Kaan Onarlıoğlu

March 17, 2009

CS 533: Information Retrieval Systems, Assignment #1

1. **a.** TREC Interpolation Rule: For each **standart** recall value 'i' from 0.0 to 1.0 with 0.1 increments, take the maximum precision obtained at any **actual** recall value greater or equal to 'i'.

For Q1:

Total no of relevant documents: 4

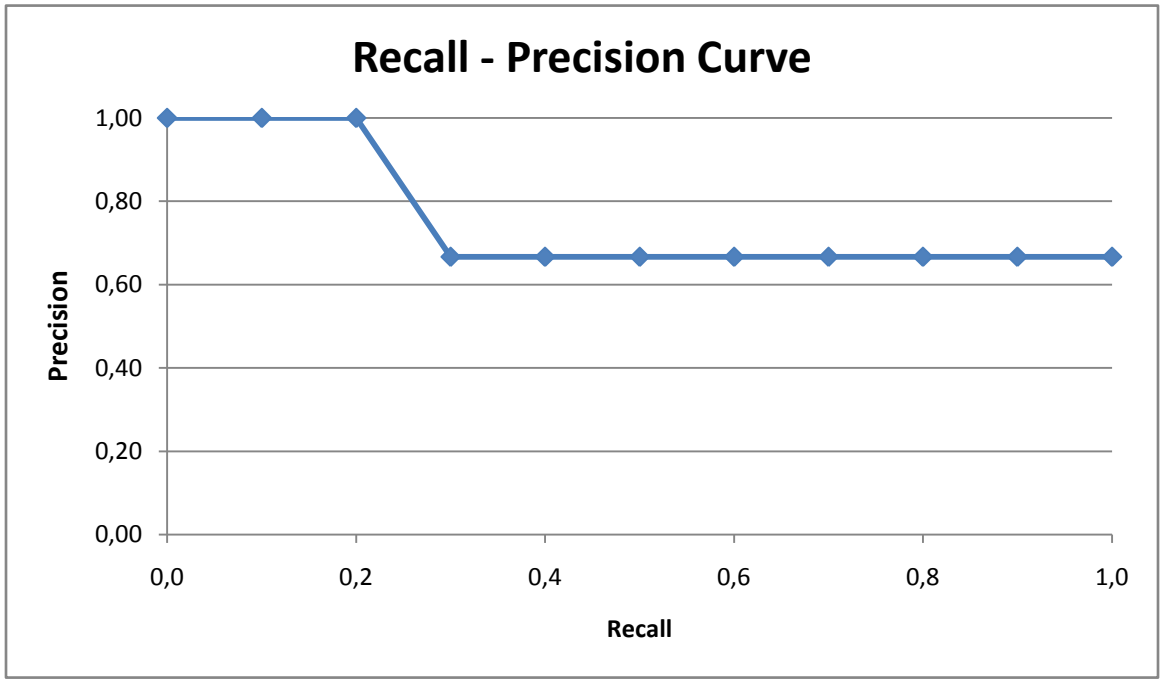
Actual Recall & Precision Table:

Rank	1	2	3	4	5	6	7	8	9	10
Relevant	Yes	No	Yes	No	Yes	Yes	No	No	No	No
Precision	1/1	1/2	2/3	2/4	3/5	4/6	4/7	4/8	4/9	4/10
Recall	0.25	0.25	0.50	0.50	0.75	1.00	1.00	1.00	1.00	1.00

Interpolated Recall & Precision Table:

Precision	1	1	1	2/3	2/3	2/3	4/6 = 2/3	4/6	4/6	4/6	4/6
Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

Recall Precision Graph:



Kaan Onarlıoğlu

March 17, 2009

For Q2:

Total no of relevant documents: 5

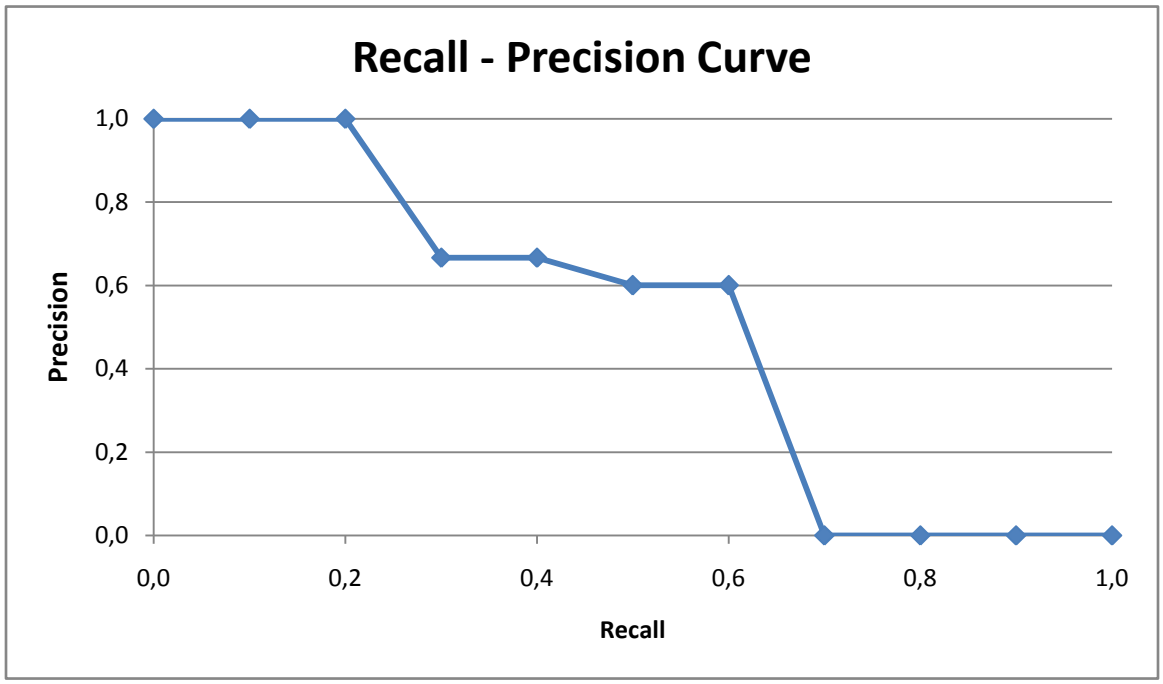
Actual Recall & Precision Table:

Rank	1	2	3	4	5	6	7	8	9	10
Relevant	Yes	No	Yes	No	Yes	No	No	No	No	No
Precision	1/1	1/2	2/3	2/4	3/5	3/6	3/7	3/8	3/9	3/10
Recall	0.2	0.2	0.4	0.4	0.6	0.6	0.6	0.6	0.6	0.6

Interpolated Recall & Precision Table:

Precision	1	1	1	2/3	2/3	3/5	3/5	0	0	0	0
Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

Recall Precision Graph:

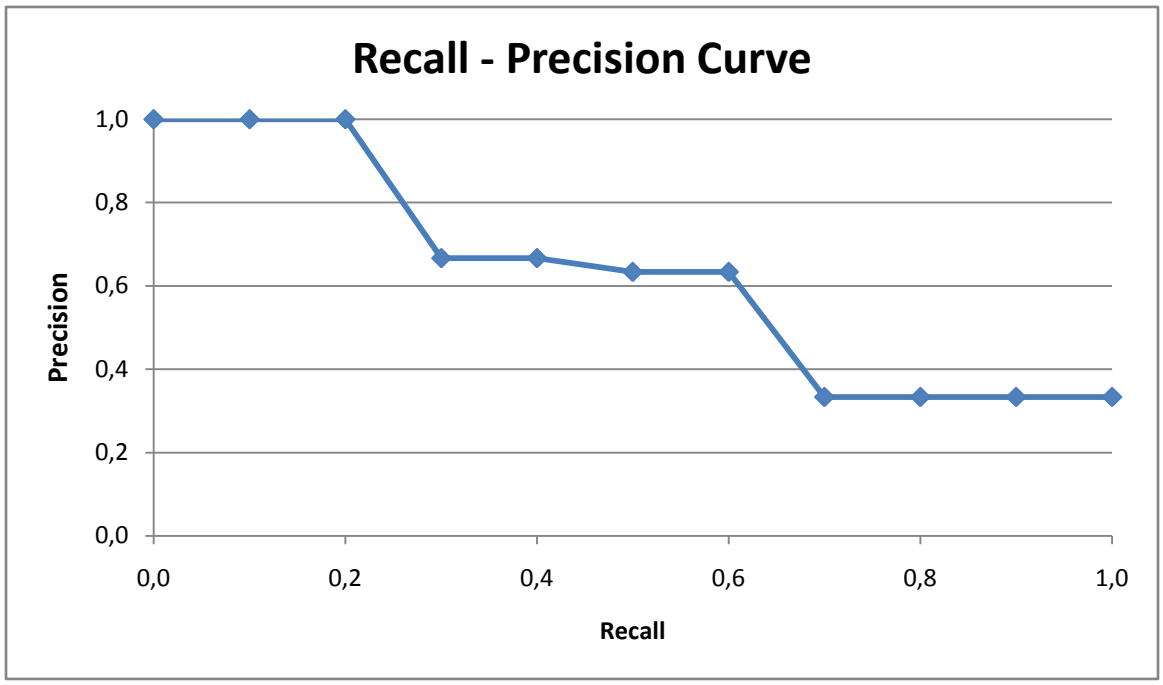


For Q1 + Q2:

Average Interpolated Recall & Precision Table:

Precision	1	1	1	2/3	2/3	19/30	19/30	1/3	1/3	1/3	1/3
Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

Recall Precision Graph:



b.

Q1: Total of 4 relevant documents.

2 relevant documents retrieved in the top 4 documents.

→ R-Precision = 2/4 = 1/2

Q2: Total of 5 relevant documents.

3 relevant documents retrieved in the top 5 documents.

→ R-Precision = 3/5

Q1 + Q2: Mean of the R-Precisions for Q1 and Q2 = $\frac{\frac{1}{2}+\frac{3}{5}}{2} = \frac{11}{20} = 0.55$

c.

Average precision for Q1 = (1 + 2/3 + 3/5 + 4/6) / 4 = 0.73

Average precision for Q2 = (1 + 2/3 + 3/5 + 0 + 0) / 5 = 0.45

MAP = (0.73 + 0.45) / 2 = 0.59

2.

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

a. The straightforward approach to construct the S matrix is to calculate the similarity between every pair of documents. Since the S matrix is symmetric and a document 'd' is similar to itself, we only need to calculate the following matrix elements:

$$S = \begin{bmatrix} 1 & s12 & s13 & s14 & s15 \\ x & 1 & s23 & s24 & s25 \\ x & x & 1 & s34 & s35 \\ x & x & x & 1 & s45 \\ x & x & x & x & 1 \end{bmatrix}$$

S_{12}, S_{13}, S_{14}, S_{15} count = 4

S_{23}, S_{24}, S_{25} count = 3

S_{34}, S_{35} count = 2

S_{45} count = 1 → Total = 10 similarity calculations

b. By using term inverted indexes to construct the S matrix, we aim to identify the documents that have at least one common term, so that the similarity between those documents is non-zero.

Consider the D matrix given above.

The resulting postings lists are as follows:

$t_1 \rightarrow d_1, d_5$ $t_4 \rightarrow d_1, d_3, d_4$ $t_7 \rightarrow d_1, d_3, d_4$

$t_2 \rightarrow d_1, d_3, d_5$ $t_5 \rightarrow d_2, d_4$

$t_3 \rightarrow d_2$ $t_6 \rightarrow d_2, d_5$

We then need to calculate the following similarity coefficients:

Consider d_1:

d_1 contains terms t_1, t_2, t_4, t_7

$t_1 \rightarrow d_1, d_5$ Calculate S_{15}

$t_2 \rightarrow d_1, d_3, d_5$ Calculate S_{13}

$t_4 \rightarrow d_1, d_3, d_4$ Calculate S_{14}

$t_7 \rightarrow d_1, d_3, d_4$ Calculate -

Consider d_2:

d_2 contains terms t_3, t_5, t_6

$t_3 \rightarrow d_2$ Calculate -

$t_5 \rightarrow d_2, d_4$ Calculate S_{24}

$t_6 \rightarrow d_2, d_5$ Calculate S_{25}

Consider d_3:

d_3 contains terms t_2, t_4, t_7

$t_2 \rightarrow d_1, d_3, d_5$ Calculate S_{35}

$t_4 \rightarrow d_1, d_3, d_4$ Calculate S_{34}

$t_7 \rightarrow d_1, d_3, d_4$ Calculate –

Consider d_4:

d_4 contains terms t_4, t_5, t_7

$t_4 \rightarrow d_1, d_3, d_4$ Calculate -

$t_5 \rightarrow d_2, d_4$ Calculate -

$t_7 \rightarrow d_1, d_3, d_4$ Calculate -

No need to consider d_5.

Total similarity coefficient calculations: 3 + 2 + 2 + 0 = 7

c. We cannot further reduce the number of similarity calculations since we must always calculate the similarity coefficients for documents that have non-zero similarity coefficients (of course, with the assumption that we cannot omit the calculation of insignificant similarity values). We can, however, improve the performance of the term inverted indexing method by organizing the posting lists in reverse order. In the previous method, we only need to consider the documents which have IDs higher than the currently processed document. Therefore, organizing the posting lists in reverse order eliminates unnecessary accesses to the indexing structure and ID comparisons with the lower ID documents.

3.

$$S = \begin{bmatrix} 1 & 0 & 0.75 & 0.40 & 0.40 \\ x & 1 & 0 & 0.20 & 0.20 \\ x & x & 1 & 0.50 & 0.20 \\ x & x & x & 1 & 0 \\ x & x & x & x & 1 \end{bmatrix}$$

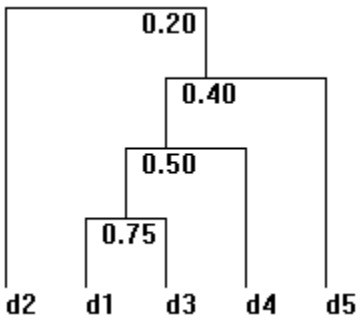


Figure 1 - Dendrogram for single-link approach

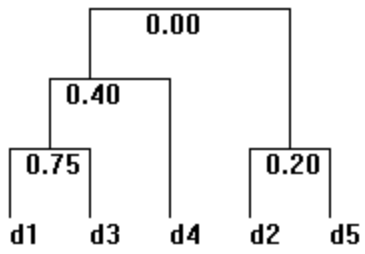


Figure 2 – Dendrogram for complete-link approach

4. The given formula is a function that could be used for calculating the weights of indexing terms for an automatic text-retrieval system:

$$w_{ij} = tf_{ij} * idf_{ij}$$

In this formula,

w_{ij} is the weight of term 'i' in document 'j', which assesses how much that term can distinguish a document from the rest of the collection

tf_{ij} is the frequency of occurrence of term 'i' in document 'j', and

idf_i is an inverse function of the number of documents in the entire collection to which term 'i' is assigned.

Provided we use an inverted index, we can efficiently incorporate the idf component to indexing during query processing. For each term in the query, we fetch the corresponding document list from the inverted index and compute idf over the list of documents in <document id, term frequency> pairs. We can also get the tf for each query term from these pairs, compute w, and retrieve the top 'r' documents that are most similar to the query.

Otherwise, if we do not have an inverted index for the document collection, we need to perform an exhaustive search of documents to compute idf; therefore it would be very inefficient to compute idf during query processing. Preprocessing the collection to pre-compute idf values would be a much better choice in this case.

5. a.

The posting lists are given as:

term-a: <1, 2> <3, 1> <9, 2> <10, 3> <12, 4> <17, 4> <18, 3>
<22, 2> <24, 2> <33, 4> <38, 5> <43, 5> <55, 3>
term-b: <28, 2> <56, 1>

Let's divide the posting lists in chunks of at most four items and sort each chunk in reverse order according to the document ID. The resulting chunks are:

term-a:
Chunk 1: <10, 3> <9, 2> <3, 1> <1, 2>
Chunk 2: <22, 2> <18, 3> <17, 4> <12, 4>
Chunk 3: <43, 5> <38, 5> <33, 4> <24, 2>
Chunk 4: <55, 3> <> <> <>

term-b:
Chunk 1: <56, 1> <28, 2> <> <>

Then our skipping scheme is as follows: Suppose we are searching for document **D**. We first compare D with the first <document_ID, frequency> pair of the first chunk. If document_ID > D, we sequentially search the current chunk; otherwise if document_ID < D, we skip the chunk and repeat the process with the next chunk.

For the given Boolean query "term-a AND term-b",

Without skipping: We need to compare each entry in the posting list of term-a with each entry in the posting list of term-b, so that we can compute the intersection.

Posting list of term-a has 13 entries.

Posting list of term-b has 2 entries.

→ We need 13 * 2 = 26 comparisons.

With skipping:

Retrieve first element of term-b list: <56, 1>

Process term-a list:

Start with Chunk 1: 56 > 10 (skip this chunk)

Process Chunk 2: 56 > 22 (skip this chunk)

Process Chunk 3: 56 > 43 (skip this chunk)

Process Chunk 4: 56 > 55 (skip this chunk)

Retrieve next element of term-b list: <28, 2>

Process term-a list:

Start with Chunk 1: 28 > 10 (skip this chunk)

Process Chunk 2: 28 > 22 (skip this chunk)

Process Chunk 3: 28 < 43 (OK)

Search the rest of Chunk 3: 28 =? 38

28 =? 33

28 =? 24

Total number of comparisons: 4 + 3 + 3 = 10

Clearly, skipping improves the performance.

If we use larger chunks, the number of chunks will be less and we will be able to eliminate more unnecessary comparisons by skipping a chunk. However, when we need to search inside a chunk, we will be doing many comparisons since there will be more elements in a chunk.

Conversely, if we use smaller chunks, we will end up with more number of chunks; which will result in smaller skip distances and more comparisons with the chunk heads; but the number of entries in a single chunk will be less and searching within a chunk will require less comparisons.

b.

Posting list of term-a ordered by $f_{d,t}$:

term-a: <38, 5> < 43, 5> <12, 4> <17, 4> <33, 4> <10, 3> <18, 3>

<55, 3> <1, 2> <9, 2> <22, 2> <24, 2> <3, 1>

Posting list of term-a ordered by frequency information in prefix form:

term-a: <5:2:38, 43> <4:3:12, 17, 33> <3:3:10, 18, 55> <2:4:1, 9, 22, 24> <1:1:3>

Storing the list in frequency-order improves the query processing since we can now process the documents with higher term-frequency values, ie. those documents that are considered more relevant to the search query, first. Then, when the frequency value drops below a threshold value, we may end the processing of the list, thus eliminating unnecessary fetches. If the posting-list is not entirely fetched but read block-by–block from the disk, the performance gain becomes more significant.

Storing the frequency-ordered list in prefix form makes it possible to eliminate repeating frequency values in the list, at the cost of introducing an overhead to store multiple document IDs in a single frequency entry. However, the combined result is a smaller size posting list than the previous document-ordered list.

6. a. In supervised classification, we are provided with a pre-classified collection of patterns and the task is to classify a newly encountered pattern according to previous knowledge based on our collection. However, in unsupervised classification such a pre-classified collection is not available, and the task is to group a set of unlabeled patterns into meaningful clusters.

Information filtering systems make use of supervised classification since they are usually trained with a collection of pre-classified data that makes it possible for the system to recognize and filter out undesired data based on filtering parameters.

b. The components of a clustering framework are:

1. Pattern representation, including feature extraction/selection: In this step the range of features available to the clustering algorithm to take into consideration are determined. Feature

selection is the process of choosing the most effective features among a set of features, and feature extraction is the process of transforming one or more of the available features to create new features. In the context of Information retrieval, documents are the patterns and they are represented by term indexes.

2. Definition of a pattern proximity measure: In this step, a function to measure the similarity, or dissimilarity as appropriate, between a pair of patterns is determined. In Information Retrieval, this step corresponds to choosing a term weighing function or a similarity measure such as the Cosine, Dice or Jacob's similarity coefficients.

3. Clustering: This is the actual step in which patterns are clustered into groups based on their similarities. The clustering process may have different results depending on the choices made in the previous steps, and on the clustering method used, such as hierarchical, probabilistic, partitional clustering. In Information Retrieval, this step corresponds to running a document clustering algorithm.

4. Data abstraction: This step is used for extracting a simple and compact representation of the formed clusters, so that further processing or understanding of the clustered data is simpler. In Information Retrieval, this step corresponds to choosing cluster representatives/centroids.

5. Assessment of output: This is the process of evaluating the results of the clustering process to decide whether the resulting clusters are meaningful and whether they conform to some pre-determined criteria such as expected statistical outcomes. It may also include comparing the effectiveness of the used clustering scheme with another scheme. In Information Retrieval, this is the cluster validation step.

c. "Cluster tendency analysis" refers to the analysis of some input data to determine whether that data domain is suitable for clustering, in other words whether clusters are present in that data set. It is carried out before the clustering process to decide whether there is any benefit in performing a clustering process on that data.

d. Data mining deals with the problem of extracting meaningful data from vast data stores; therefore clustering this data into appropriate clusters is usually one of the first steps of data mining approaches.

One of the uses of clustering in data mining is "segmentation" of databases, which means dividing large databases into homogenous groups. This is used for increasing the efficiency of processing data by working on individual segments as opposed to the full database, compression of data and identifying characteristics dominant in different segments.

Another use of clustering is "predictive modeling" for statistical analysis of data. Predictive modeling uses clustering to group data items into clusters, to characterize those groups by inferring rules on them and to suggest predictive models in order to aid a researcher in discovering potential hypotheses to test the statistical analysis performed.

Finally, clustering could be used for “visualization of data”, by grouping data sets in large databases into clusters and making it possible for researchers to identify sub-groups with similar or different characteristics.

Some practical applications of clustering in data mining include mining very large unstructured databases, such as World Wide Web, by clustering web documents, or clustering geological databases to identify oil & gas reserves and drill areas.

7. a. The Cranfield methodology is a paradigm for evaluating and comparing information retrieval technologies in a laboratory setting. Following the Cranfield methodology, different tasks and user groups for an information retrieval system are left aside; instead, a “test collection”, which is composed of a set of documents, a set of topics and a relevance judgment of documents to topics, is used for experimenting with the information retrieval system, with the sole task of ranking the relevant documents for a topic over the non-relevant ones.

b. TREC was originally established by NIST and DARPA to build a large test collection in order to evaluate an information retrieval system developed by NIST. However, today its purpose is to encourage information retrieval research by providing a forum for the researchers all around the world to communicate and exchange ideas, to support evaluation and comparison of competing information retrieval systems by building and providing large test collections and required evaluation tools, and to promote the increase of retrieval effectiveness.

c. TDT has five major tasks. These are:

1. Story segmentation, involves segmenting a data source to topically coherent subsections
2. Topic tracking, involves tracking events of interest in future stories
3. Topic detection, involves detecting stories on the same topic and grouping them together
4. First story detection, involves detecting whether a story is the first one about a new topic
5. Link detection, involves detecting whether given two stories are topically linked

8.

a. I could not derive this proof on my own since it requires knowledge and experience of advanced graph theory. Many sources refer to the book “Mathematical Taxonomy” by Nicholas Jardine and Robin Sibson as the main work related to the formal foundations of single-link clustering, which also includes the proof for order invariance of inputs to the single-link clustering algorithms; however, the book is not available online, or at the METU and Bilkent University libraries.

The following proof presented proves the order independence of the single-link clustering algorithm with the assumption that all the similarity pair values are sorted in descending order which is the

common form of application in popular implementations. For the complete proof not based on this assumption, I will refer to the mentioned book when available.

This proof is based on the proof provided by Ramazan Yılmaz but it has been modified in part since I do not completely agree on the original version of the proof, and it reflects my own interpretation of the problem and its solution.

Proof:

Let 'L' be the list of ordered similarity pair values. We shall first prove that swapping two adjacent similarity pair values in L does not affect the final dendrogram structure.

Let AB be the i^{th} pair and let CD be the $i+1^{th}$ pair in L, where A, B, C and D are not necessarily distinct documents, and AB & CD are two pairs with equal similarity values so that we can change the order of processing them. We shall show that processing these two pairs in either order results in the same dendrogram.

After processing the $i-1^{th}$ pair, one of the following cases must hold:

Case 1: A and B are already in the same cluster.

Then processing the pair AB does not require any change to the dendrogram as they are already clustered with their maximum similarity value. Therefore we only need to consider CD, in which case, it does not matter if AB or CD appears first in L. Similarly this case holds if C and D are in the same cluster.

Case 2: A = C (or A = D or B = C or B = D)

Assume that AB and CD pairs have the similarity values 'S' and we first process the pair AB. Then we first merge the clusters of A and B to form a single cluster with the similarity value 'S'. Since A = C, processing CD, is equivalent to processing a pair AD; therefore the result is a dendrogram composed of merging the clusters of A, B and D at the same similarity value S.

Now let's assume we first process CD which is equivalent to AD, i.e. we merge the clusters of A and D at the similarity value S. Then, we process AB resulting in merging the clusters of A, B and D at the same similarity value S.

In both cases, we just connect the three separate dendrograms corresponding to the clusters of A, B and C at the similarity value S; therefore the resulting structure is independent of the processing order.

Case 3: $A \neq C$ and $A \neq D$ and $B \neq C$ and $B \neq D$

Then, the two clustering operations for processing AB and CD are operations performed on two separate dendrogram pieces (or sub-trees) that are not yet connected at a common similarity value. Therefore, the processing order does not matter.

=> Processing AB and CD in either order results in the same dendrogram structure.

=> Swapping adjacent similarity pairs does not affect the final dendrogram structure.

Since we can obtain any ordering of the similarity pairs by a number of subsequent adjacent pair swaps (proof is trivial and is omitted), and since swapping adjacent pairs does not affect the cluster structure, single-link cluster method is order independent.

b. Consider the similarity matrix:

$$S = \begin{bmatrix} 1 & 0.60 & 0 & 0 \\ x & 1 & 0.60 & 0 \\ x & x & 1 & 0.40 \\ x & x & x & 1 \end{bmatrix}$$

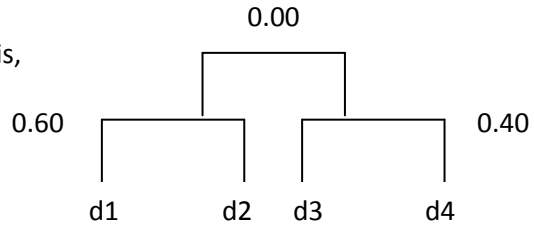
Let's consider the non-zero similarity coefficients in this order:

1) $S_{12} = 0.60$

2) $S_{23} = 0.60$

3) $S_{34} = 0.40$

Then, the resulting dendrogram is,



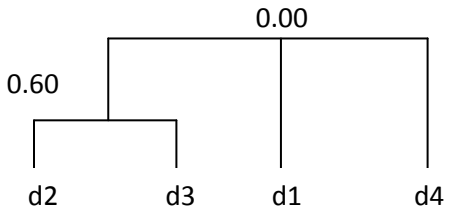
However, if we process the same coefficients in a different order, as follows,

1) $S_{23} = 0.60$

2) $S_{12} = 0.60$

3) $S_{34} = 0.40$

Then, the resulting dendrogram is,



Clearly, the complete-link clustering method is order-dependent.

9. a. There are several types of classification:

1. Based on the classification task:

   i. Subject classification: Documents are classified according to their topics

   ii. Functional classification: Documents are classified according to their roles

   iii. Sentiment classification: Documents are classified according to the point of view presented

   iv. Many other application specific classifications, e.g. genre, spam classification

2. Based on the number of classes available:

   i. Binary classification: There are only two classes to classify the documents

   ii. Multi-class classification: There are more than two classes available for the classification

3. Based on the number of classes each document can be assigned to:

   i. Single-label classification: Each document can be assigned to exactly one class

   ii. Multi-label classification: Each document can be assigned to one or many classes

4. Based on the type of class assignment:

   i. Hard classification: Each document is either assigned to a class or not; there cannot be an intermediary state

   ii. Soft classification: Each document is predicted to be in a class with a probability of assignment

5. Based on the organization of the classes:

   i. Flat classification: All the classes are considered to be at an equivalent level; there is no hierarchical organization among them

   ii. Hierarchical classification: Classes are organized as a tree-like structure; classes may have sub-classes

b. Web classifications are useful in several application areas.

1. Web directories that are mostly maintained manually by human editors can be automatically generated and updated by creating classifiers using user specified hierarchies.

2. The quality of web search results could be enhanced by classifying web documents and providing specific search classes with the queries. Additionally, results of regular searches can be presented in categories for a more efficient human-computer interface.

3. The performance of question answering systems could be increased by classifying the retrieved web documents using functional classification techniques to identify topic pages that may contain the relevant answers.

4. Focused crawlers, in other words crawlers that only visit pages relevant to a specific topic, can make use of classifiers to evaluate the relevance of the pages for crawling.

5. Web content filtering, assisted web browsing and contextual advertising applications can be enhanced by using web classification.

c. According to the authors, Web classification and text classification differ in the following three aspects:

1. While regular text documents are usually well-structured and written in a consistent style, Web documents need not have any such structure.

2. Web documents contain HTML markup for formatting and rendering purposes, whereas text documents do not.

3. Web documents are linked to many other documents through hyperlinks, which establish the “web” structure. Such a property is not common in text documents.