 & 0.50 \\ 0.50 & 1.00 & 0.57 & 0.57 & 0.50 \\ 0.50 & 0.57 & 1.00 & 0.67 & 0.50 \\ 0.50 & 0.57 & 0.67 & 1.00 & 0.50 \\ 0.50 & 0.50 & 0.50 & 0.50 & 1.00 \end{bmatrix}$$

Variance is calculated as 0.94.

The original similarity matrix:

$$S = \begin{bmatrix} 1.00 & 0.50 & 0.50 & 0.50 & 0.50 \\ 0.50 & 1.00 & 0.57 & 0.57 & 0.50 \\ 0.50 & 0.57 & 1.00 & 0.67 & 0.50 \\ 0.50 & 0.57 & 0.67 & 1.00 & 0.50 \\ 0.50 & 0.50 & 0.50 & 0.50 & 1.00 \end{bmatrix}$$

Variance is calculated as 3.00.

Covariance of these two matrices is 1.44; therefore cov(x,y)/sqrt(var(x) var(y)) = **0.86**.

10. The total number of True Positives and True Negatives divided by the total will give us the cluster purity. That is, purity = (TP+TN) / (TP+TN+FP+FN).

Using the example given in the classroom; if our ground truth has two clusters {a,b,c} and {d,e,f}; and the constructed clusters are {a,b} {c,d} and {e,f}; then we will have TP=2, TN=8, FP=1, FN=4. Therefore purity = (2+8)/15 = **0.67**. This is true because 1/3 of the samples are misclassified.

11. The word corpus:

```
ÖDEV, UZUN, UZADI, YORGUN, YORULDUM, BİTMEK, BİTİR, BİTEN,
BİTMİYOR, NEDEN, YETER, YETMEK, YETİŞTİRMEK, YAZIK, BİZE
```

Words to analyse:
`YORGUN, YETMEK`

Prefix	**Letters**	**Successor Variety**	**Prefix**	**Letters**	**Successor Variety**
Y	A,E,O	3	Y	A,E,O	3
YO	R	1	YE	T	1
YOR	U,G	2 <- CUT HERE	YET	E,M,İ	3 <- CUT HERE
YORG	U	1	YETM	E	1
YORGU	N	1	YETME	K	1
YORGUN	-	0	YETMEK	-	0

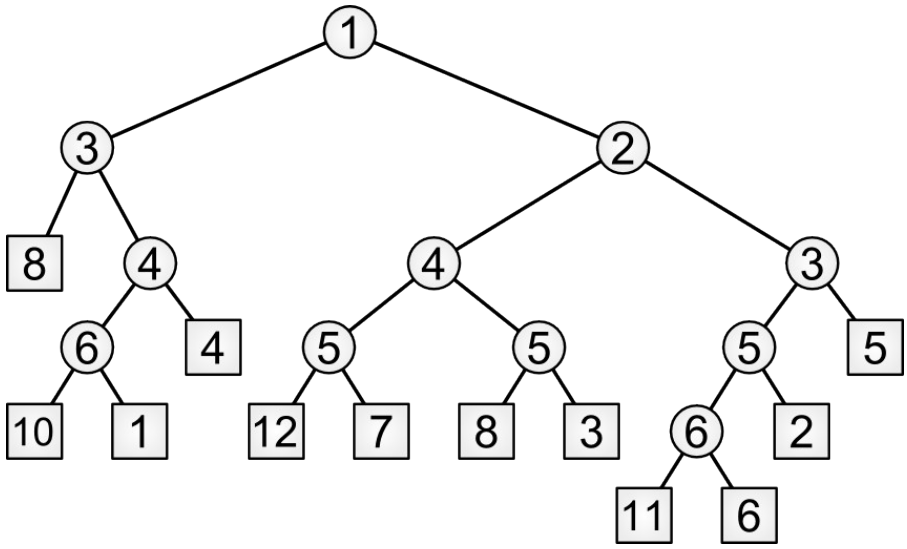
Stems obtained: `YOR, YET`

12. n-gram method seems more robust because SV requires a proper corpus (is highly prune to errors in the corpus), cannot detect prefixes (e.g.: "*na*mümkün") and it is somewhat vague where to 'cut' the word. On the other hand, n-grams are not dependent on the language, and can successfully detect prefixes.

13. Below are the sistrings (rows: sistring #, cols: bit pos.):

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
1	0	1	1	0	1	1	1	0	1	0	1	1	0	1	0	0	1	1	1
2	1	1	0	1	1	1	0	1	0	1	1	0	1	0	0	1	1	1	...
3	1	0	1	1	1	0	1	0	1	1	0	1	0	0	1	1	1	...	
4	0	1	1	1	0	1	0	1	1	0	1	0	0	1	1	1	...		
5	1	1	1	0	1	0	1	1	0	1	0	0	1	1	1	...			
6	1	1	0	1	0	1	1	0	1	0	0	1	1	1	...				
7	1	0	1	0	1	1	0	1	0	0	1	1	1	...					
8	0	1	0	1	1	0	1	0	0	1	1	1	...						
9	1	0	1	1	0	1	0	0	1	1	1	...							
10	0	1	1	0	1	0	0	1	1	1	...								
11	1	1	0	1	0	0	1	1	1	...									
12	1	0	1	0	0	1	1	1	...										

Instead of showing the trees step by step; I determined the bits that require branching as above (how: I sort the first bit, for both the 1 and 0 groups I do the sorting within, and descend recursively until each call has only 1 items. Then I remove the intermediate bits that are common for each of these cases, because we need not test them) and using this I construct the final tree at once:



Note that for every inner node, the left-hand side branch is for 0, and vice versa.