IR ENDSEM

SHWETA SOOD
2012164

Q1

Latent Semantic Indexing is a popular technique that uses a partial singular value decomposition (PSVD) to overcome problems of vector space representation such as synonymy – vector space representation fails to capture the relationship between synonyms (underestimating the similarity user perceives) and polysemy – vector space representation fails to represent that a term could have multiple meanings (overestimating the similarity the user perceives). We use a k-dimensional LSI representation of the matrix to overcome such problems.

Traditional LSI based approaches require us to calculate the entire PSVD again from scratch, which is a time consuming. To avoid this, incremental update of LSI is preferred.

One of the approaches is to use **folding-in** : PSVD of a term-document matrix is represented as $U_k \sum_k V_k^T$. It only updates the U_k matrix (term matrix) and the $\sum_k$, V_k^T (document matrix) are left unchanged. But the addition of new documents may require the update of $\sum_k$, V_k^T as well.

Instead, we look at a new **folding-up** approach that uses a combination of folding-in and updating at each increment in order to reduce the computational cost of updating without significantly degrading the results. The idea behind folding-up is to fold-in documents until the number of folded in documents reaches a pre-selected percentage of the current term-document matrix. If no updates have previously been done, the current term-document matrix is the initial matrix; otherwise it is the last updated term-document matrix. Once the number of documents that have been folded-in reaches the pre-selected percentage of the current matrix, the vectors that have been appended to V_k during folding-in are discarded. The PSVD is then updated to reflect the addition of all the document vectors that have been folded-in since the last update. These document vectors are then discarded. The process continues with folding-in until the next update.

Presenting the design and implementation of an algorithm for incrementally update of LSI:

Constructing the updated term document matrix X_2 (t+p) x (d+q) from the original term-document matrix X_1 (txd):

For terms

- The terms that are retained from the original matrix are positioned in the same row as the original
- The term that was deleted from the X_1 is still positioned in the same row, but the corresponding cells in the columns will be filled by zeros.
- A new term that is added is positioned below the rows of existing and deleted terms

For documents

- A document that is retained is in the same column as the original matrix.
- A renamed document also occupies its original position as in X_1
- A deleted document is placed in the same column as X_1 but it will be filled by zeros.
- An added document is positioned to the right of all columns of existing and deleted documents

Explaining the algorithm

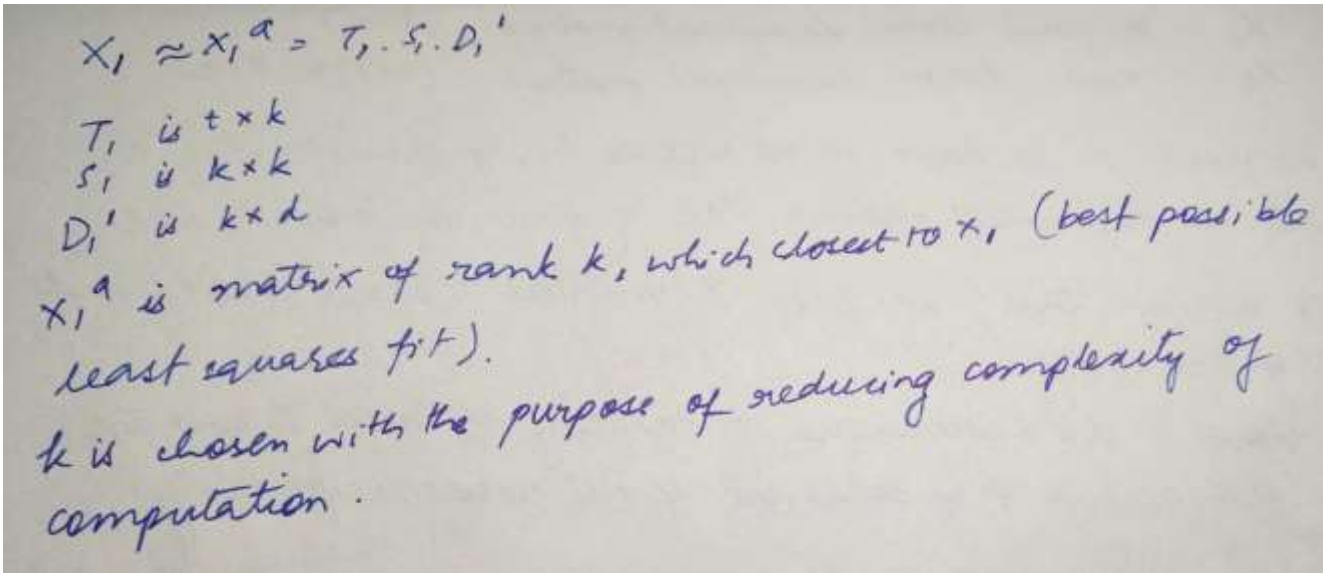
The original term-document matrix X_1 can be decomposed into a product of 3 matrices using **SVD**:

$$\mathbf{X_1 = T_{01} \cdot S_{01} \cdot D_{01}'} \qquad \textbf{(1)}$$

S_{01} is the diagonal matrix of singular values. T_{01} is the matrix of eigenvectors of the square symmetric matrix $X_1 \cdot X_1'$ (X_1' is the transpose of X_1). T_{01} is unitary. Since $X_1 \cdot X_1'$ is real, T_{01} has orthonormal columns and rows, i.e., $T_{01}' \cdot T_{01} = I$ and $T_{01} \cdot T_{01}' = I$. Meanwhile, D_{01} is the matrix of eigenvectors of $X_1' \cdot X_1$. Similarly, D01 also has orthonormal columns and rows, i.e., $D_{01}' \cdot D_{01} = I$ and $D_{01} \cdot D_{01}' = I$. Note that T_{01} is a $t \times m$ matrix, S_{01} is a $m \times m$ matrix, and D_{01}' is a $m \times d$ matrix ($t, d, m \in N$).

If the singular values in S_{01} can be ordered by size, the first k largest may be kept and the remaining smaller ones set to zero. Since zeroes were introduced into S_{01}, the representation can be simplified by deleting the zero rows and columns of S_{01} to obtain a new diagonal matrix S_1, and then by deleting the corresponding columns of T_{01} and D_{01} to obtain T_1 and D_1 such that

$$\mathbf{X_1 \approx X_1^a = T_1 \cdot S_1 \cdot D_1'} \qquad \textbf{(2)}$$



X_1 – original term document matrix $(t \times d)$

X_2 – new term document matrix $(t+p) \times (d+q)$

We want X_1 to have same size as X_2, by transforming X_1 into an expanded matrix X_{1e} of dimension $(t+p) \times (d+q)$

To achieve this, we take 2 matrices U of size $(t+p) \times t$ and V of size $d \times (d+q)$.

U and V are extensions of Identity matrices I_t and I_d with p rows & q columns of all zeros, respectively.

Now, to construct $X_{1e} = U \cdot X_1 \cdot V$

But from (1) this can be written as -

$X_{1e} = U \cdot X_1 \cdot V = U \cdot T_{01} \cdot S_{01} \cdot D'_{01} \cdot V$

$= (U \cdot T_{01}) \cdot S_{01} \cdot (V' \cdot D_{01})'$

$$\boxed{X_{1e} = T_{01e} \cdot S_{01} \cdot D'_{01e}} \quad - (3)$$

where $T_{01e} = U \cdot T_{01}$, $D_{01e} = V'$

$T_{01e} \cdot T'_{01e} = (U \cdot T_{01}) \cdot (U \cdot T_{01})' = U \cdot T_{01} \cdot T'_{01} \cdot U' = U \cdot I \cdot U' = U \cdot U' = I$

Similarly $T'_{01e} \cdot T_{01e} = I$, $D_{01e} \cdot D'_{01e} = I$, $D'_{0e} \cdot D_{01e} = I$

$\Rightarrow$ We can compute an SVD for X_{1e}

From SVD of X_{1e}, we can perform reduction by keeping k largest singular values as in X_1.

$$\boxed{X_{1e} \approx X^a_{1e} = T_{1e} \cdot S_1 \cdot D'_{1e}} \quad - (4)$$

$\therefore$ SVD of X_{1e} & X^a_{1e} is obtained.

$\therefore$ X_{1e} & X_2 have same size of $(t+p) \times (d+q)$

$\therefore$ $X_2 = X_{1e} + A$ where A is modification matrix

We can represent $X_2 = X_{1e} + A$ as X_2 & X_{1e} are of same dimensions.

$$\boxed{P = T'_{1e} \cdot X_2 \cdot D_{1e}} \quad -(5)$$

$T'_{1e} \rightarrow k \times (t+p)$ matrix

$X_2 \rightarrow (t+p) \times (d+q)$ matrix

$D_{1e} \rightarrow (d+q) \times k$ matrix

P's dimension is $k \times k$.

Applying SVD on P, we get $P = T'_{1e} \cdot X_2 \cdot D_{1e} = T_{temp} \cdot S_2 \cdot D'_{temp}$

$T_{1e} \cdot T'_{1e} \cdot X_2 \cdot D_{1e} \cdot D'_{1e} = T_{1e} \cdot T_{temp} \cdot S_2 \cdot D'_{temp} \cdot D'_{1e}$

(orthonormal)

$\Rightarrow X_2 = T_{1e} T_{temp} \cdot S_2 \cdot D'_{temp} \cdot D'_{1e}$

$\Rightarrow \boxed{X_2 = T_{02} \cdot S_2 \cdot D'_{02}}$ $-(6)$

where $T_{02} = T_{1e} \cdot T_{temp}$

$D_{02} = D_{1e} \cdot D_{temp}$

Incremental Update of LSI –

(1) Obtain X_2 as described

(2) Compute $T_{01e} = U \cdot T_{01}$ & $D_{01e} = V'$.
Derive T_{1e}, D_{1e}

(3) Compute $P = T'_{1e} \cdot X_2 \cdot D_{1e}$.

(4) Perform SVD to get $P = T_{temp} \cdot S_2 \cdot D'_{temp}$ (P is $k \times k$)

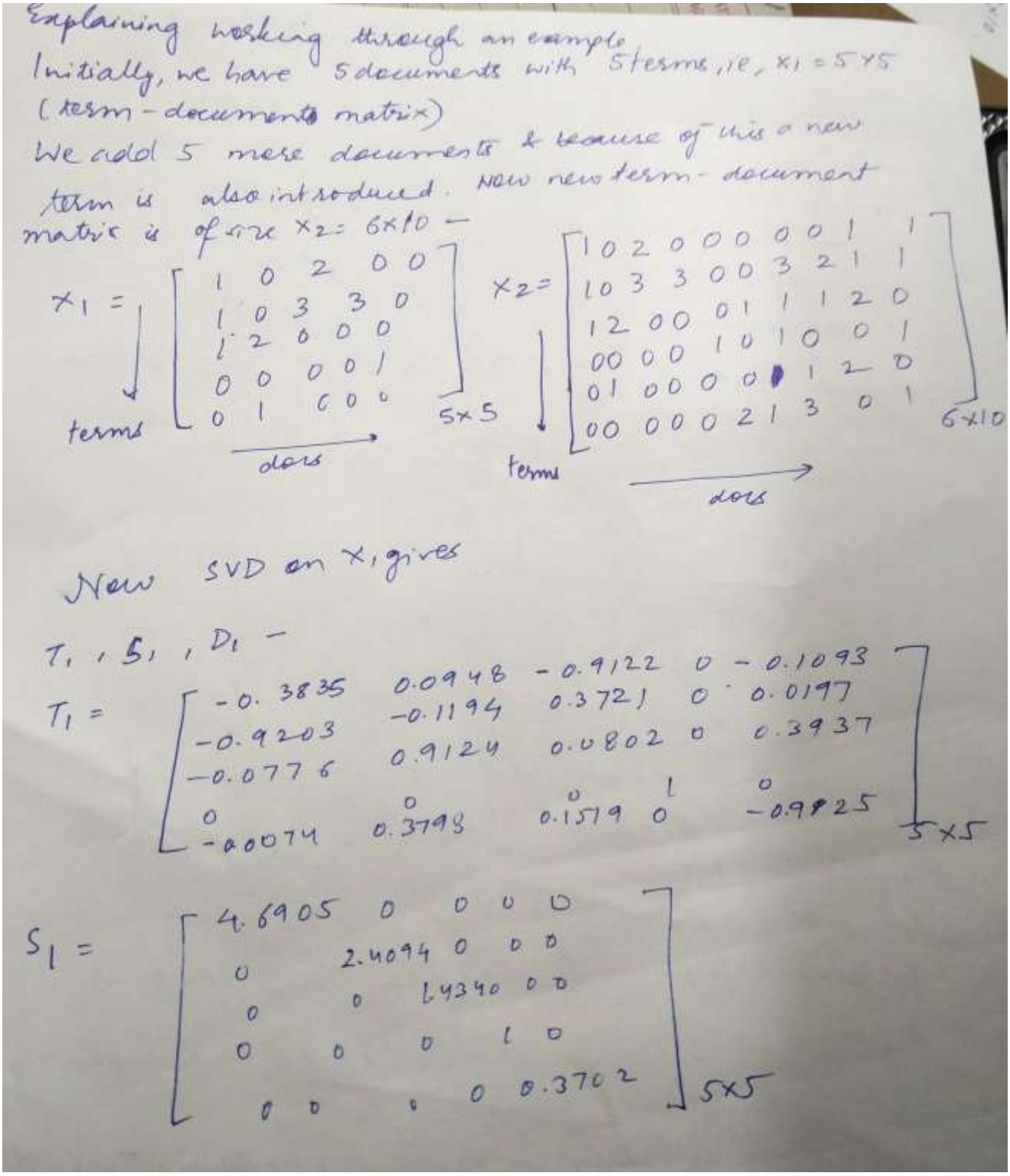
(5) Compute T_{02}, D_{02}. Derive T_2 & D_2 by the reduction step.

(6) $X_2^a = T_2 \cdot S_2 D'_2$.

Thus, we have found out a new decomposition T_2, S_2, D'_2 for $X_2^a \approx X_2$.

This is the incremental update of the LSI that saves time by not requiring one to reconstruct the SVD from scratch.

Explaining the working of the algorithm through an example of 5 documents, showing how adding 5 more documents change the document and term vectors. [**Code is attached in folder for calculations shown**] :



$$D_1 = \begin{bmatrix} -0.2945 & 0.3685 & -0.3206 & 0 & 0.8214 \\ -0.0346 & 0.9150 & 0.2179 & 0 & -0.3378 \\ -0.7521 & -0.0700 & -0.4937 & 0 & -0.4309 \\ -0.5886 & -0.1487 & 0.7785 & 0 & 0.1596 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}_{5\times5}$$

Constructing U & V - Identity matrices -

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{6\times5} \quad V = \begin{bmatrix} 1 & 0 & \cdots & & \\ 0 & 1 & 0 & \cdots & \\ 0 & 0 & 1 & 0 & \cdots \\ 0 & 0 & 0 & 1 & 0 \cdots \\ 0 & 0 & 0 & 0 & 1 \ 0 \cdots \end{bmatrix}_{5\times10}$$

$X_{1e} = U \cdot X_1 \cdot V$

$$X_{1e} = \begin{bmatrix} 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 3 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Finding SVD of $X_{1e} = T_{01e} \cdot S_{1e} \cdot D'_{01e}$

$$F_{01e} = \begin{bmatrix} -0.3835 & -0.0948 & 0.9122 & 0 & -0.1093 & 0 \\ -0.9203 & 0.1194 & -0.3721 & 0 & 0.0197 & 0 \\ -0.0776 & -0.9124 & -0.0802 & 0 & 0.3937 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ -0.0074 & -0.3798 & -0.1519 & 0 & -0.9125 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{6\times6}$$

$$S_{1e} = \begin{bmatrix} 4.6905 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 2.4094 & 0 & 0 & 0 & 0 & \cdots & \\ 0 & 0 & 1.4340 & 0 & 0 & 0 & \cdots & \\ 0 & 0 & 0 & 1 & 0 & 0 & \cdots & \\ 0 & 0 & 0 & 0 & 0.3702 & 0 & \cdots & \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

all 0s.

$$Doie = \begin{bmatrix} -0.2945 & -0.3685 & 0.3206 & 0 & 0.8214 & 0 & \cdots & 0 & \cdots \\ -0.0346 & -0.9150 & -0.479 & 0 & -0.3378 & 0 & 0 & 0 & \cdots \\ -0.7521 & 0.0700 & 0.4937 & 0 & -0.4304 & 0 & 0 & 0 & \\ -0.5886 & 0.1487 & -0.7785 & 0 & 0.1596 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 1 & 0 \cdots \\ \vdots & \vdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 1 \end{bmatrix}_{10\times 10}$$

Choosing $k = 4$

$$Tie = \begin{bmatrix} 0 & 0 & 0 & -1 \\ -0.0074 & -0.3798 & -0.1519 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{6\times 4}$$

$$Sie = \begin{bmatrix} 4.6905 & 0 & 0 & 0 \\ 0 & 2.4094 & 0 & 0 \\ 0 & 0 & 1.434 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4\times 4}$$

$$Die = \begin{bmatrix} -0.2945 & -0.3685 & 0.3206 & 0 \\ -0.346 & -0.9150 & -0.2179 & 0 \\ -0.7521 & 0.0700 & 0.4937 & 0 \\ -0.5886 & 0.1487 & -0.7785 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}_{10\times 4}$$

$$Tie = \begin{bmatrix} -0.3835 & -0.0948 & 0.9122 & 0 \\ -0.9203 & -0.1194 & -0.372 & 0 \\ -0.0776 & -0.9124 & -0.0802 & 0 \\ 0 & 0 & 0 & -1 \\ -0.074 & -0.0948 & -0.1379 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{6\times 4}$$

$$P = \begin{bmatrix} 4.6905 & -1.11\times e^{-15} & -4.44e^{-16} & 0 \\ 1.387e^{-16} & 2.4094 & -4.1633e^{-16} & 0 \\ 2.2204e^{-16} & -8326e^{-16} & 1.4340 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4\times 4}$$

Now SVD of P

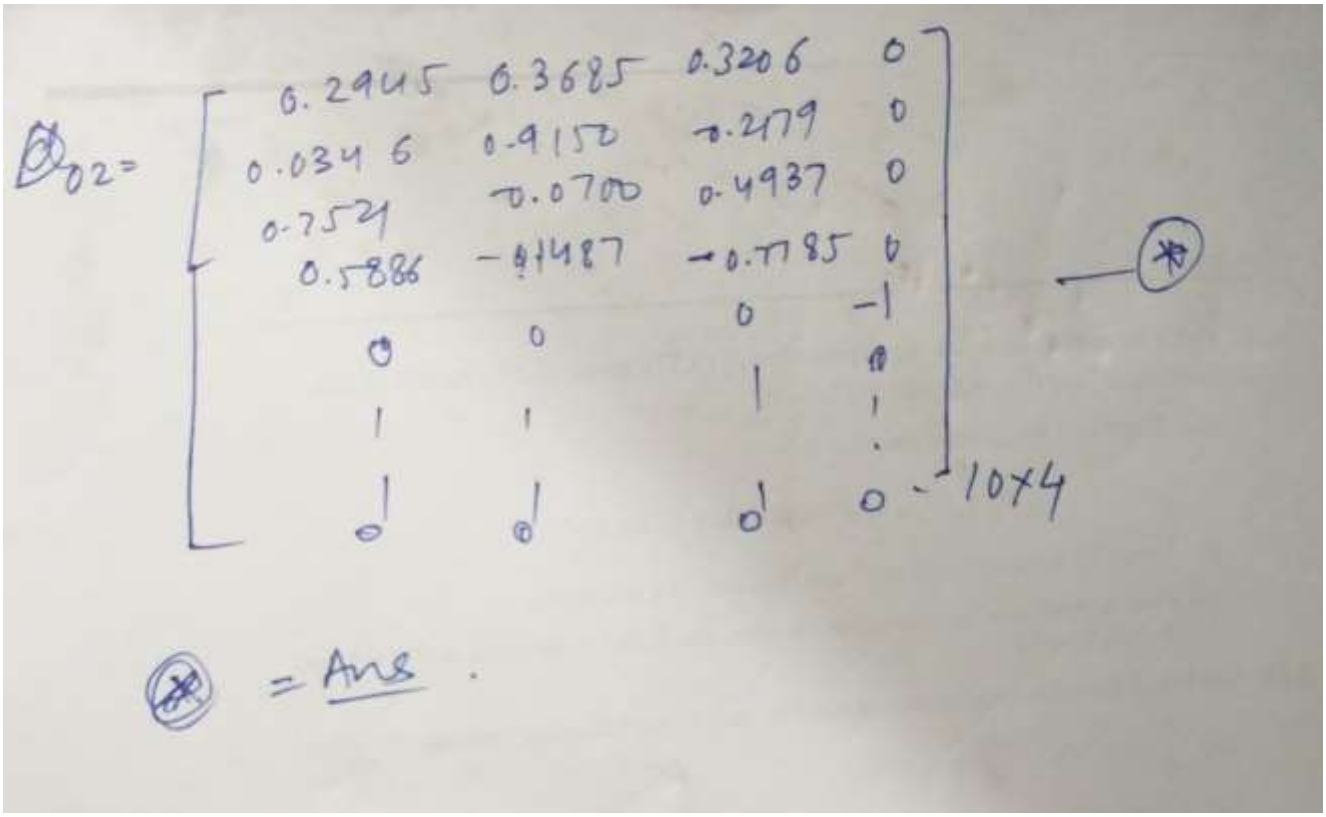
Ttemp S2 Dtemp

$$T_{temp} = \begin{bmatrix} 1 & 2.958e{-17} & -4.7339e{-17} & 0 \\ -2.95e{-17} & 1 & 5.8948e{-16} & 0 \\ -4.7339e{-17} & 6.903e{-16} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4\times4}$$

$$S_2 = \begin{bmatrix} 4.6905 & 0 & 0 & 0 \\ 0 & 2.4094 & 0 & 0 \\ 0 & 0 & 1.424 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4\times4} \quad \text{(*)}$$

$$D_{temp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 5.54e{-16} & 0 \\ 0 & 5.551e{-16} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4\times4}$$

$$T_{02} = \begin{bmatrix} 0.3835 & 0.0448 & 0.9122 & 1 \\ 0.9203 & -0.1194 & -0.3724 & 0 \\ 0.0776 & 0.9124 & -0.0802 & 0 \\ 0 & 0 & 0 & -1 \\ 0.0074 & 0.3798 & -0.1579 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{6\times4} \quad \text{(*)}$$



Q2

Various Clustering techniques can be useful in Information Retrieval Tasks:

- **Search Result Clustering** – One of the most useful applications in Information Retrieval based systems is to cluster the search results of user query. For example, If the user searches for ‘apple’ on Google it should not only display the fruit apple but the Apple brand and cluster similar results together. This will speed up the lookup and user will be able to identify the required information.
- **Scatter-Gather** – Clusters the whole collection to get groups of documents that the user can select or gather. These selected groups are further merged and result is clustered. Result is repeated until a cluster of interest is found. This helps in searching without typing by providing alternative user interface.
- **Cluster-based navigation** – Many a times, users prefer browsing over searching when they are unsure of which search terms to use. This is an alternative to user-mediated scatter-gather technique that requires computing a static hierarchical clustering of a collection that is not influenced by user interactions.
- **Cluster hypothesis to improve search results** – Using standard inverted index to identify an initial set of documents that match the query, that are merged with other documents from the same cluster. Example, car matches to automobile, vehicle. So, apart from taking documents containing the term car, we use documents that contain the related words like vehicle, automobile.
- **K-means based clustering for improved web search** – User’s query goes to a meta search engine and retrieves the relevant documents. It also uses the terms in the query to find their co-occuring terms from Wikipedia or related sites, and compares each co-occurent term in the document. The count of

co-occurent terms helps to weight the document. Next, a knowledge base for clustering is formed as a meta-directory tree with the obtained search results. Using the meta-directory tree, clustering is performed with K-Means. We initialize the number of clusters and assign thresholds for each. Then, calculate the weight of a web-page using outlinks and inlinks in the page. Use a threshold to assign the webpages to clusters, and repeat until all pages are clustered. Finally, the cluster with highest threshold is returned. If a node has more number of relevant links, it is treated as the centroid of the cluster. The centroid value is determined by considering the weight of a document.

- **Language Modelling** – Language modelling is a probability distribution over sequence of words. Given such a sequence, it assigns a probability to the whole sequence. Clustering helps to avoid sparse data problems in the language modeling approach. The model of a document d can be interpolated with a collection model. If the collection model contains many documents with terms not in d, the collection model can be replaced by a model derived from d's cluster and get an accurate representation of the probability distribution of terms in d.

These techniques can help improve performance over the traditional IR tasks based on inverted index:

- In standard IR systems, search results are presented in a simple list that user has to scroll down all the way to find the required result. Search result clustering clusters the results so that similar documents appear together.
- Instead of searching for keywords, cluster based navigation is useful in scenarios where they are unsure of what search terms to use and hence prefer browsing over searching.
- Clustering is well suited for access to a collection of news stories because news stories is not really search, but rather a process of selecting a subset of stories about recent events.
- Clustering helps to speed up search. Inverted index supports fast nearest-neighbor search for the standard IR setting. But we may not be able to use this inverted index efficiently, like in latent semantic indexing – synonymy : we cannot identify the documents containing synonyms to the query term which may be useful in providing relevant information ; polysemy : we cannot identify the context/ meaning in which the term is used in the document. Using cluster hypothesis, we can find the clusters closest to the query and only consider the documents from these clusters. Within this smaller set, we can compute similarity and rank documents in the usual way. There are fewer clusters than the documents, making the process of finding the closest cluster fast. The documents matching the query would be in the same cluster, reducing the search time over traditional inverted index.