Latent Semantic Indexing

Stuti Chaturvedi
202161006

Haripriya Goswami
202161003

Yashita Vajpayee
202162012

***Abstract*—In this experiment, we have experimented and hence learned about Singular Value Decomposition and consequently Latent Semantic Indexing.**

I. Introduction : Latent Semantic Indexing

Latent semantic indexing (LSI) or Latent Semantic Analysis (LSA) is a method of dimentionality reduction. It is a process of examining a collection of documents for exploring statistical co-occurrences of words that seem concurrently which then provide understandings into the subjects of those words and documents.
The two major problems that can be solved by LSI are:

1) *Polysemy*: It refers to Terms and Phrases that have more than one meaning. For instance, “minute” can mean offset of an hour as well as it can mean size or shape.
2) *Synonymy*: Multiple words that direct to the same meaning.

With the help of LSI, a machine can better know how to answer a query by accurately linking the appropriate keywords to the search query by comprehending how words occur together.

II. Singular Value Decomposition

SVD is one of the dimensionality reduction techniques (similar to Principal Component Analysis, which can only be applied to square matrices) that can be applied on any matrix. It accepts a matrix A as an input and characterizes it as Aˆ in a lower-dimensional space in order to minimize the “distance” between 2 matrices (as measured by the 2-norm):

$$\Delta = \left\| A - \hat{A} \right\|_2$$

Then, it projects the n-dimensional space onto a k-dimensional space where n ¿¿ k.

There are many different mappings from higher-dimensional to lower-dimensional spaces. LSI chooses the mapping which is optimal such that it minimizes the distance .

The SVD projection is computed by decomposing the document-by-term matrix At×d into the product of three matrices, Tt×n, Sn×n, Dd×n (There can be only one possible decomposition of any given matrix):

$$A_{t\times d} = T_{t\times n} S_{n\times n} (D_{d\times n})^T$$

where t is the number of terms, d is the number of documents, $n = \min(t,d)$, T and D have orthonormal columns, i.e. $TT^T = D^T D = I$, $rank(A) = r$, $S = diag(\sigma_1, \sigma_2, ..., \sigma_n,)$, $\sigma_i > 0$ for $1 \le i \le r, \sigma_j = 0$ for $j \ge r+1$.

III. Approach and Implementation

A. Creating term-document matrix

```
In [ ]: vectorizer = CountVectorizer()
        td = vectorizer.fit_transform(corpus)

        vectorizer.get_feature_names_out()  #vocabulry in ascending order

Out[ ]: array(['000', '01', '02', ..., 'zero', 'zone', 'zoom'], dtype=object)

In [ ]: td_mat = td.T.toarray()
        print("Term-document matrix\n", td_mat)

Term-document matrix
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

B. Decomposing the matrix into SVD (singular matrices)

```
In [28]: #SVD of the Term-Document matrix
         [u,s,vt] = svd(td_mat,False)
         print(u)
         print(s)
         print(vt)

[[-3.28147049e-03 -2.20692233e-04  3.30412749e-04 ...  2.60801001e-02
   4.26681604e-02  8.06280165e-03]
 [-3.06009282e-03  6.18691577e-03 -4.21341452e-03 ...  7.64734320e-03
  -6.08814238e-03  2.05914210e-02]
 [-1.26428758e-03  1.38508936e-03 -2.03424063e-03 ... -1.26947786e-02
   4.79556231e-02  3.03599277e-02]
 ...
 [-8.25039625e-03  3.04790653e-03  1.09283178e-03 ... -1.12757026e-17
   1.11889664e-16 -6.86299975e-17]
 [-1.14908949e-04 -5.69106190e-04  1.68572165e-03 ... -7.56172836e-03
   1.24016603e-02 -1.65962883e-02]
 [-2.43490601e-04  7.63616206e-04 -2.74698607e-04 ... -5.03416569e-02
   2.66991809e-02  4.23238775e-03]]
[5.99885869e+01 2.58684822e+01 2.44801670e+01 ... 4.23050503e-15
 4.23050503e-15 4.23050503e-15]
[[-3.70907376e-02 -1.98079458e-02 -1.52261733e-02 ... -1.77321570e-02
  -1.40711215e-02 -2.25736772e-02]
 [ 5.37071629e-03 -1.39656033e-02 -6.60808257e-02 ...  2.43348548e-02
   2.17260456e-02 -6.94974656e-04]
 [-1.91540433e-02  1.81860034e-02  1.20911862e-02 ...  1.84691917e-02
  -8.35401276e-03 -2.09918924e-02]
 ...
 [ 0.00000000e+00 -4.24432877e-17  4.20579839e-04 ...  2.68146686e-03
   2.06956410e-03  5.84860640e-03]
 [ 0.00000000e+00 -1.14503639e-17  1.17067952e-03 ... -6.31366665e-03
   5.35265198e-03  1.47114354e-03]
 [ 0.00000000e+00 -6.64913681e-17  2.30265643e-03 ... -6.00631807e-03
   8.53840335e-03  1.10476615e-03]]
```

C. Vectorizing query

```
In [29]:
#Transforming the query into a vector
q = vectorizer.transform(query)
q = q.toarray()
print(q)

[[0 0 0 ... 0 0 0]]
```

D. Computing new query vector from decomposed matrices

```
In [30]:
#Finding the new query vector
q = np.matmul(np.matmul(q,u),np.linalg.inv(np.diag(s)))
print(q)

[[-0.02320735 -0.00431524  0.04513855 ...  0.0389036   0.01993874
  -0.06482973]]
```

E. Finding the most similar document by taking inner product of query vector with the right singular matrix

```
In [31]:
#finding the most similar document for the query vector by taking the dot product between query and
#each column in the vt matrix and normalize them
similarities = []
for i in range(len(vt)):
    similarities.append(list(np.dot(q,vt[:,i])/(np.linalg.norm(q)*np.linalg.norm(vt[:,i])))[0])

print('Most similar document:')
idx = similarities.index(max(similarities))
corpus[idx]

Most similar document:
Out[31]:
'base pressure at subsonic speeds in the presence of\na supersonic jet .'
```

Further reference codes can be found at https://github.com/ir4h/LAB-CS653-2021.

IV. Conclusion

A. Advantages

1) Latent Semantic indexing ensures that we get better representation of document and queries.
2) If we use reduced representation of Latent semantic indexing ,noise is reduced.
3) The same underlying concept can be represented by using different words
4) The Factors of Latent Semantic Indexing are orthogonal such that definitions and terms are located in a reduced space and they shows the correlations among the usage across the documents

B. Disadvantages

Storage

Representation of SVD is more dense and documents contains more unique terms .So sparse vector representation will take more storage space than compact SVD representation

Efficiency

In Latent Semantic Indexing ,query needs to be compared with every documents present If there are more terms in the query representation in the LSI vector space, then inner product similarity scores will take more time to compute in term space.

Computation Overhead

It takes more time and there are a lot of challenges in handling such a large matrix and become exponentially more time consuming with the increase in the count of words

V. References

Introduction to information retrieval by Christopher D Manning Prabhakar Raghavan Hinrich Schutze

Elastic, 2021. Free and open search: The creators of Elasticsearch, Elk Kibana. Elastic. Available at: https://www.elastic.co/