Natural Language Processing with Latent Semantic Analysis in Python

Maalik Matthews, Jalen Roberson, Jon Scott Smith

***Abstract* – As more documents written in natural language become available in the digital world the problem of how to automate analysis of these documents grows proportionally. A way to computationally determine similarity between documents and cluster those documents together is needed in order to aid automated retrieval of similar documents in, for example, a search engine query. We identified Latent Semantic Analysis (LSA) as a simple yet effective means of finding these similarities between documents and grouping those documents accordingly. We then researched the details of this method, constructed our own term-document matrix from a set of Wikipedia articles, and used our own Python implementation of latent semantic analysis to group similar articles. To construct the term-document matrix we used Term Frequency–Inverse Document Frequency (TF-IDF) to assign weights to each word-document combination. This method helps find words which help distinguish documents and filter out words which appear often in all documents and do not signal similarity. With the resulting term-document matrix we performed a Singular Value Decomposition (SVD) to break it into 3 matrix factors which represent the terms as related to latent concepts, latent concepts as related to each document, and the significance of each latent concept. These term and document SVD factors were clustered using k-means and cosine similarity. Finally, we truncated the resulting matrices and multiplied to produce a lower rank approximation of the original term-document matrix.**

I. Introduction

A big problem in the history of data science has been the automated retrieval of information from data in the form of written natural human languages. Human beings have developed a highly complex and nuanced form of communication which is mostly optimized for another human being to comprehend when reading or writing it, but is highly unoptimized for a computer to easily process and generate. While some of the properties of the component parts of this natural language data can be more easily specified (the meaning of a word for example), more complex properties emerge when forming collections of these relatively simple component parts. How can a computer, for example, determine that rush and sprint can be considered to have the same meaning within certain contexts? This is the problem of synonymy. How can a computer determine that the word “crush” can have completely different meanings if used in the context of an industrial accident vs. a romantic relationship? This is the problem of polysemy. As digital information processing grew in practicality with the advent of increasingly faster computers, mass use of the internet, and mass digitization of written documents, data scientists looked for a way to algorithmically extract the meaning of words and documents in order to aid in retrieving information from this wealth of data. In the late 80s a team of data scientists developed a combination of mathematical techniques that can identify similarity between words and documents. They called this technique Latent Semantic Analysis. It works by creating a term document matrix where each row corresponds to a term and each column to a document, performing a Singular Value Decomposition (SVD) on this matrix, and truncating the SVD factors by a given amount. This results in a simplified version of the original matrix which has essentially had less relevant, “noisy” data filtered out, leaving a stronger signal of word to word, document to document, and word to document similarity and dissimilarity encoded within the SVD factors and within the elements of the new matrix.

II. Background

Several key pieces of math make this possible. First the term document matrix must be constructed. There are many different weighting methods that can be used to assign each element of this matrix. Binary weighting is often used when simply wanting to determine the presence or absence of a term within a document. Any element with a 1 would correspond to that document having that term. A simple count of the number of occurrences in that document could also be used, but this can lead to issues with common terms that don’t contribute much to the meaning of the text dominating over others that, while less abundant, convey more meaning. Which weighting method you choose has an enormous impact on the quality of output you will get at the end of the LSA process.

Logarithms can be helpful in just such a situation as described above. Logarithms graph such that as their domain increases their range plateaus.

See figure 1 below.

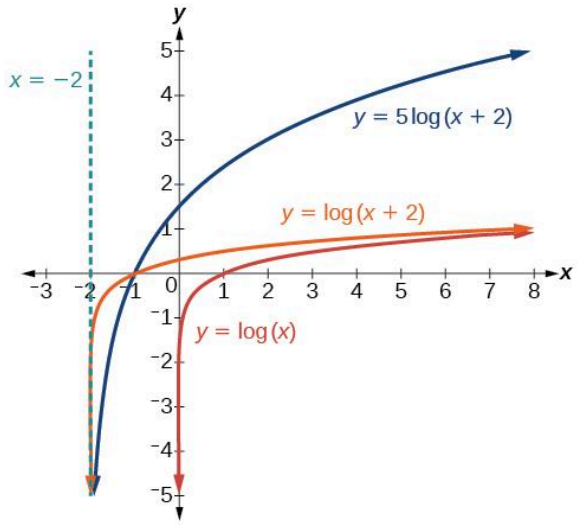

Figure 1

In this graph the value that is plugged into the logarithmic function does not increase the output linearly. This can be helpful in reducing the dominance of words which appear in high frequency.

We can assign weights that not only help distinguish a word within the context of a single document but also help to distinguish a word within a corpus of documents. The Term Frequency-Inverse Document Frequency (TF-IDF) is one such method. Term Frequency (TF) is the measure of the word's importance within a single document, and Inverse Document Frequency (IDF) is a measure of the word's rarity within a corpus of documents, that is, how important is that word in distinguishing one document from the others. By taking the product of these two measures as our weight we are left with a single weight which captures both considerations.

When combining this technique with logarithms a highly informative term-document matrix can be formed which then increases the quality of the information pulled from the subsequent LSA processing steps. A logarithmic TF term is determined by the following formula with t being the term and d being the document:

$$TF(t,d) = log(1 + \sum_{t \in d} f(t,d))$$

In short, the TF weight is the log of the count of the occurrences of the term within a given document. 1 is added to avoid 0 which is out of log's domain.

Inverse document frequency is determined by the following formula with t being the term, and D being the corpus of documents d with N number of total documents in the corpus:

$$IDF(t,D) = log(\frac{N}{|d:d \in D \text{ and } t \in d|})$$

In short, the IDF weight is the log of the total number of documents divided by the number of documents in which the term occurs one or more times. This form assumes that the term appears at least once in the document as no occurrence would result in a divide by 0. Sometimes a 1 is added to the denominator to mitigate this, but this was an assumption we were able to make in our case. Note that as the count of the documents in which the term appears approaches the total number of documents the value being passed into the logarithmic function will approach 1 from the positive direction. Therefore, the final value of the IDF weight will approach 0, the 0 value indicating that the term appears in all documents and therefore provides no distinguishing information for that document.

By taking the product of these two weights in the following formula:

$$TF - IDF(TF, IDF) = TF \times IDF$$

This combines both metrics together. If a term has no occurrence whatsoever in a given document then its TF weight will be 0 which cancels out the IDF. If the term is in all documents then its IDF weight will be 0 which will cancel out the TF. The resulting TF-IDF weighted term-document matrix provides a solid foundation for the following steps of the Latent Semantic Analysis.

With this weighted term-document matrix completed we can then do our Singular Value Decomposition (SVD). This step is usually completed with a computer due to the size of the typical term document matrix, but the math will be described. An SVD factors a matrix into 3 matrices which are equivalent to 3 transformations. The formula for an SVD is as follows:

$$M = U\Sigma V^T$$

M is the original matrix.
U is the matrix of the left singular vectors of M.
V^T is the transposed matrix of the right singular vectors of M.
Σ is the diagonal matrix of the singular values of M.
These matrices represent a set of 3 sequential transformations which, when done in order, perform the same transformation as M.

U and V are the orthogonal eigenvectors of 2 symmetric matrices that are created from the following matrix multiplications S left and right:

$$S_L = AA^T$$
$$S_R = A^TA$$

When a matrix is multiplied by its transposed form then the resulting matrix is automatically symmetric. One of the useful properties of a symmetric matrix is that its eigenvectors are orthogonal meaning they can be thought of as being perpendicular to each other though this becomes difficult to visualize beyond 3 dimensions. The eigenvectors and their corresponding eigenvalues of these to S left and right matrices are all that is needed to make up the 3 matrices that form the SVD factors.

Once we have the eigenvectors of S left and right then we can intersect the sets of S left and S right's eigenvalues such that only the eigenvalues that are in both sets remain. These are then sorted from largest to smallest. The square root of these values are placed in this sorted order into a diagonal matrix that has the same dimensions as the original matrix M. This forms the singular value matrix $\boldsymbol{\Sigma}$. U contains the eigenvectors or S left whose eigenvalues are in the intersected set, and they are placed in the same order as their corresponding eigenvalues in $\boldsymbol{\Sigma}$. V contains the eigenvectors of S right whose eigenvalues are in the intersected set, and they are placed in the same order as their corresponding eigenvalues in $\boldsymbol{\Sigma}$. V is then transposed. This completes the SVD.

What, however, do these matrices do exactly? We can think of V^T as performing a rotation which takes the right standard vectors of M and places them collinear with the standard basis vectors. $\boldsymbol{\Sigma}$ then reduces the dimensionality so that it matches the original matrix M, and scales that reduced dimensional space such that with only one more rotation our final transformation will match M. U, that final rotation, transforms the space such that the standard basis vectors will be collinear with the left singular vectors.

In Latent Semantic Analysis (LSA) the singular vector $\boldsymbol{\Sigma}$ can be thought of as representing the dimension and scaling of the latent concepts within the original data. The left singular matrix U represents all the terms and how they relate to the latent concepts. The right singular matrix V^T represents the documents and how they relate to the latent concepts. With the data now formatted in this way we can do some operations on these matrices and gain insight.

The last operation necessary to perform the LSA is to truncate the matrices. We first look at $\boldsymbol{\Sigma}$. Each diagonal value of $\boldsymbol{\Sigma}$ communicates the importance of each latent concept. We only want to keep the important ones. We can choose a value, k, latent concepts to keep. We take a $k \times k$ submatrix of $\boldsymbol{\Sigma}$ to truncate it. We take the first k columns of U to truncate it. We take the first k rows of V^T to truncate it. This essentially performs noise filtering on the dataset by cutting out the least influential latent concepts, and by multiplying these truncated factors we get a lower dimensional approximation of the original matrix M.

This essentially completes the LSA, but additional steps can be performed on the SVD factors to make associations between the tokens more clear. The U (row per term) and V^T (column per document) factors can have term/document rows/columns sorted using a clustering algorithm. These algorithms aim to group similar vectors in the latent space together. There are different measures that can be used to determine the similarity between two vectors, but one of the most common is cosine similarity. The following formula is used:

$$cos(\theta) = \frac{u \bullet v}{||u|| \times ||v||}$$

By dividing the dot product by the product of the magnitudes the influence of the magnitude is cancelled out and only the angular difference remains. The result ranges between 3 possible values which have different meanings:

1	Max Similarity
0	No Similarity
-1	Max Dissimilarity

These values can be used to determine similarity, but this must be combined with some other technique to create clusters. K-Means Clustering is an oft used technique which randomly creates a pre-specified number of clusters, k. These are called cluster centroids and are what the vectors being clustered will be compared to with cosine similarity. Vectors are clustered with the closest randomly generated centroid. The centroids are then updated to be the mean of the dataset vectors paired with them. This cycle is repeated until the clusters converge to a stable state.

III. Methodology

Due to the need for large datasets for LSA to begin to produce meaningful results, and due to the infeasibility in analyzing them and doing the calculations by hand we opted to use Python to both analyze the document data and to perform all computations. Using ChatGPT to get a healthy start on the code, and some documentation of the needed libraries we created a script which fetches a given number of random Wikipedia articles (100 is the highest we tried), preprocesses the data, constructs the weighted term-document matrix, performs a SVD on the term-document matrix, truncates the resultant SVD factors, and clusters the factors.

In order to use the built in random article URL for Wikipedia we had to use the requests Python package which can be used to call RESTful web APIs. This allowed Wikipedia to send us a fresh set of articles for every run of the script, but still get the data in a structured format that Python could easily manipulate.

Once the articles had all been fetched then the text content had to be processed to make the word occurrence counting more convenient. All non-alphabetical characters were replaced with spaces. The text was then split into a data structure that could be easily iterated over. All words were put into a dictionary where the key was the word and the value was the count of that word in that document. All these document specific term counts were placed in another dictionary where the key was the article title and the value was that document's term count dictionary.

With these counts it becomes trivial to determine the number of documents each term occurs in. With the total number of occurrences of each term per document, and the total number of documents that each term occurs in then TF-IDF weighting can be performed. We used logarithmic weighting for both TF and IDF and assigned them to a special 2D matrix data structure from the numpy Python library along with dictionaries which contain the words with their row position and the documents with their column position in the 2D matrix.

The now completed term document matrix is then passed into the numpy library's SVD method which returns the U, $\boldsymbol{\Sigma}$, V^T factors. We chose to use an energy retention method to determine how many k singular values we wanted to keep after the truncation. The value k is determined based on the number of the top singular values whose sum is equal to 95% or more of the sum total of all the singular values. This can be described as keeping 95% of the energy of the latent concepts.

With the truncated SVD factors we then performed k-means clustering to group like words and documents together.

IV. Results

Unfortunately sufficient time was not available to complete a visually appealing representation of the results of this data. Crude command line output does, however, give an idea of the results and confirm that the script is working correctly. See the following data, and the conclusion thereafter.

Term-Document Matrix (Partial Representation):

```
[[0.39184748 0.85526916 0.49445646 ... 0.         0.63907571 0.69405739]
 [0.         0.         0.         ... 0.         0.         1.59603037]
 [0.         0.         0.         ... 0.         1.59603037 0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```

Terms (Rows, Partial Representation):

```
['a', 'abbey', 'abkhazia', 'academy', 'accompaniments', 'acoustic', 'actions', 'added',
'addington', 'administered', 'administrative', 'adopted', 'advantage', 'aerospace', 'affairs',
'africa', 'after', 'against', 'aggression', 'air', 'albania', 'album', 'alcatra', 'alentejo',
'already', 'also', 'although', 'amando', 'amazed', 'america', 'american', 'americas', 'an',
'and', 'anett', 'ang', 'angola', 'angolan', 'animal', 'announced', 'any', 'appears',
'appointed', 'april', 'aqajan', 'arena', 'argentina', 'armed', 'army', 'as', 'assembly',
'assistant', 'association', 'at', 'athlete', 'athletes', 'athletics', 'august', 'australia',
'australian', 'austrian', 'auxiliary', 'available', 'awarded', 'b', 'back', 'band',
'barbecue', 'bavarian', 'be', 'became', 'become', 'beef', 'been', 'before', 'began',
'beijing', 'berlin', 'best', 'between'...
```

10 Documents (Columns):

['James Reid (Ontario politician)', 'Picanha', 'Angola at the World Athletics Championships', "2022 St. Petersburg Ladies' Trophy – Singles", 'Oybont', 'Amando Samo', 'Rumeshkan Rural District', 'Lists of lagoons', 'Igor Konashenkov', 'Soily']

U (Left Singular Vectors):

```
[[ 3.02258022e-02  3.45331400e-02 -3.06846739e-02 ...  1.95202447e-02
   4.88334059e-02 -1.52287107e-03]
 [ 2.19655872e-03  7.90290229e-03 -8.02367381e-02 ... -4.04227612e-04
  -8.28803110e-04  1.89157238e-04]
 [ 5.82694177e-02 -1.00558394e-02  1.69155644e-03 ... -1.77963112e-03
  -2.11019149e-03 -5.79862841e-04]
 ...
 [ 7.32838317e-04  2.98834737e-04 -4.68442952e-04 ... -1.01634489e-05
   2.07115327e-01  1.83649922e-03]
 [ 9.04061590e-04  2.00360847e-03 -1.57502202e-03 ... -3.29878646e-04
  -2.10882399e-02 -2.18856760e-03]
 [ 9.04061590e-04  2.00360847e-03 -1.57502202e-03 ... -3.29878646e-04
  -2.10882399e-02 -2.18856760e-03]]
```

Sigma (Singular Values):

```
[27.03488086 24.50361453 19.66425191 16.39170669 14.09414069 13.55426211
 11.84472883  9.77270567  7.66543999  6.56151825]
```

V^T (Right Singular Vectors, Partial Represetation):

```
[[ 2.30721294e-02  1.48828832e-01  3.20102602e-02  1.00896383e-02
   1.24134208e-02  1.42706856e-02  1.53137421e-02  3.86088758e-03
   9.87015534e-01  3.72071263e-02]
 [ 4.72201506e-03  9.79002208e-01  3.75387522e-02  2.27670627e-02
   4.58796485e-03  4.52930034e-03  3.07611000e-02  7.78647787e-03
  -1.54385792e-01  1.21332072e-01]
 [-1.31688983e-02  1.29560197e-01 -6.68339291e-02 -2.08359154e-02
  -5.77155699e-03 -3.73421588e-03 -1.94054139e-02  1.21262595e-03
   2.08412025e-02 -9.88574820e-01]
 [-1.54359978e-02  3.53774006e-02 -9.94309133e-01 -4.75006518e-02
  -7.18384087e-03 -1.08088340e-02 -3.44634782e-02 -7.34846320e-03
   2.57657799e-02  7.43583223e-02]
 [ 1.15943883e-02 -1.95673537e-02 -5.02138876e-02  9.98203889e-01
   8.04828776e-05  9.66126728e-03 -3.16752329e-03 -1.24063122e-03
  -5.21187123e-03 -2.04491721e-02]
 [-1.00791704e-02  2.93024920e-02  3.83341061e-02 -1.26695073e-05
  -1.00840271e-01 -2.25535487e-03 -9.93356384e-01 -7.66508053e-03
   1.04993339e-02  2.16916877e-02]
 [-9.94765375e-01  5.79546800e-03  1.71947142e-02  1.38814468e-02
  -3.59185998e-03 -9.53204949e-02  1.20306509e-02  3.65673435e-04
   2.24185197e-02  1.31738067e-02]
 [-9.59828103e-02 -4.96962993e-03 -9.51790327e-03 -9.19816981e-03
  -6.22321458e-05  9.95189813e-01 -2.01989071e-03  7.46255763e-03…
```

Selected k: 8

Truncated U_k:

```
[[ 3.02258022e-02  3.45331400e-02 -3.06846739e-02 ...  1.60540788e-03
  -3.18994596e-02  1.95202447e-02]
 [ 2.19655872e-03  7.90290229e-03 -8.02367381e-02 ...  2.55422183e-03
   1.77511835e-03 -4.04227612e-04]
 [ 5.82694177e-02 -1.00558394e-02  1.69155644e-03 ...  1.23630896e-03
   3.02080686e-03 -1.77963112e-03]
 ...
 [ 7.32838317e-04  2.98834737e-04 -4.68442952e-04 ... -1.18740609e-02
  -4.83988927e-04 -1.01634489e-05]
 [ 9.04061590e-04  2.00360847e-03 -1.57502202e-03 ... -1.16968887e-01
   1.62108263e-03 -3.29878646e-04]
 [ 9.04061590e-04  2.00360847e-03 -1.57502202e-03 ... -1.16968887e-01
   1.62108263e-03 -3.29878646e-04]]
```

Truncated Sigma_k:

```
[[27.03488086  0.          0.          0.          0.          0.
   0.          0.        ]
 [ 0.         24.50361453  0.          0.          0.          0.
   0.          0.        ]
 [ 0.          0.         19.66425191  0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.         16.39170669  0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.         14.09414069  0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.         13.55426211
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
  11.84472883  0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          9.77270567]]
```

Truncated Vt_k (Partial Representation):

```
[[ 2.30721294e-02  1.48828832e-01  3.20102602e-02  1.00896383e-02
   1.24134208e-02  1.42706856e-02  1.53137421e-02  3.86088758e-03
   9.87015534e-01  3.72071263e-02]
 [ 4.72201506e-03  9.79002208e-01  3.75387522e-02  2.27670627e-02
   4.58796485e-03  4.52930034e-03  3.07611000e-02  7.78647787e-03
  -1.54385792e-01  1.21332072e-01]
 [-1.31688983e-02  1.29560197e-01 -6.68339291e-02 -2.08359154e-02
  -5.77155699e-03 -3.73421588e-03 -1.94054139e-02  1.21262595e-03
   2.08412025e-02 -9.88574820e-01]
 [-1.54359978e-02  3.53774006e-02 -9.94309133e-01 -4.75006518e-02
  -7.18384087e-03 -1.08088340e-02 -3.44634782e-02 -7.34846320e-03…
```

Clusters (Terms, Partial Representation):

already	Cluster 0
bavarian	Cluster 0
cap	Cluster 0
cover	Cluster 0
cut	Cluster 0
femoris	Cluster 0
frying	Cluster 0
grill	Cluster 0
berlin	Cluster 1
britain	Cluster 1
british	Cluster 1
denver	Cluster 1
dvd	Cluster 1
edition	Cluster 1
encore	Cluster 1
end	Cluster 1
aggression	Cluster 2
armed	Cluster 2
army	Cluster 2
career	Cluster 2
chief	Cluster 2
collective	Cluster 2
command	Cluster 2
commander	Cluster 2
catholic	Cluster 7
diocese	Cluster 7
ordained	Cluster 7
priesthood	Cluster 7
roman	Cluster 7

V. Conclusion

Latent Semantic Analysis, as demonstrated in the results, clearly has the ability to determine similarity relationships within the data set. The process, once understood, is actually quite straightforward, but produces significant insights into the data. Potential applications are too numerous to mention, but this could be used to help search for documents relevant to a search term, thesaurus construction by finding synonymous words, recommend documents similar to a particular document, and many more. The biggest issue with this particular implementation is that the k value was not high enough for the k-means clustering. Far more clusters were needed with a word set this size to refine clusters into something more distinct. Regardless, the selected data shown here does clearly demonstrate like terms being clustered. Better visualization of this data is needed to more clearly demonstrate and deal with the scale of the data at hand. Corpuses with higher word count documents and with a much larger number of documents become infeasible to visualize in any way.

VI. References

[1] Wikipedia Article on Latent Semantic Analysis

Wikipedia contributors, "Latent semantic analysis," *Wikipedia*, [Online]. Available: https://en.wikipedia.org/wiki/Latent_semantic_analysis. [Accessed: Nov. 24, 2024].

[2] Wikipedia Article on TF-IDF

Wikipedia contributors, "Tf–idf," *Wikipedia*, [Online]. Available: https://en.wikipedia.org/wiki/Tf%E2%80%93idf. [Accessed: Nov. 24, 2024].

[3] YouTube Video

StatQuest with Josh Starmer, "Latent semantic analysis (LSA) clearly explained!!!," *YouTube*, Sep. 15, 2023. [Online]. Available: https://youtu.be/vSczTbgc8Rc?si=M8Cn0gcjKRDOXyFe. [Accessed: Nov. 24, 2024].