
<https://www.overleaf.com/project/638faea9770efec762787814>

Topic Modeling of Taylor Swift Lyrics Using Singular Value Decomposition and Non-negative Matrix Factorization

Isabella Bates

Dr. Bobby Benim

December 8, 2022

0.1 Abstract

In statistics and natural language processing, the ability to access hidden topics within a set of documents, a corpus, is extremely valuable. This is known as topic modeling. Existing approaches for topic modeling mostly rely on Singular Value Decomposition (SVD). However, SVD has limitations: the vectors can have negative values and each document in the corpus must contain only one topic, otherwise we can only recover the span of the topic vectors instead of the topic vectors themselves. As a result of these limitations, there is another matrix factorization that has been found to be better for topic modeling known as Non-negative Matrix Factorization (NMF). NMF is an analog factorization to SVD, except all vectors are non-negative. In this paper, topic modeling will be achieved in about four steps: isolating and pre-processing data, filling a document term matrix using Term Frequency-Inverse Document Frequency (TF-IDF), applying Latent Semantic Analysis (LSA), and finally using either SVD or NMF to attain the hidden topics of the corpus.

Through this paper, I will use both methods of topic modeling to analyze the hidden topics in a set of documents, specifically the collection of lyrics of a popular artist with an interesting history: Taylor Swift. For this analysis, I used a data set containing all of Taylor Swift's lyrics, from her 2006 self-titled album to her 2019 album, "Lover", sourced from Kaggle. Using the linear algebra concepts, SVD and NMF, this paper aims to find and analyze the most important and frequent terms used in Taylor Swift's songwriting and find the core themes of her work. In addition, it will use cosine similarity to see the similarity between Taylor Swift songs based on their lyrics.

0.2 Attribution

I researched the topic model methods and Python packages, implemented the process and made visuals of the data in Python, analyzed the results, and prepared the project report.

0.3 Introduction

The main goal of topic modeling is to distill the content of a large set of text documents into its core topics. These topics are determined by the frequency and similarity of certain words throughout the documents. "Exploring popular topic models" by Poonam Tijare and Jhansi Rani P outlines a procedure for finding the hidden topics in a corpus using LSA with SVD and NMF. To summarize, they first input a document term matrix and find the TF-IDF representation of the document term matrix. Then to find the topic model with SVD or NMF, they apply the respective factorization to the document term matrix. They implemented this process with a collection of tweets from Twitter using Twitter API. They found ten topics and the top 15 words in each topic using three methods: Latent Dirichlet Allocation (LDA), LSA with SVD, and NMF [6].

In this paper, we will focus on two of those methods: LSA with SVD and NMF. By following the methodology discussed in "Exploring popular topic models", we will investigate the corpus of Taylor Swift lyrics spanning her self-titled album through her "Lover" album. This will allow us to see the hidden, core topics of songwriting in a more efficient and quicker way than listening through every song in her discography. This paper will be an example of how topic models created with SVD and NMF can reveal the key groups of similar and frequent topics in a large set of data. I utilized Python and scikitlearn package implementations for the decomposition and analysis of the document matrix with Taylor Swift lyrics. In addition, I used Pandas, NumPy, and Matplotlib for data visualization. These can be seen in Appendix A.

In order to understand the underlying process of topic modeling, one needs to understand term-frequency-inverse document frequency (TF-IDF), Latent Semantic Analysis (LSA), Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), and cosine similarity. Since the only topic covered in APPM 3310 of these were SVD and cosine similarity, this paper will discuss how all of these are implemented for the use of topic modeling in the Mathematical Formulation section.

0.4 Mathematical Formulation

Topic Modeling takes a large set of documents and extracts the important topics across all documents in the set. These topics are determined by the frequency and similarity of terms in the documents. To do this, methods of topic modeling make use of matrix factorization to decompose a document term matrix into multiple matrices that separate information about the document term matrix. These factorizations and their given matrices will be

described in greater detail in the Singular Value Decomposition and Non-negative Matrix Factorization sections.

0.4.1 Term Frequency-Inverse Document Frequency (TF-TDF)

To build a topic model, we need to extract features from the corpus to model. Let the document term matrix D be an $m \times n$ matrix for m documents with n words in the corpus. One way that data is extracted for topic modeling is using a Bag-of-Words model (BoW). This model turns text into vectors by counting the frequency of each word in a corpus. For this project, we will use term frequency-inverse document frequency (TF-TDF) which is similar to BoW but goes deeper. There are two parts to TF-IDF: the *term frequency* which is how often a term appears in a document and the *inverse document frequency* which is how common or rare a term is in a document. This method uses the frequency of words to determine how important those words are to a given document and gives them a score based on this. The TF-IDF score is found using:

$$w_{ij} = TF_{ij} \cdot \log \left(\frac{M}{df_j} \right) \quad (1)$$

TF_{ij} is the number of times T_j occurs and df_j is the number of documents containing the term T_j . M is the documents.

0.4.2 Singular Value Decomposition (SVD)

Let the document term matrix D be an $m \times n$ matrix for m documents with n words in the corpus. Singular Value Decomposition (SVD) factors a document term matrix into three matrices.

$$D = U_t S_t V_t^T \quad (2)$$

U is an $m \times m$ left singular vector matrix that is obtained by the eigen decomposition of the Gram matrix DD^T . The columns of U are orthonormal eigenvectors of DD^T . S is an $m \times n$ diagonal matrix with the singular values, $\sqrt{\lambda_i}$, of DD^T in descending order. These represent the importance of each topic. The higher the singular value, the higher of importance the topic. V^T is an $n \times n$ right singular vector matrix. This matrix holds the topics of the text along its rows and is also called the *term topic* matrix. Both U and V are orthogonal matrices.

0.4.3 Non-negative Matrix Factorization (NMF)

Non-negative Matrix Factorization (NMF) factors a document term matrix into a document-topic matrix and topic-term matrix. An important limitation is that the input document matrix must be completely non-negative. Let the document term matrix D be an $m \times n$ matrix for m documents with n words in the corpus where D_{ij} is greater than 0. NMF decomposes D into two matrices such that:

$$D \approx W \cdot H \quad (3)$$

Every element of matrices W_{ij} and H_{ij} are greater than or equal to 0 and $z < \min(x, y)$ [6]. W is an $m \times t$ document topic matrix for m documents with t topics. W is the basis matrix that holds the topics, or clusters, discovered from the documents. H is an $t \times n$ document topic matrix for t topics and n words in corpus. H is the coefficient matrix that holds the corresponding weight, or importance, of the topics in each document.

W and H are an approximation of D . To optimize our approximation, we can use an algorithm present in the scikitlearn package known as the "multiplicative update rules". This algorithm calculates W and H by measuring the error between document term matrix D and the product of its factors W and H , on the basis of Euclidean distance:

$$\|D - WH\| \quad (4)$$

The Euclidean distance is non-increasing under the update rules. Using this, we can derive two equations to update matrices W and H . To update W matrix:

$$W_{ic} \leftarrow W_{ic} \frac{(W^T D)_{ic}}{(W^T W D)_{ic}} \quad (5)$$

To update H matrix:

$$H_{cj} \leftarrow H_{cj} \frac{(H^T D)_{cj}}{(H^T H D)_{cj}} \quad (6)$$

Using equations 4, 5, and 6, we simultaneously update the values of W and H . The matrices W and H are initialized with values. Then, the algorithm is run iteratively until we find a W and H that minimize the cost function, equation 4. With the new matrices we computed for W and H , we compute the error using equation 4 and repeat this process until we converge. Thus, the multiplicative update rules will modify the initial values of W and H until the approximation error converges or until the product of them approaches D .

0.4.4 Cosine Similarity

Cosine similarity calculates the cosine of the angle between two vectors of an inner product space. The cosine similarity between two non-zero vectors is given by:

$$similarity = \cos(\theta) = \frac{A \cdot B}{||A|| ||B||} \quad (7)$$

It can be used as a method for finding the most relevant information in a corpus. Python has a built-in function for cosine similarity that takes a matrix, or set of vectors, as input and computes the similarity between them.

0.5 Examples and Numerical Result

0.5.1 Dataset

The goal is to find the main hidden topics across Taylor Swift's discography and then find the similarity between her songs based on their lyrics. As described previously, the data set I will be using in this paper is sourced from *Kaggle* [1]. It is a data set containing all the lyrics across Taylor Swift's albums, starting with her first album "Taylor Swift" to her album "Lover". The original data frame before preprocessing was saved in the variable *tay* and can be seen below and in Appendix A.

0.5.2 Data Preprocessing

Before any matrix factorizations can be applied, the data must be preprocessed. This means removing punctuation, capitalization, stop words, stemming and lemmatization. Stop words are the words which are filtered out before processing of natural language data because they do not provide any meaningful information. For example, "the" is a common word in songs, but it does not help with detecting underlying themes of the song. Some of the words I removed are "oh", "ahh", "don", "cause", etc. This helps with achieving more accurate results. The code for this is in Appendix B.

	track_album	track_artist	track_title	danceability	energy	key	loudness	mode	speechiness	acousticness	...	valence	tempo	duration_ms	time_signature	track_id
0	Taylor Swift	Taylor Swift	Tim McGraw	0.580	0.491	0	-6.462	1	0.0251	0.57500	...	0.425	76.009	232107	4	spotify:track:0Om9WAB5RS09L80dyOfTI
1	Taylor Swift	Taylor Swift	Picture To Burn	0.658	0.877	7	-2.098	1	0.0323	0.17300	...	0.821	105.586	173067	4	spotify:track:32mVHdy0b1XKgr0ajsB
2	Taylor Swift	Taylor Swift	Teardrops On My Guitar	0.621	0.417	10	-6.941	1	0.0231	0.28800	...	0.289	99.953	203040	4	spotify:track:7zMcNqs55Mxer82bvZFk
3	Taylor Swift	Taylor Swift	A Place In This World	0.576	0.777	9	-2.881	1	0.0324	0.05100	...	0.428	115.028	199200	4	spotify:track:73OX8GdpOeGzKC60vGSb
4	Taylor Swift	Taylor Swift	Cold as You	0.418	0.482	5	-5.769	1	0.0266	0.21700	...	0.261	175.558	239013	4	spotify:track:7an1exwMnfYRcdVQm0yD
...
106	Lover	Taylor Swift	Miss Americana & the Heartbreak Prince	0.662	0.747	11	-6.926	0	0.0736	0.02800	...	0.487	150.088	234147	4	spotify:track:214nt20w5wOxJnY462kl
107	Lover	Taylor Swift	Paper Rings	0.811	0.719	9	-6.553	1	0.0497	0.01290	...	0.865	103.979	222400	4	spotify:track:4y5bvROuBDPr5fuwXblB
108	Lover	Taylor Swift	Soon You'll Get Better (feat. Dixie Chicks)	0.433	0.182	0	-12.566	1	0.0641	0.90700	...	0.421	207.476	201587	4	spotify:track:4AYtqFyFbX0Xkc2wtcyg
109	Lover	Taylor Swift	The Archer	0.292	0.574	0	-9.375	1	0.0401	0.12000	...	0.166	124.344	211240	4	spotify:track:3pHkh7d0LzM2AldUtzx

Figure 1. Data frame of various information about Taylor Swift’s discography including song titles, lyrics, albums, and more. Data sourced from *Kaggle*.

0.5.3 Finding the TF-IDF Scores

After reading in the file for Taylor Swift lyrics and preprocessing the data, I use TF-IDF to populate the document term matrix D . I use the TF-IDF vectorizer from the scikitlearn package in Python. This can be seen in Appendix C and D. After this, each row in D represents documents and each column represents words in the document with TF-IDF scores. The updated document term matrix can be seen below. Higher TF-IDF scores represent higher importance and frequency in the data set.

	ain	baby	bad	beautiful	best	better	blue	break	car	change	...	trying	turn	waiting	walk	wanted
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.264438	0.000000	0.000000	0.0	...	0.000000	0.055182	0.0	0.000000	0.0000
1	0.000000	0.123388	0.403619	0.000000	0.080724	0.077898	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.0000
2	0.000000	0.000000	0.000000	0.339606	0.000000	0.141475	0.000000	0.152281	0.497438	0.0	...	0.000000	0.162098	0.0	0.000000	0.0000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.095507	0.000000	0.000000	0.0	...	0.498249	0.000000	0.0	0.000000	0.0000
4	0.000000	0.000000	0.000000	0.000000	0.143627	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.287255	0.1522
...
106	0.000000	0.000000	0.645764	0.000000	0.000000	0.000000	0.293275	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.0000
107	0.000000	0.096601	0.000000	0.000000	0.000000	0.000000	0.133942	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.0000
108	0.000000	0.000000	0.200046	0.000000	0.200046	0.579130	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.0000
109	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.0000
110	0.171212	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.0	0.000000	0.0000

Figure 2. Document term matrix showing the various importance and frequency of words in Taylor Swift’s discography.

0.5.4 Topic Model with SVD

To decompose the document term matrix of Taylor Swift lyrics, I use `linalg.svd()` which is a built-in function that returns a left singular vector matrix, a singular values matrix, and right singular vector matrix. I save these respectively in created variables U , S , and Vt . The code for this can be referenced in Appendix D.

I then plot a scree plot of the singular values of S to visualize how many eigenvectors we will use for our analysis.

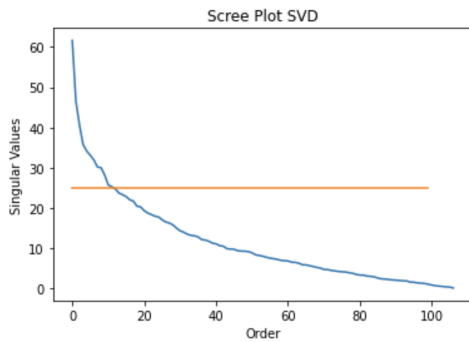


Figure 4. Scree plot of all singular values.

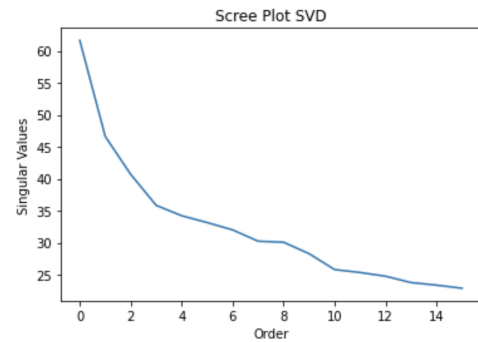


Figure 5. Scree plot of top 15 singular values.

Figure 4 shows all singular values in S . These singular vectors rank importance of the topics found in Taylor Swift's lyrics. We can immediately see from this plot that the first singular vector is the most important, with a singular value of 62.2. As our x values approach 40, the topics have a similar level of importance that is significantly lower. To choose how many components to include in this analysis, I chose an area of the graph where the singular values momentarily flattened. This can be seen in Figure 5 where the singular values have been reduced to the first 15. See Appendix E for the code for this.

The diagonal singular value matrix S helps in visualizing what components are important to focus on. From the data, I decided to focus on the first 15 components based on their higher ranking of importance. I created a function called `print_topics()` which takes in a vector v of topics and returns a vector with the most popular words for each topic. This function implementation is in Appendix G. I chose to focus on the top five topics. I call `print_topics(Vt[: 9])` which takes the top five topics from the term topic matrix Vt as input and returns the top 15 words for each topic. It returns 15 words for each topic because that is the number of important components I determined from the scree plots of S previously. The results are below.

Topic 0: "time", "stay", "love", "baby", "think", "say", "look", "said", "bad", "right", "come", "let", "good", "night", "tell"

Topic 1: "stay", "time", "think", "best", "hand", "let", "people", "wanted", "hold", "took", "times", "leave", "turn", "try", "fight"

Topic 2: "love", "baby", "new", "good", "say", "waiting", "right", "come", "break", "knew", "let", "night", "end", "gone", "life"

Topic 3: "say", "good", "stay", "feel", "light", "bad", "love", "girl", "look", "end", "forever", "play", "home", "hand", "world"

Topic 4: "think", "good", "say", "said", "car", "feel", "girl", "bad", "light", "better", "break", "things", "dress", "eyes", "place"

According to this SVD topic model, these topics and words represent some of the most frequent themes in Taylor Swift's songwriting. The themes I created for each of these five topics respectively are: Spending nights with someone, Connection, Ambivalent love, Long-lasting love, and Nights out.

0.5.5 Finding the Number of Components Using Latent Semantic Analysis (LSA)

Latent Semantic Analysis is parallel to SVD. It decomposes a matrix based on its eigenvalues just like SVD. I need to know the number of components in order to perform NMF, so I use built-in Python functions for LSA to plot a line of the eigenvalues of the components. This will help decide how many components to include in the analysis similar to the Scree plots used for SVD. I use *TruncatedSVD* which helps with gathering a small amount of topics that best convey the content of the text. The code for this is in Appendix F.

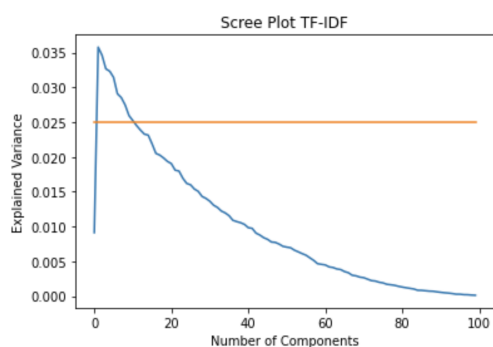


Figure 6. Scree plot of all components.

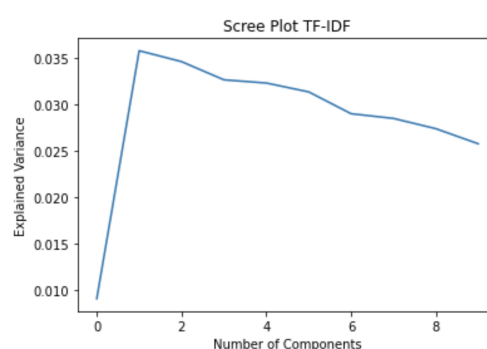


Figure 7. Scree plot for top 9 components.

Figure 6 is a plot of a large amount of the eigenvalues. This is to see areas where the line is less steep. Figure 7 is a plot of the number of topics I decided to analyze, which was nine.

0.5.6 Topic Model with NMF

To topic model the Taylor Swift document term matrix, I need to find two parts of the NMF equation: W and H . The initial values of W and H are the TF-IDF scores of the Taylor Swift lyrics data set. I compute the NMF decomposition of the document term matrix by using a function `nmf_function()` which takes the number of components, the document term matrix, and TF-IDF vectorizer as inputs. It calls the built-in function `NMF()` which takes the number of components as input. The `nmf_function()` prints the top topics with their corresponding top terms found from the H matrix. The code for this can be seen in Appendix H. I used this function to access the top nine topics and their corresponding top 15 terms, which can be seen below.

Topic 0: "know", "better", "best", "run", "won", "going", "day", "looking", "try", "trying", "say", "drive", "feel", "eyes", "didn"

Topic 1: "love", "beautiful", "knew", "lights", "life", "saw", "right", "waiting", "home", "hands", "high", "heart", "eyes", "let", "little"

Topic 2: "said", "think", "girl", "home", "gone", "did", "forever", "look", "wrong", "dress", "say", "thought", "didn", "feel", "phone"

Topic 3: "baby", "bad", "smile", "did", "really", "know", "like", "high", "won", "coming", "things", "life", "eyes", "let", "new"

Topic 4: "stay", "little", "think", "right", "beautiful", "hand", "hold", "won", "home", "getting", "let", "like", "time", "light", "make"

Topic 5: "like", "yeah", "know", "end", "trying", "head", "night", "new", "street", "tell", "best", "hands", "feeling", "need", "make"

Topic 6: "time", "tell", "sorry", "long", "night", "let", "dreams", "hold", "remember", "hard", "didn", "town", "life", "turn", "thing"

Topic 7: "come", "day", "like", "feeling", "miss", "right", "kiss", "feel", "need", "new", "forget", "rain", "look", "things", "fight"

Topic 8: "good", "remember", "looking", "bad", "coming", "break", "light", "right", "play", "say", "make", "heart", "room", "did", "getting"

According to this NMF topic model, these clusters of words represent some of the most prevalent themes in Taylor Swift's songs. The themes I created for each of these nine topics respectively are: Permanency/belonging, New things, Light/love, Trying/moving on, Staying/not moving on, Missing something/sad, Regret/remorse, New relationship, and Contemplation/remembering.

0.5.7 Finding Similar Songs with Cosine Similarity

I found the cosine similarity of the vectors of the H matrix from the NMF decomposition I did previously.

	0	1	2	3	4	5	6	7	8
Tim McGraw	0.000	0.000	0.000	0.346	0.057	0.025	0.049	0.000	0.000
Picture To Burn	0.006	0.207	0.057	0.000	0.000	0.182	0.023	0.016	0.026
Teardrops On My Guitar	0.061	0.045	0.221	0.013	0.073	0.124	0.000	0.000	0.033
A Place In This World	0.155	0.000	0.000	0.000	0.000	0.000	0.092	0.002	0.000
Cold as You	0.000	0.020	0.000	0.061	0.007	0.079	0.004	0.181	0.045
...
Miss Americana & the Heartbreak Prince	0.082	0.141	0.008	0.034	0.000	0.000	0.037	0.061	0.143
Paper Rings	0.015	0.097	0.051	0.004	0.012	0.104	0.043	0.095	0.000
Soon You'll Get Better (feat. Dixie Chicks)	0.000	0.038	0.000	0.000	0.024	0.463	0.000	0.000	0.000
The Archer	0.000	0.000	0.000	0.000	0.314	0.000	0.068	0.111	0.065
You Need to Calm Down	0.000	0.000	0.000	0.000	0.002	0.138	0.025	0.075	0.016

Figure 8. Data frame representation of matrix H.

The H matrix holds the song titles and the weight, or importance, based on the lyrics within them. I made a new data frame to display the cosine similarity that had been found among the vectors of H.

	0	1	2	3	4	5	6	7	8	9	...	101	102	103
0	1.000000	0.056294	0.119737	0.070456	0.313695	0.072861	0.008973	0.107140	0.920081	0.832942	...	0.179817	0.196053	0.071475
1	0.056294	1.000000	0.581425	0.060102	0.375359	0.667763	0.798958	0.149444	0.240009	0.137443	...	0.783860	0.071670	0.216307
2	0.119737	0.581425	1.000000	0.189581	0.229217	0.464801	0.277363	0.771028	0.105503	0.423399	...	0.386574	0.190126	0.686642
3	0.070456	0.060102	0.189581	1.000000	0.019046	0.011943	0.012711	0.000000	0.030192	0.102515	...	0.197655	0.630658	0.264255
4	0.313695	0.375359	0.229217	0.019046	1.000000	0.401170	0.145459	0.021964	0.301670	0.461343	...	0.220712	0.032400	0.016369
5	0.072861	0.667763	0.464801	0.011943	0.401170	1.000000	0.114500	0.000000	0.008645	0.001192	...	0.268199	0.019214	0.016696
6	0.008973	0.798958	0.277363	0.012711	0.145459	0.114500	1.000000	0.068049	0.325141	0.087346	...	0.855979	0.020274	0.092975
7	0.107140	0.149444	0.771028	0.000000	0.021964	0.000000	0.068049	1.000000	0.000000	0.326682	...	0.282518	0.343595	0.636845
8	0.920081	0.240009	0.105503	0.030192	0.301670	0.008645	0.325141	0.000000	1.000000	0.825484	...	0.340638	0.008639	0.000000
9	0.832942	0.137443	0.423399	0.102515	0.461343	0.001192	0.087346	0.326682	0.825484	1.000000	...	0.106401	0.028544	0.374951
10	0.795718	0.145915	0.185300	0.224683	0.601077	0.059607	0.141124	0.024322	0.815719	0.826951	...	0.207539	0.063269	0.040062
11	0.131294	0.578580	0.425349	0.450017	0.319016	0.833989	0.078276	0.009996	0.007759	0.025637	...	0.332000	0.480377	0.263751
12	0.176252	0.384652	0.854608	0.000274	0.132817	0.033686	0.344083	0.815158	0.222085	0.546444	...	0.319360	0.101795	0.790041
13	0.000000	0.141257	0.326025	0.000000	0.204115	0.141052	0.088427	0.196757	0.000000	0.115843	...	0.088369	0.000000	0.256040
14	0.226347	0.363149	0.225984	0.464369	0.175497	0.370925	0.137801	0.131525	0.071989	0.042878	...	0.441762	0.848623	0.471086

Figure 9. Data frame representation of the cosine similarity of the vectors of matrix H.

Each row of this data frame holds the cosine similarity of a set of songs that were found to be similar to each other based on their lyric terms. By viewing the top largest values in each row, I was able

to find clusters of songs that were found to be similar. The code for this can be found in Appendix J. In row 0, the Taylor Swift songs "Delicate", "New Year's Day", "Starlight", and "Fearless" were found to share similar words for their lyrics. They had cosine similarity values of 0.99999, 0.99998, 0.99987, and 0.99971 respectively. These songs share themes of partying and vulnerable, fun love. In row 13, "State of Grace", "Sad Beautiful Tragic", and "This Love" were found to be similar. These songs share themes of ambivalent love that changes between good and bad quickly. They had cosine similarity values of 1.0, 1.0, and 0.998 respectively.

0.6 Discussion

Through this project, I explored using two different matrix factorizations, SVD and NMF, for topic modeling Taylor Swift's lyrics. In the beginning, I wanted to find the most significant terms and topics discussed across Taylor Swift's discography. With topic modeling, I was able to find clusters of frequent words in her music that had similarity. By analyzing these words, I identified a theme for each cluster, thus the main topics of Taylor Swift's discography. Furthermore, I was able to find clusters of her songs that shared similar lyrics. Besides the factorization methods, the process for topic modeling was very similar between both. For both methods, I started with a raw data set of all lyrics from Taylor Swift's discography, beginning with her "Taylor Swift" album and ending with her "Lover" album. I preprocessed the data set before doing anything else. With the use of TF-IDF vectorizer in Python, I found the importance and frequency of each lyric and saved that in a matrix. Then, I found the top topics and words used in Taylor Swift's songwriting using both SVD and NMF, but received varying results.

The top words from SVD were different than NMF. This is partly due to SVD not ensuring that all values will be non-negative which could change the order of words. SVD also splits the document term matrix into three matrices while NMF only splits it into two matrices. This would also cause difference in values. Based on the results of the two topic models, NMF gave more accurate and workable data. The top words within the topics found by SVD seemed repetitive, thus making the top topics less distinct from one another. On the other hand, the topics given by NMF were more distinct from each other. Each topic from NMF seemed to have an immediate, more obvious theme while the topics from SVD were more unclear. The NMF matrices also proved to be valuable for finding similarities between Taylor Swift songs with cosine similarity.

Nonetheless the goal of this project was to function as an application and analysis of two methods of topic modeling. Although finding the similarity between a popular artist's lyrics is a trivial example, it demonstrates one of the many applications of topic modeling using SVD and NMF. This is applicable in so many other areas such as bio-informatics, opinion summarization, classification, spam filters, and more. In conclusion, the methods and results of the paper demonstrate how hidden, important topics of large sets of documents can be extracted more efficiently using matrix methods including eigenvalues, SVD, NMF, and cosine similarity.

References

- References [1] Aditya, Ishaan. "Taylorswiftlyricsfeatures." Kaggle, 15 July 2020, <https://www.kaggle.com/datasets/ishaanaditya/taylorswiftlyricsfeatures?select=LyricsFeaturesGenre.csv>.
- [2] Arora, Sanjeev, et al. "Learning Topic Models – Going beyond SVD." 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, 2012, <https://doi.org/10.1109/focs.2012.49>.
- [3] "Learn." Scikit, <https://scikit-learn.org/stable/>.
- [4] R. Vangara et al., "Finding the Number of Latent Topics With Semantic Non-Negative Matrix Factorization," in IEEE Access, vol. 9, pp. 117217-117231, 2021, doi: 10.1109/ACCESS.2021.3106879.
- [5] Stevens, Keith et al. "Exploring Topic Coherence over Many Models and Many Topics." Conference on Empirical Methods in Natural Language Processing (2012).
- [6] Tijare, Poonam, and P Jhansi Rani. "Exploring Popular Topic Models." Journal of Physics: Conference Series, vol. 1706, no. 1, 2020, p. 012171., <https://doi.org/10.1088/1742-6596/1706/1/012171>.

Appendix

Appendix A

Python code for reading in raw data set of information for Taylor Swift discography and extracting the song titles and lyrics for the document term matrix.

```
# analysis of Taylor Swift lyrics using NMF
import numpy as np
import pandas as pd
import seaborn as sns

import re
import string

from sklearn.feature_extraction import text
from sklearn.datasets import fetch_20newsgroups
from sklearn import decomposition
from scipy import linalg

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import TruncatedSVD, NMF, PCA
from sklearn.metrics.pairwise import cosine_similarity

from sklearn.cluster import KMeans, DBSCAN
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt
%matplotlib inline
np.set_printoptions(suppress=True)

# read data from file
tay = pd.read_csv('TaylorSwiftLyricsFeatureSet.csv')

tay['year_released'] = tay['track_album']
tay['year_released'] = tay['year_released'].replace(['Taylor Swift', 'Fearless', 'Speak Now', 'Red', '1989',
    'reputation', 'Lover'],[2006,2008,2010,2012,2014,2017,2019])
tay['world_sales_USD'] = tay['track_album'].replace(['Taylor Swift', 'Fearless', 'Speak Now', 'Red', '1989',
    'reputation', 'Lover'],[7000000,12000000,5500000,6000000,10500000,5000000,4000000])
song_titles = tay['track_title'].values
album = tay['track_album'].values
tay
```

	track_album	track_artist	track_title	danceability	energy	key	loudness	mode	speechiness	acousticness	...	valence	tempo	duration_ms	time_signature	track_id
0	Taylor Swift	Taylor Swift	Tim McGraw	0.580	0.491	0	-6.462	1	0.0251	0.57500	...	0.425	76.009	232107	4	spotify:track:00m9WAB5RS09L80DyOFTI
1	Taylor Swift	Taylor Swift	Picture To Burn	0.658	0.877	7	-2.098	1	0.0323	0.17300	...	0.821	105.586	173067	4	spotify:track:32mVHdy0bi1XKgr0aJsB
2	Taylor Swift	Taylor Swift	Teardrops On My Guitar	0.621	0.417	10	-6.941	1	0.0231	0.28800	...	0.289	99.953	203040	4	spotify:track:7zMcNqs55Mxer82bvZFkj
3	Taylor Swift	Taylor Swift	A Place In This World	0.576	0.777	9	-2.881	1	0.0324	0.05100	...	0.428	115.028	199200	4	spotify:track:73OX8GdpOeGzKC6OvGSb
4	Taylor Swift	Taylor Swift	Cold as You	0.418	0.482	5	-5.769	1	0.0266	0.21700	...	0.261	175.558	239013	4	spotify:track:7anlexwMmfYRcdVQm0yD
...
106	Lover	Taylor Swift	Miss Americana & the Heartbreak Prince	0.662	0.747	11	-6.926	0	0.0736	0.02800	...	0.487	150.088	234147	4	spotify:track:214nt20w5wOxJnY462kil
107	Lover	Taylor Swift	Paper Rings	0.811	0.719	9	-6.553	1	0.0497	0.01290	...	0.865	103.979	222400	4	spotify:track:4y5bvRQubDPf5fuwXbiB:
108	Lover	Taylor Swift	Soon You'll Get Better (feat. Dixie Chicks)	0.433	0.182	0	-12.566	1	0.0641	0.90700	...	0.421	207.476	201587	4	spotify:track:4AYtqFyFbX0Xkc2wtcyg
109	Lover	Taylor Swift	The Archer	0.292	0.574	0	-9.375	1	0.0401	0.12000	...	0.166	124.344	211240	4	spotify:track:3pHkh7d0lzM2AldUtzx

Appendix B

Python code for pre-processing data: removing capitalization, punctuation, stop words, etc.

```
# clean data before applying NMF
# remove \n
n = lambda x: re.sub('\n',' ',x)
alpha = lambda x: re.sub('\w*\d*\w*', ' ', x)
# remove punctuation and make all lyrics lowercase
lowercase = lambda x: re.sub('[%s]' % re.escape(string.punctuation), ' ', x.lower())

tay['track_lyric'] = tay.track_lyric.map(n).map(lowercase).map(alpha)

# data after cleaning
tay.track_lyric[0]

'he said the way my blue eyes shined put those georgia stars to shame that night i said that s a lie just a boy in a chevy truck that had a tendency of gettin stue
k on backroads at night and i was right there beside him all summer long and then the time we woke up to find that summer gone but when you think tim mcgraw i hope you
think my favorite song the one we danced to all night long the moon like a spotlight on the lake when you think happiness i hope you think that little black dress thin
k of my head on your chest and my old faded blue jeans when you think tim mcgraw i hope you think of me september saw a month of tears and thankin god that you weren
t here to see me like that but in a box beneath my bed is a letter that you never read from three summers back it s hard not to find it all a little bittersweet and lo
okin back on all of that it s nice to believe when you think tim mcgraw i hope you think my favorite song the one we danced to all night long the moon like a spotlig
ht on the lake when you think happiness i hope you think that little black dress think of my head on your chest and my old faded blue jeans when you think tim mcgraw i
hope you think of me and i m back for the first time since then i m standin on your street and there s a letter left on your doorstep and the first thing that you ll
read is when you think tim mcgraw i hope you think my favorite song someday you ll turn your radio on i hope it takes you back to that place when you think happines
s i hope you think that little black dress think of my head on your chest and my old faded blue jeans when you think tim mcgraw i hope you think of me oh think of me
mmmm he said the way my blue eyes shine put those georgia stars to shame that night i said that s a lie '
```

```
# remove words that we don't want to include in analysis
all_remove = ['just','don','gonna','cause','ll','ve','got','oh','eh','aah','want','way','away','ooh','wanna','ain','hey']
remove = text.ENGLISH_STOP_WORDS.union(all_remove)
```

Appendix C

Implementation of the TF-IDF vectorizer from the sklearn package. It was used to create a vectorizer *cv*, and find the TF-IDF scores for the document term matrix *df*.

```
tfidf_v = TfidfVectorizer(stop_words=remove,min_df=0.1)
tfidf_freq = tfidf_v.fit_transform(tay.track_lyric)
```

```
# TF-IDF vectorizer
cv = TfidfVectorizer(stop_words=remove,min_df=0.1,max_df=0.7)
x = cv.fit_transform(tay.track_lyric).toarray()
df = pd.DataFrame(x,columns=cv.get_feature_names())

df
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names_out is deprecated in 1.0
and will be removed in 1.2. Please use get_feature_names_out instead.
warnings.warn(msg, category=FutureWarning)
```

	baby	bad	beautiful	best	better	blue	break	car	change	cold	...	trying	turn	waiting	walk	wanted	won	world	wrong	yeah
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.264438	0.000000	0.000000	0.0	0.000000	...	0.000000	0.055182	0.0	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000
1	0.123388	0.403619	0.000000	0.080724	0.077898	0.000000	0.000000	0.000000	0.0	0.000000	...	0.000000	0.000000	0.0	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.339606	0.000000	0.141475	0.000000	0.152281	0.497438	0.0	0.000000	...	0.000000	0.162098	0.0	0.000000	0.0000	0.304561	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.095507	0.000000	0.000000	0.0	0.000000	...	0.498249	0.000000	0.0	0.000000	0.0000	0.000000	0.280844	0.28652	0.071603
4	0.000000	0.000000	0.000000	0.143627	0.000000	0.000000	0.000000	0.000000	0.0	0.499054	...	0.000000	0.000000	0.0	0.287255	0.1522	0.000000	0.000000	0.000000	0.000000
...
106	0.000000	0.645764	0.000000	0.000000	0.000000	0.293275	0.000000	0.000000	0.0	0.000000	...	0.000000	0.000000	0.0	0.000000	0.0000	0.000000	0.095822	0.000000	0.000000
107	0.096601	0.000000	0.000000	0.000000	0.000000	0.133942	0.000000	0.000000	0.0	0.146395	...	0.000000	0.000000	0.0	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000
108	0.000000	0.200046	0.000000	0.200046	0.579130	0.000000	0.000000	0.000000	0.0	0.000000	...	0.000000	0.000000	0.0	0.000000	0.0000	0.415574	0.000000	0.000000	0.000000
109	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	...	0.000000	0.000000	0.0	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000
110	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	...	0.000000	0.000000	0.0	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000

111 rows x 107 columns

Appendix D

Implementation of the TF-IDF vectorizer and SVD function *linalg.svd()* from the scikitlearn package.

```
# TF-IDF vectorizer
vectorizer = CountVectorizer(stop_words=remove,min_df=0.1,max_df=0.7)
V = vectorizer.fit_transform(tay_track_lyric)
V = pd.DataFrame(V.toarray(), columns=vectorizer.get_feature_names())
vocab = np.array(vectorizer.get_feature_names())

/opt/conda/lib/python3.10/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names_out is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names_out is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)

%time U, S, Vt = linalg.svd(V, full_matrices=False)
S

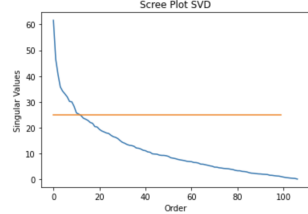
CPU times: user 190 ms, sys: 446 ms, total: 636 ms
Wall time: 200 ms
array([[61.63003829, 46.60376603, 40.66234268, 35.82680724, 34.20718373,
        33.13788433, 31.98830558, 30.22306611, 30.05621776, 28.26764948,
        25.78384195, 25.32659127, 24.7471778 , 23.75041418, 23.34931289,
        22.86115383, 22.07935657, 21.74824608, 20.49432167, 20.32822328,
        19.4010284 , 18.79877021, 18.42022354, 18.01343755, 17.82483345,
        17.12435557, 16.57801217, 16.31989249, 15.83795232, 15.04677523,
        14.3807834 , 14.03131545, 13.54014649, 13.2545208 , 13.13774992,
        12.85891144, 12.21116083, 12.09226405, 11.79202457, 11.31642391,
        11.14740612, 10.68986083, 10.55585561, 9.9468665 , 9.86142967,
        9.79620727, 9.46082961, 9.3415574 , 9.32894247, 9.2028965 ,
        8.93500533, 8.4247576 , 8.26120085, 8.07712614, 7.81086943,
        7.6117069 , 7.4637451 , 7.29437975, 7.05943514, 6.97573701,
        6.89842978, 6.59021041, 6.54833245, 6.35732995, 5.95522887,
        5.92111104, 5.70880293, 5.52529325, 5.29981292, 5.13804762,
        4.79001826, 4.76335518, 4.55212542, 4.44178623, 4.29404621,
        4.22157428, 4.15421404, 3.9866511 , 3.84519636, 3.55868107,
        3.39712879, 3.38321773, 3.14400666, 3.07985609, 2.94723334,
        2.65659584, 2.46280582, 2.40756472, 2.29550237, 2.18954137,
        2.14637569, 2.04781642, 1.98851831, 1.92959547, 1.64981154,
        1.60560173, 1.47336755, 1.37288457, 1.30979357, 1.11666099,
        0.92787877, 0.75706036, 0.65956491, 0.55684904, 0.47601587,
        0.41184982, 0.1724708 ]])
```

Appendix E

Python code for Scree plots displaying the behavior of the singular values of matrix S.

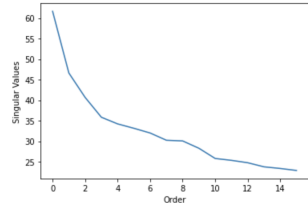
```
# singular values of s
plt.plot(S);
xvals = np.linspace(0, 99)
yvals = [25 for x in xvals]
plt.plot(xvals,yvals)
plt.xlabel('Order')
plt.ylabel('Singular Values')
plt.title('Scree Plot SVD')

Text(0.5, 1.0, 'Scree Plot SVD')



plt.plot(S[:16]);
plt.xlabel('Order')
plt.ylabel('Singular Values')
plt.title('Scree Plot SVD')

Text(0.5, 1.0, 'Scree Plot SVD')

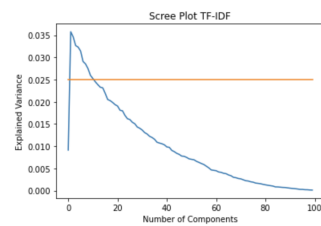

```


Appendix F

Python code for Scree plots displaying the relationship between the number of components and explained variance.

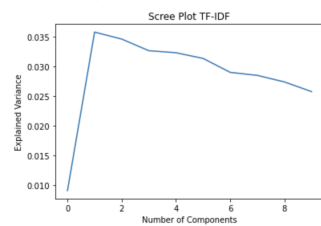
```
# Line plot of the eigenvalues of components
lsa = TruncatedSVD(100)
lsa.fit(df)
lsa_features = lsa.transform(df)
plt.figure()
plt.plot(lsa.explained_variance_ratio_)
xvals = np.linspace(0, 99)
yvals = [0.025 for x in xvals]
plt.plot(xvals, yvals)
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
plt.title('Scree Plot TF-IDF')
```

Text(0.5, 1.0, 'Scree Plot TF-IDF')



```
# Line plot of the eigenvalues of components
lsa = TruncatedSVD(10)
lsa.fit(df)
lsa_features = lsa.transform(df)
plt.figure()
plt.plot(lsa.explained_variance_ratio_)
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
plt.title('Scree Plot TF-IDF')
```

Text(0.5, 1.0, 'Scree Plot TF-IDF')



Appendix G

Function to print a given number of topics from the Vt matrix. The Vt matrix was obtained through SVD in Appendix D.

```
# topics found using SVD
num_top_words=15;

def print_topics(v):
    top_words = lambda t: [vocab[i] for i in np.argsort(t)[-num_top_words:-1]]
    topic_words = ([top_words(t) for t in v])
    return [' '.join(t) for t in topic_words]

print_topics(Vt[:9])

['time stay love baby think say look bad said right come let good night tell',
 'stay time think best hey hand let people wanted hold took times leave turn try',
 'love baby new good say waiting did right come break knew let gone light end',
 'good say stay bad light feel hey did look girl think end love play hand',
 'think time good said car did bad better break light things feel say baby eyes',
 'love time good bad hands light beautiful right life end world high let feel really',
 'baby think bad new hey did really play look break sorry things won head life',
 'think love car home blue place little met new good old beautiful head dress night',
 'love baby home girl think come beautiful things end gone life little said forever hey']
```

Appendix H

Implementation of a function to print topics *print_topics()* which takes a matrix, feature names, and number of topics as input. Implementation of a function to perform NMF called *nmf_function()*. When applied to the document term matrix of Taylor Swift lyrics and specified to give the top nine topics, it gave the following:

```
def print_topics(model, feature_names, no_top_words, topic_names=None):
    for ix, topic in enumerate(model.components_):
        if not topic_names or not topic_names[ix]:
            print("\nTopic ", ix)
        else:
            print("\nTopic: '%s', topic_names[ix], """)
            print(", ".join([feature_names[i]
                             for i in topic.argsort()[::-no_top_words - 1:-1]]))
```

```
# apply NMF
def nmf_function(num_components, matrix, vectorizer):
    nmf = NMF(num_components)
    topic = nmf.fit_transform(matrix)

    index = []
    for i in range(num_components):
        index.append(i)
    topic_word = pd.DataFrame(nmf.components_.round(3),
                              index = index,
                              columns = vectorizer.get_feature_names())

    print(print_topics(nmf, vectorizer.get_feature_names(), 15))
    return topic_word
```

```
# apply NMF
nmf_function(9,df,cv)

Topic 0
girl, home, look, forever, long, say, yeah, drive, place, feel, trying, phone, world, lights, fine

Topic 1
baby, bad, smile, won, really, high, life, let, time, hand, eyes, sorry, coming, new, did

Topic 2
love, beautiful, lights, time, life, knew, saw, long, right, high, heart, little, eyes, end, let

Topic 3
said, think, gone, did, thought, wrong, things, dress, little, head, mind, didn, forever, say, rain

Topic 4
stay, little, think, beautiful, right, hand, won, home, hold, getting, light, let, time, make, night

Topic 5
better, tell, time, night, say, looking, best, sorry, try, dreams, didn, let, run, need, day

Topic 6
yeah, trying, head, end, best, hands, new, right, street, feeling, waiting, dress, night, blue, heard

Topic 7
come, day, feeling, miss, kiss, right, new, need, feel, rain, long, forget, fight, change, sorry

Topic 8
good, remember, bad, coming, looking, break, play, light, right, say, make, heart, room, getting, feel
```

Appendix I

Implementation of a function to get H matrix from NMF decomposition called *nmf_HMatrix()* which takes the number of components, the document term matrix, and vectorizer as input. When specified to give the top nine topics, it gave the following:

```
# get H matrix
def nmf_HMatrix(num_components, doc_text_matrix, vectorizer):
    nmf = NMF(num_components)
    doc_topic = nmf.fit_transform(doc_text_matrix)

    idx = []
    for i in range(num_components):
        idx.append(i)
    H = pd.DataFrame(doc_topic.round(3),
                     index = song_titles,
                     columns = idx)

    return H

h9 = nmf_HMatrix(3,df,cv)
h9
```

	0	1	2	3	4	5	6	7	8
Tim McGraw	0.000	0.000	0.000	0.346	0.057	0.025	0.049	0.000	0.000
Picture To Burn	0.006	0.207	0.057	0.000	0.000	0.182	0.023	0.016	0.026
Teardrops On My Guitar	0.061	0.045	0.221	0.013	0.073	0.124	0.000	0.000	0.033
A Place In This World	0.155	0.000	0.000	0.000	0.000	0.000	0.092	0.002	0.000
Cold as You	0.000	0.020	0.000	0.061	0.007	0.079	0.004	0.181	0.045
...
Miss Americana & the Heartbreak Prince	0.082	0.141	0.008	0.034	0.000	0.000	0.037	0.061	0.143
Paper Rings	0.015	0.097	0.051	0.004	0.012	0.104	0.043	0.095	0.000
Soon You'll Get Better (feat. Dixie Chicks)	0.000	0.038	0.000	0.000	0.024	0.463	0.000	0.000	0.000
The Archer	0.000	0.000	0.000	0.000	0.314	0.000	0.068	0.111	0.065
You Need to Calm Down	0.000	0.000	0.000	0.000	0.002	0.138	0.025	0.075	0.016

111 rows x 9 columns

Appendix J

Python code for using the cosine similarity of the H matrix, found in Appendix I, to find similarity among Taylor Swift songs based on their lyrics.

```
# cosine similarity
doc_similarity_matrix9 = pd.DataFrame(cosine_similarity(h9))
doc_similarity_matrix9.head(15)
```

	0	1	2	3	4	5	6	7	8	9	...	101	102	103	104	105	106	107	108	
0	1.000000	0.056294	0.119737	0.070456	0.313695	0.072861	0.008973	0.107140	0.920081	0.832942	...	0.179817	0.196053	0.071475	0.320370	0.059690	0.165581	0.103385	0.078388	0.172
1	0.056294	1.000000	0.581425	0.060102	0.375359	0.667763	0.798958	0.149444	0.240009	0.137443	...	0.783860	0.071670	0.216307	0.294965	0.727464	0.543642	0.848628	0.697116	0.057
2	0.119737	0.581425	1.000000	0.189581	0.229217	0.464801	0.277363	0.771028	0.105503	0.423399	...	0.386574	0.190126	0.686642	0.654414	0.174330	0.285946	0.594651	0.472979	0.261
3	0.070456	0.060102	0.189581	1.000000	0.019046	0.011943	0.012711	0.000000	0.030192	0.102515	...	0.197655	0.630658	0.264255	0.767371	0.036105	0.389906	0.194444	0.000000	0.103
4	0.313695	0.375359	0.229217	0.019046	1.000000	0.401170	0.145459	0.021964	0.301670	0.461343	...	0.220712	0.032400	0.016369	0.205345	0.111366	0.458431	0.709459	0.379175	0.346
5	0.072861	0.667763	0.464801	0.011943	0.401170	1.000000	0.114500	0.000000	0.008645	0.001192	...	0.268199	0.019214	0.016696	0.182304	0.026252	0.110346	0.576186	0.986146	0.032
6	0.008973	0.798958	0.277363	0.012711	0.145459	0.114500	1.000000	0.068049	0.325141	0.087346	...	0.855979	0.020274	0.092975	0.163335	0.987348	0.655311	0.600764	0.159519	0.020
7	0.107140	0.149444	0.771028	0.000000	0.021964	0.000000	0.068049	1.000000	0.000000	0.326682	...	0.282518	0.343595	0.636845	0.467149	0.000000	0.025796	0.249135	0.034422	0.605
8	0.920081	0.240009	0.105503	0.030192	0.301670	0.008645	0.325141	0.000000	1.000000	0.825484	...	0.340638	0.008639	0.000000	0.268541	0.386784	0.351832	0.195843	0.026827	0.000
9	0.832942	0.137443	0.423399	0.102515	0.461343	0.001192	0.087346	0.326682	0.825484	1.000000	...	0.106401	0.028544	0.374951	0.450105	0.102341	0.275514	0.300996	0.003698	0.079
10	0.795718	0.145915	0.185300	0.224683	0.601077	0.059607	0.141124	0.024322	0.815719	0.826951	...	0.207539	0.063269	0.040062	0.514821	0.170680	0.604986	0.271415	0.009002	0.176
11	0.131294	0.578580	0.425349	0.450017	0.139016	0.833989	0.078276	0.009996	0.007759	0.025637	...	0.332000	0.480377	0.263751	0.433185	0.009279	0.160036	0.605988	0.828413	0.113
12	0.176252	0.384652	0.854608	0.000274	0.132817	0.033686	0.344083	0.815158	0.222085	0.546444	...	0.319360	0.101795	0.790041	0.527262	0.258557	0.327520	0.413605	0.030634	0.222
13	0.000000	0.141257	0.326025	0.000000	0.204115	0.141052	0.088427	0.196757	0.000000	0.115843	...	0.088369	0.000000	0.256040	0.422297	0.000000	0.606192	0.072967	0.000000	0.18
14	0.226347	0.363149	0.225984	0.464369	0.175497	0.370925	0.137801	0.131525	0.071989	0.042878	...	0.441762	0.848623	0.471086	0.422243	0.090999	0.206418	0.470082	0.367057	0.357

15 rows x 111 columns

```
doc_similarity_matrix9[0].nlargest(5)

0      1.000000
83     0.999991
93     0.999979
58     0.999865
14     0.999705
Name: 0, dtype: float64

tay_track_title[0],tay_track_title[83],tay_track_title[93],tay_track_title[58],tay_track_title[14]

('Tim McGraw', 'Delicate', 'New Year's Day', 'Starlight', 'Fearless')
```

```
doc_similarity_matrix9[13].nlargest(5)
```

```
13    1.000000
44    1.000000
55    1.000000
73    0.998069
7     0.996488
Name: 13, dtype: float64
```

```
tay.track_title[44],tay.track_title[55],tay.track_title[73]
```

```
('State of Grace', 'Sad Beautiful Tragic', 'This Love')
```