Topic Modeling with Singular Value Decomposition and Non-negative Matrix Factorization

Shu Xia <sxia38@wisc.edu>
Ziqian Deng <zdeng67@wisc.edu>

May 1, 2020
ECE/CS/ME 532 Spring 2020

1 Abstract

Natural Language Processing (NLP) is a rapidly advancing branch of unsupervised machine learning concerned with the ability of a computer to understand, analyze, manipulate, and potentially generate human language. Applications of NLP include Topic Modeling, where abstract "topics" that occur in a collection of documents need to be detected; Machine Translation, where text or speech are translated from one language to another; Sentiment Analysis, which allows businesses to identify customer sentiment toward products, brands or services in online conversations and feedback; Spam Filter, where unsolicited and unwanted email and messages are prevented from getting into user' inbox, etc.

And just like in many other machine learning fields, matrix decomposition is one of the most important underlying methods in NLP. To help us start the study in NPL, this activity will introduce how to apply Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) to conduct Topic Modeling on text data.

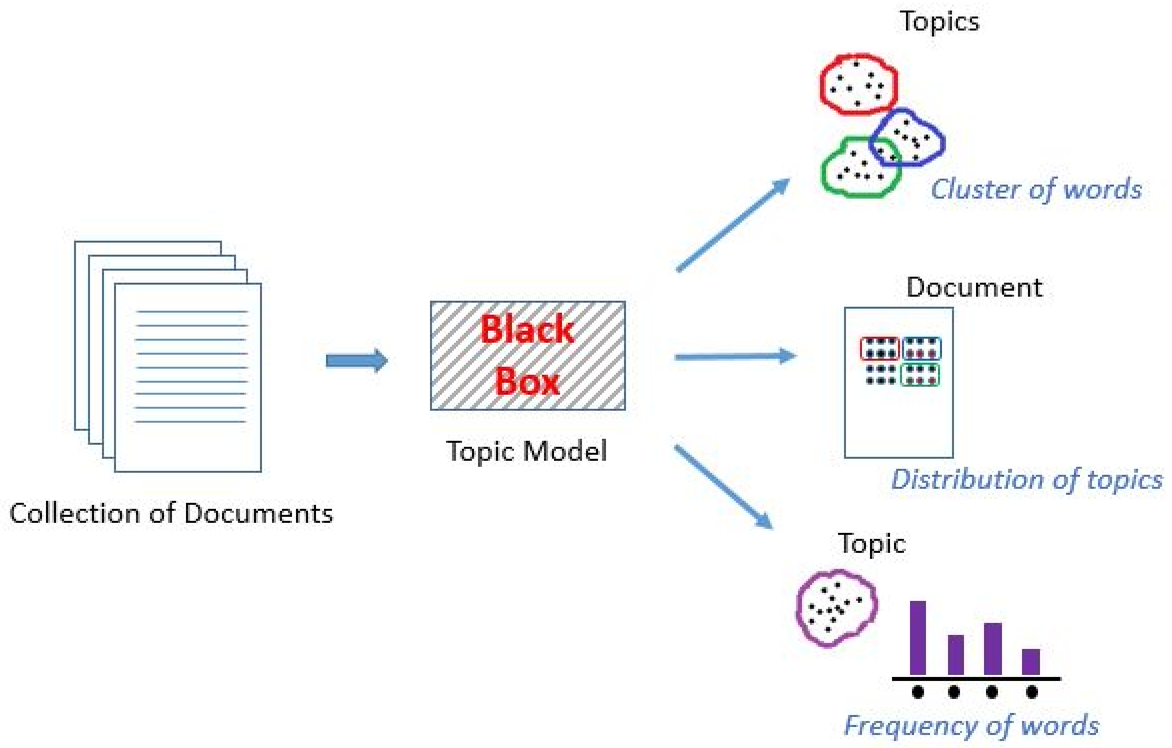
1.1 Learning objectives

- Have a knowledge of the wide range of applications in NLP
- Understand what is topic modeling, and learn how to represent topic modeling problems using matrix method.
- Learn to conduct basic text preprocessing (Tokenization, Stemming, Lemmatization, etc)
- Refresh their knowledge in SVD and apply SVD to topic modeling problems
- Learn the Non-negative Matrix Factorization (NMF) and apply NMF to topic modeling problems
- Compare these two matrix factorization techniques and explain the advantages of NMF

2 Background

2.1 What is a topic model

A Topic Model can be defined as an unsupervised technique to discover topics across various text documents. These topics are abstract in nature, i.e., words which are related to each other form a topic. Similarly, there can be multiple topics in an individual document [4]. For the time being, let's understand a topic model as a black box, as illustrated in the below figure:



This black box (topic model) forms clusters of similar and related words which are called topics. These topics have a certain distribution in a document, and every topic is defined by the proportion of different words it contains.

2.1.1 Ethical issues in Natural Language Processing

Throughout the years, ethical issues have been a controversial topic in the field of Machine Learning and Natural Language Processing is not an exception. For example, the privacy issue derived from text data collecting process has sparked a discussion on the proper practices during the NLP tasks. To learn about the wide range of ethical issues in NLP, you can refer to the article written by Jochen L. Leidner and Vassilis Plachouras: Ethics Best Practices for Natural Language Processing [8].

2.2 Introduction to basic text preprocessing

2.2.1 Tokenization, Stemming, Lemmatization

We will start by learning the basic text preprocessing techniques. Cleaning and preparation are crucial for many tasks, and NLP is no exception. Text preprocessing is usually the first step you'll take when faced with an NLP task [2]. Without preprocessing, our computer interprets "the", "The", and "<p>The" as entirely different words. There is a lot we can do here, depending on the formatting you need. Common tasks include:

Noise removal — stripping text of formatting (e.g., HTML tags, stop words).
Tokenization — breaking text into individual words.
Normalization — cleaning text data in any other way:

- ***Stemming*** is a blunt axe to chop off word prefixes and suffixes. "booing" and "booed" become "boo", but "sing" may become "s" and "sung" would remain "sung."
- ***Lemmatization*** is a scalpel to bring words down to their root forms. For example, NLTK's savvy lemmatizer knows "am" and "are" are related to "be."

In Python, various packages will do most of the preprocessing tasks for us. Use the given code to try different methods to deal with text data.

Warm-up Question 1: try lemmatizing and stemming the following collections of words. (Since the CAE Jupyter system doesn't allow us to install the required packages, you don't need to rerun the code for this activity. We have run the code locally and are keeping the results for you to read through)

- fly, flies, flying
- organize, organizes, organizing
- universe, university

Did you find anything interesting? Did the meaning of the individual word change a lot after lemmatization and stemming? (Hint: The running results can also be viewed in section 5 Appendix, in case that the results in the Jupyter file are lost.)

2.2.2 Represent text documents with matrix

Next, we will learn to use the term-document matrix to represent the text data. A document-term matrix or term-document matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a document-term matrix, rows correspond to documents in the collection and columns correspond to terms. For example:

Document 1 = " the cat sat on my face"
Document 2 = " the dog sat on my bed"

The simplest strategy is to create a vector of all possible words, and for each document count how many times each word appears. In this way, the mathematical matrix will be:

	bed	cat	dog	face	my	on	sat	the
Document 1	0	1	0	1	1	1	1	1
Document 2	1	0	1	0	1	1	1	1

The problem with this counting strategy is that we use a lot of words commonly, that just don't mean much. In fact, the most commonly used word in English "the" makes up 7% of the words we speak, which is double the frequency of the next most popular word "of". So, if we construct our document matrix out of counts, then we end up with numbers that don't contain much information, unless our goal is to see who uses "the" most often.

Rather than just counting, a technique in NLP known as ***term frequency-inverse document frequency (tf-idf)*** is better. TF-IDF score of a word is a numerical statistic for ranking its importance by deprioritizing the most common words and prioritizing less frequently used terms as topics.

The TF-IDF score of word, *w*, is:

$$tf(w) \ * \ idf(w)$$

Where

$$tf(w) \ = \ \frac{Number\ of\ times\ the\ word\ appears\ in\ a\ document}{Total\ number\ of\ words\ in\ the\ document}$$

$$idf(w) \ = \ log(\frac{Number\ of\ documents}{Number\ of\ documents\ that\ contain\ word\ w})$$

If we use the TF-IDF for the previous example, the matrix will become:

	bed	cat	dog	face	my	on	sat	the
Document 1	0	0.115525	0	115525	0	0	0	0
Document 2	0.115525	0	0.115525	0	0	0	0	0

Compared with the previous matrix, this new matrix only gave scores to meaningful words such as “cat” and “face” for document 1 and “bed” and “dog” for document 2, while it gave 0 to those less important words appearing commonly and equally in both documents. Therefore, this matrix is easier for human and machine to interpret.

Warm-up Question 2: try different strategies to count words in three of Donald Trump’s tweets and turn them into a matrix.(The text data has already been tokenized for you)

1. Use the simple counting strategy first
2. Then complete the line of code that is unfinished and try the TF-IDF strategy

---

2.3 Singular Value Decomposition and Latent Semantic Analysis

Now we will dive into using Singular Value Decomposition (SVD) to address Topic Modeling. Sometimes we hear topic modeling as ***Latent Semantic Analysis (LSA)*** which is an algorithm that leverages SVD to produce a set of concepts related to the documents and terms.

2.3.1 Overview of Latent Semantic Analysis

All languages have their own intricacies and nuances which are quite difficult for a machine to capture (sometimes they're even misunderstood by us humans!). This can include different words that mean the same thing, and also the words which have the same spelling but different meanings.

For example, consider the following two sentences:

1. I liked his last novel quite a lot.
2. We would like to go for a novel marketing campaign.

In the first sentence, the word 'novel' refers to a book, and in the second sentence it means new or fresh.

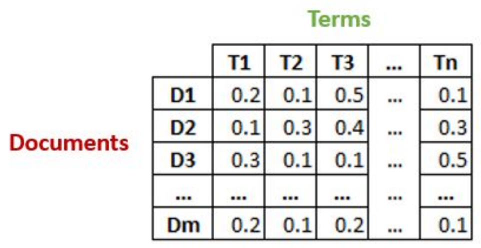
We can easily distinguish between these words because we are able to understand the context behind these words. However, a machine would not be able to capture this concept as it cannot understand the context in which the words have been used. This is where Latent Semantic Analysis (LSA) comes into play as it attempts to leverage the context around the words to capture the hidden concepts, also known as topics.

So, simply mapping words to documents won't really help. What we really need is to figure out the hidden concepts or topics behind the words. LSA is one such technique that can find these hidden topics. Let's now deep dive into the inner workings of LSA.

2.3.2 Singular Value Decomposition in Latent Semantic Analysis

Let's say we have *m* number of text documents with *n* number of total unique terms (words). We wish to extract *k* topics from all the text data in the documents. The number of topics, k, has to be specified by the user. Here are the steps to conduct the LSA:

- Generate a document-term matrix of shape m x n having TF-IDF scores.


Terms
Documents

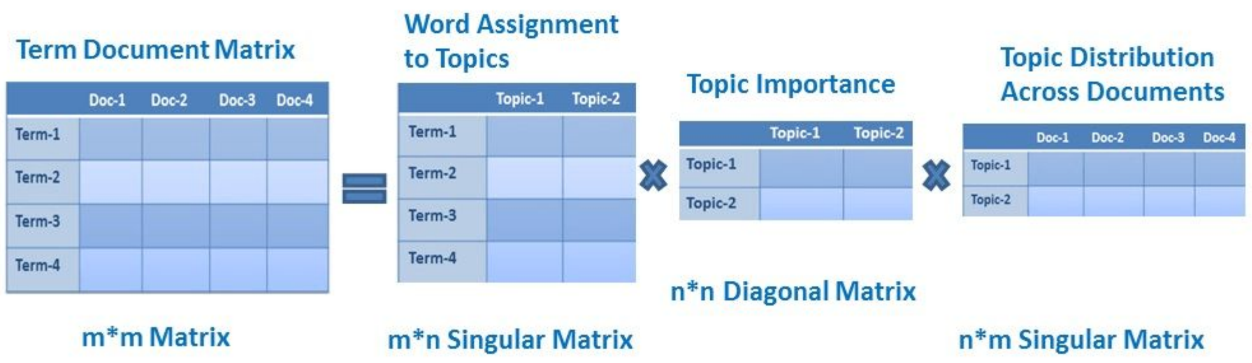
	T1	T2	T3	...	Tn
D1	0.2	0.1	0.5	...	0.1
D2	0.1	0.3	0.4	...	0.3
D3	0.3	0.1	0.1	...	0.5
...	...	...	...	...	...
Dm	0.2	0.1	0.2	...	0.1

- Then, we will reduce the dimensions of the above matrix to k (number of desired topics) dimensions, using SVD.

We have learned SVD: The $m \times n$ matrix A with rank r $\leq$ $min\{m, n\}$ has a SVD of the form:

$$A_{m\times n} = \sum_{i=1}^{r} \sigma_i u_i v_i^T = U_{m\times r} \Sigma_{r\times r} V_{r\times n}^T$$

where U is a $m \times r$ matrix with orthonormal columns, V is a $n \times r$ matrix with orthonormal columns, and S is an $r \times r$ diagonal matrix with the entries sorted in decreasing order. The entries of the S matrix are the singular values, and the U and V matrices are the left and right singular vectors, corresponding to term and document vectors. This is simply a re-representation of the A matrix using orthogonal indexing dimensions [3].



Specifically, LSA uses the truncated SVD, in which the k largest singular values are retained, and the remainder set to 0. The resulting representation is the best k-dimensional approximation to the original matrix in the least-squares sense. Each document and term is now represented as a k-dimensional vector in the space derived by the SVD.

Warm-up Question 3: In the previous question, we define our own functions to compute the TF-IDF scores, but actually, Python has various packages such as scikit-learn that can help us compute the TF-IDF quickly. Please follow the provided code and learn to conduct LSA with SVD on a text data *posts_parsed.csv* which contains posts from a data science forum.

1. After printing out the topics and their related terms, could you summarize the topics using your own words? Try to use a sentence or a phrase to summarize topic 3, 4 and 7. For example, the topic 0 could be summed up as "using big data".

2.4 Non-negative Matrix Factorization

In this section, we will briefly introduce another decomposition algorithm — Non-negative Matrix Factorization for solving Topic Modeling problem.

2.4.1 Non-negative Matrix Factorization Problem

The non-negative matrix factorization (NMF) problem can be expressed formally as the following:

> **NMF problem:** Given a non-negative matrix $A \in R^{m \times n}$ and a positive integer $k < minm, n$, find non-negative matrices $W \in R^{m \times k}$ and $H \in R^{k \times n}$ to minimize the cost function
>
> $$f(W, H) = \frac{1}{2} \|A - WH\|_F^2 \qquad (1)$$

The product WH is called a non-negative matrix factorization of A. Through NMF, we get a low-rank (at most rank k) approximation, WH, of the original matrix A.

There are two note-worthy properties about NMF that we would like to mention at this moment. First is that, NMF can be written column by column as:

$$a_i = Wh_i, \quad i \in 1, 2, ..., n \qquad (2)$$

Where a_i is the i-th column vector of matrix A, and h_i is the corresponding column vector of matrix H. That is to say, each data vector a_i is approximated by a linear combination of the columns of W, weighted by the components of h_i. Therefore, W can be regarded as a basis that is optimized for the linear approximation of data A. Usually, the parameter k is small compared to the dimension of A, so NMF will be a good approximation if there exists latent structures in vectors of data matrix A.

Second property of NMF is that, negative entries are not allowed in W and H. For this reason, NMF is intuitively a part-based representation, since the whole is formed by combining the parts. These two features make NMF a good method for text classification, we will discuss this with details in later sections.

2.4.2 Basic Algorithms for Solving NMF Problem

Though NMF problems have a simple form, it has been proven that finding the exact solution of these problems in general is NP-hard. Nonetheless, it is possible to approximately solve the NMF problems using iterative measures. In this section, we will introduce three fundamental algorithms based on which other advanced algorithms are built. They are multiplicative update algorithm, gradient descent algorithm and alternating least squares algorithm.

Multiplicative Update Algorithm for NMF:

```
$W = rank(m,k)$; initialize $W$ as random dense matrix
$H = rank(m,k)$; initialize $H$ as random dense matrix
for $i = 1 : maxiter$
    $H_{ij} \leftarrow H_{ij} \frac{(W^T A)_{ij}}{(W^T W H)_{ij} + \epsilon}$
    $W_{ij} \leftarrow W_{ij} \frac{(A H^T)_{ij}}{(W H H^T)_{ij} + \epsilon}$
end
```

Basic Gradient Descent Algorithm for NMF:

```
$W = rank(m,k)$; initialize $W$
$H = rank(m,k)$; initialize $H$
for $i = 1 : maxiter$
    $H_{ij} \leftarrow H_{ij} - \epsilon_H \frac{\partial f}{\partial H}$
    $W_{ij} \leftarrow W_{ij} - \epsilon_W \frac{\partial f}{\partial W}$
end
```

Basic Alternative Least Squares (ALS) Algorithm for NMF:

```
$W = rank(m,k)$; initialize $W$ as random dense matrix
for $i = 1 : maxiter$
    (LS)          $H \leftarrow (W^T W)^{-1} W^T A$
    (NONNEG)      Set all negative elements in $H$ to 0
    (LS)          $W \leftarrow ((H H^T)^{-1} W A)^T$
    (NONNEG)      Set all negative elements in $W$ to 0
end
```

All above three algorithms can only find local solution but not global optimal, due to the fact that the cost function $f = \frac{1}{2}\|A - WH\|_F^2$ is not convex in both variables W and H together. In addition, for the last two algorithms, we need to imply a "projection" step to ensure the non-negative property of W and H, that is, to set all non-negative entries to 0 after each iteration step. Due to limitation of space, we are not going to extend our discussion into details on the algorithms. The presentation here is simply to provide some basic understanding on how to deal with the NMF problem. Of course, there exist more advanced and faster algorithms, and we will learn to use tool-kit *sciki-learnt* to solve the NMF problem in later sections.

2.4.3 Application of NMF on Text Classification

As we have seen in previous sections, a collection of documents can be represented as data matrix A, each column vector represents a document in word-frequency space. And we have learned how to classify it using singular value decomposition. In this section, we will learn to apply NMF in text classification, and compare the method of NMF with SVD.

The NMF problem for text classification can be expressed as the following:

> Given a collection of document represented by a data matrix $A \in R^{m \times n}$, find the non-negative matrix factorization
>
> $$A_{m \times n} \approx W_{m \times k} H_{k \times n}$$
>
> that minimize the cost function in (1). In the above equation, k is a positive integer that $k \ll min(m, n)$.

Remember that the non-negative matrix factorization can be written column by column, it leads to a natural interpretation for W and H:

> W -- basis document matrix
> H -- weight matrix

To finish up this section, we will briefly compare NMF and SVD in topic modeling problems.

- Both SVD and NMF can be used to solve text classification problems. The interpretation of decomposed matrices as a result are similar.
- NMF only allows non-negative entries, which leads to the interpretation that it uses parts to represent the whole. While SVD gives a more "deeper" factorization based on the structural features of the entire data set.
- The solution to SVD is unique; while the solution to NMF is not. Since topic modeling problems are often presented as un-supervised learning problems, one might need to try different rank approximations (different k) using NMF.

Warm-up Question 4:

1. Prove that NMF is non-unique. (Hint: Insert matrix (BB^{-1}) between W and H .)

2. Suppose that, in the NMF problem described in equation (1), the matrix W is fixed. Prove that the solution to the least square problem $min_H \|A - WH\|_F^2$ is:

$$H = (W^TW)^{-1}W^TA$$

   (Hint: Use the property that NMF can be written column by column.)

3 Main Activity

3.1 Main Activity 1

In this activity, the topic modeling algorithm — Latent Semantic Analysis is further explored by applying it to a corpus of over one million news article headlines published by the ABC.

Please run the given script "Main_Activity_1.ipybn" in Jupyter Notebook, and try to answer the activity questions. Hint: if this large dataset makes your machine take a long time to run LSA code, we suggest that you take a sample of the whole dataset to run LSA using the following code: "small_text_sample = reindexed_data.sample(n=10000, random_state=0).values"

Run through the code and explore the data using the provided methods. The given LSA model uses **CountVectorizer** as the word counting strategy which is the simplest kind. Try to use the code you learn from the warm-up questions to run the LSA model again using **TfidfVectorizer**. Compare the results from both counting strategies.

3.2 Main Activity 2

In this section, we will try to analyze some text data using both SVD and NMF. The data comes from *scikit-learn*'s "the 20 newsgroups text dataset", we will choose a portion of articles from this database, and provide a script to analyze the chosen data.

Please run the given script "Main_Activity_2.ipybn" in Jupyter Notebook, and try to answer the activity questions. Make sure to upload the data package "20news-bydate_py3.pkz" into the same directory as the script file. Also, if you are using your own computer, please install sklearn and pandas packages.

Questions:

A. Compare the result from SVD and NMF, especially the topic-encoded matrix. How do you interpret the negative values in the matrix in SVD?
B. Remember that NMF is not unique with respect to component number k. Try different values of k and see what result you would get.

4 References

[1] Jason Brownlee (2017) What Is Natural Language Processing?[Online]. machinelearningmastery.com/natural-language-processing/

[2] Badreesh Shetty (2018) Natural Language Processing(NLP) for Machine Learning [Online]. towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b

[3] Thomas K Landauer and Susan Dumais (2008) Latent semantic analysis. Scholarpedia, 3(11):4356., revision #142371

[4] Prateek Joshi (2018)Text Mining 101: A Stepwise Introduction to Topic Modeling using Latent Semantic Analysis (using Python) [Online]. analyticsvidhya.com/blog/2018/10/stepwise-guide-topic-modeling-latent-semantic-analysis/

[5] Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. Nature, 401(6755), 788-791.

[6] Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In Advances in neural information processing systems (pp. 556-562).

[7] Berry, M. W., Browne, M., Langville, A. N., Pauca, V. P., & Plemmons, R. J. (2007). Algorithms and applications for approximate nonnegative matrix factorization. Computational statistics & data analysis, 52(1), 155-173.

[8] Leidner, J.L., & Plachouras, V. (2017). Ethical by Design: Ethics Best Practices for Natural Language Processing. EthNLP@EACL.

5 Appendix

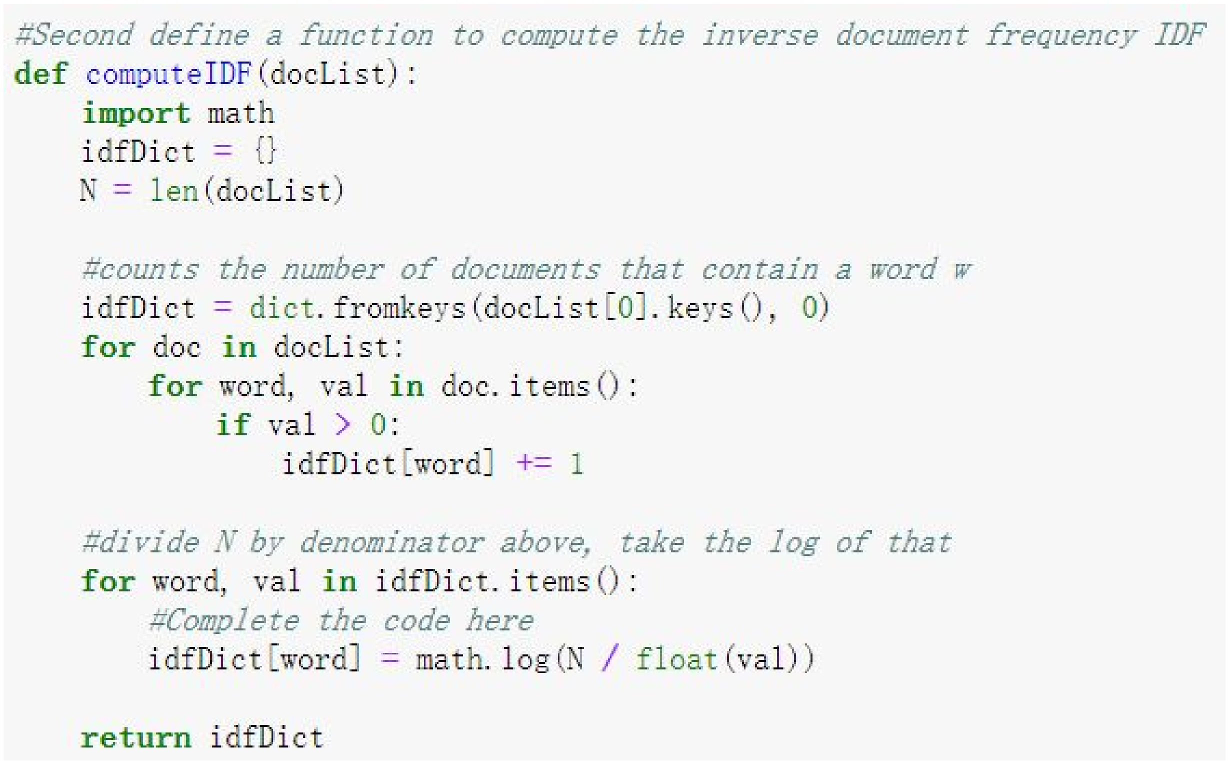
Solutions to Warm-up Questions:

Warm-up question 1
For the first set of words “fly, flies, flying”, they are all converted to “fli” after stemming and the root “fly” after lemmatization. The meanings of the words do not change.

For the second set of words “organize, organizes, organizing”, stemming has converted them to “organ” whose meaning is completely different.

For the last set of words “universe, university”, stemming and lemmatization do not affect them too much.

Warm-up question 2



```
#Second define a function to compute the inverse document frequency IDF
def computeIDF(docList):
    import math
    idfDict = {}
    N = len(docList)

    #counts the number of documents that contain a word w
    idfDict = dict.fromkeys(docList[0].keys(), 0)
    for doc in docList:
        for word, val in doc.items():
            if val > 0:
                idfDict[word] += 1

    #divide N by denominator above, take the log of that
    for word, val in idfDict.items():
        #Complete the code here
        idfDict[word] = math.log(N / float(val))

    return idfDict
```

Warm-up question 3
Topic 3: Data science analyzes problem to make better decisions
Topic 4: The goal of data science is to find solution by analyzing data and information
Topic 7: Data science understands people’s insights and predicts trends

Warm-up question 4

1: Prove that NMF is non-unique.
Solution:
For any NMF, A = WH, insert a non-negative none-zero matrix BB^{-1} between W and H:

$$\begin{aligned} A &= W (B B^{-1}) H \\ &= (WB) (B^{-1} H) \end{aligned}$$

We can get a new factorization A = W' H', where W' = WB and H' = B^{-1} H are different from the original W and H.

2. Suppose that, in the NMF problem described in equation (1), the matrix W is fixed. Prove that the solution to the least square problem:

$$min_H \ \|A - WH\|_F^2$$

is $H = (W^T W)^{-1} W^T A$.
Solution:
Use the fact that NMF can be written column by column:

$$\begin{aligned} &min_H \ \|A - WH\|_F^2 \\ &= min_H \ \|\sum_i (a_i - Wh_i)\|_F^2 \\ &= min_H \ \sum_i \|a_i - Wh_i\|^2 \end{aligned}$$

The last step stands because we are calculating the Frobenius norm, so the sum of the norm of all columns equal to the Frobenius norm of the matrix.
Now the least square problem can be treated as a set of individual least square problems:

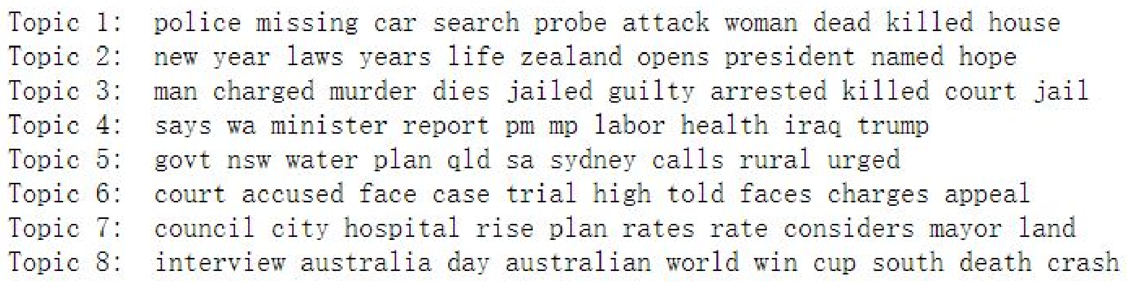
$$min_{h_i} \ \|a_i - Wh_i\|$$

Each has solution $h_i = (W^T W)^{-1} W^T a_i$.
Paste the sub-solutions together, we get $H = (W^T W)^{-1} W^T A$.

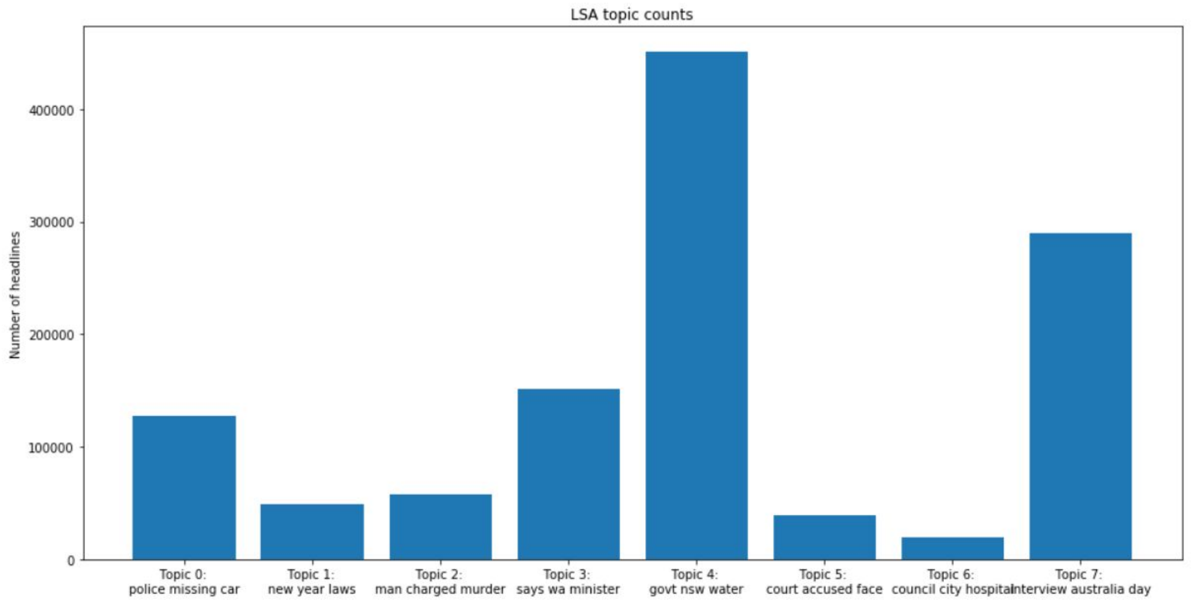
Solutions to Main Activity Questions:

Main Activity 1

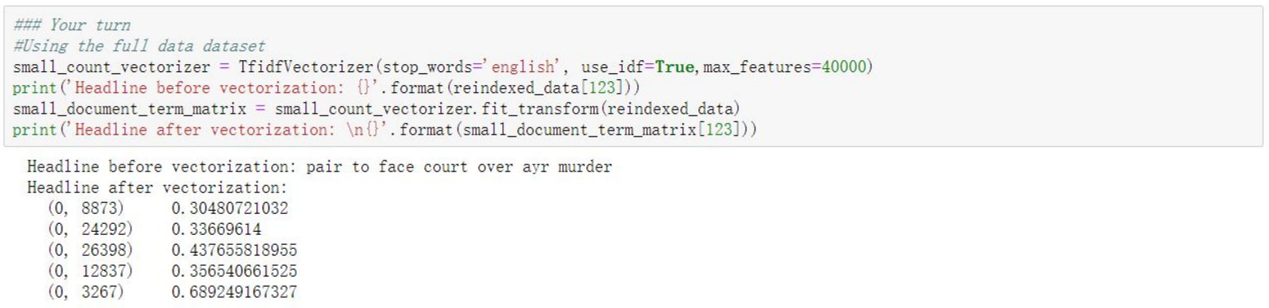
CountVectorizer



```
Topic 1:  police missing car search probe attack woman dead killed house
Topic 2:  new year laws years life zealand opens president named hope
Topic 3:  man charged murder dies jailed guilty arrested killed court jail
Topic 4:  says wa minister report pm mp labor health iraq trump
Topic 5:  govt nsw water plan qld sa sydney calls rural urged
Topic 6:  court accused face case trial high told faces charges appeal
Topic 7:  council city hospital rise plan rates rate considers mayor land
Topic 8:  interview australia day australian world win cup south death crash
```



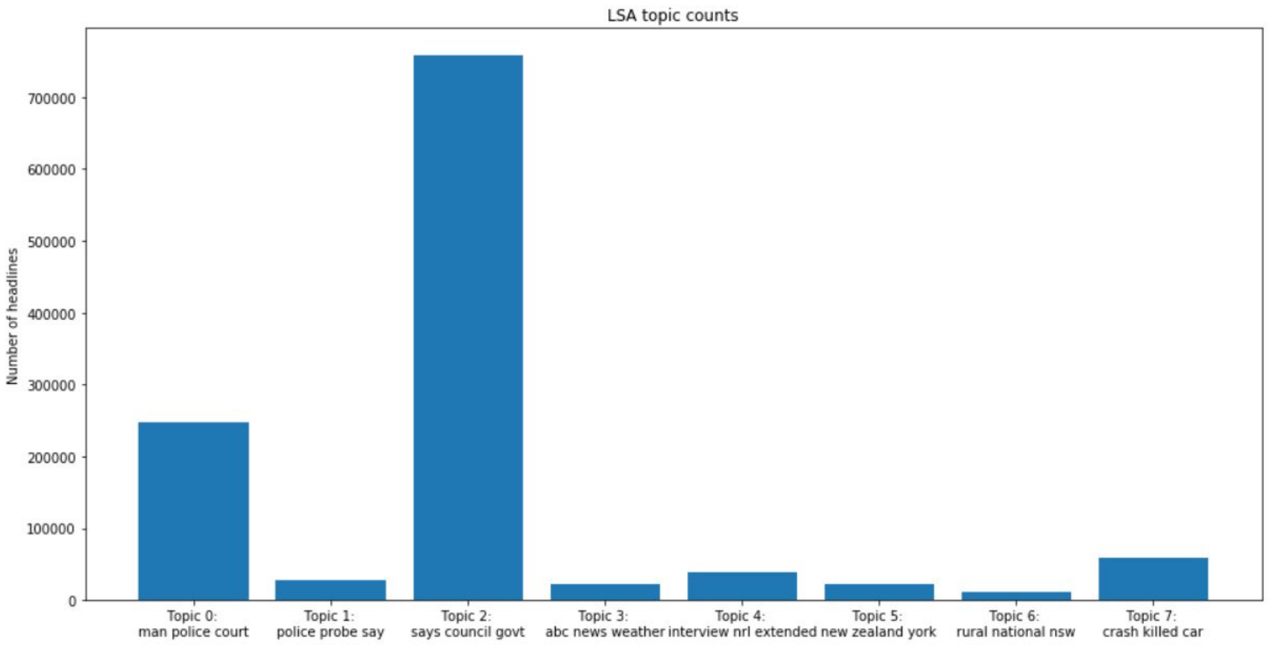
TfidfVectorizer



```
### Your turn
#Using the full data dataset
small_count_vectorizer = TfidfVectorizer(stop_words='english', use_idf=True,max_features=40000)
print('Headline before vectorization: {}'.format(reindexed_data[123]))
small_document_term_matrix = small_count_vectorizer.fit_transform(reindexed_data)
print('Headline after vectorization: \n{}'.format(small_document_term_matrix[123]))
```

```
Headline before vectorization: pair to face court over ayr murder
Headline after vectorization:
  (0, 8873)     0.30480721032
  (0, 24292)    0.33669614
  (0, 26398)    0.437655818955
  (0, 12837)    0.356540661525
  (0, 3267)     0.689249167327
```

```
Topic 1:  man police court death charged murder sydney missing accused attack
Topic 2:  police probe say investigate seek clash drivers protesters hunt warn
Topic 3:  says council govt australia water plan nsw health wa australian
Topic 4:  abc news weather business drum sport rural national market friday
Topic 5:  interview nrl extended michael smith david john james scott clarke
Topic 6:  new zealand york laws year home opens deal centre president
Topic 7:  rural national nsw reporter park sa qld press club parks
Topic 8:  crash killed car dies dead road woman fatal injured plane
```

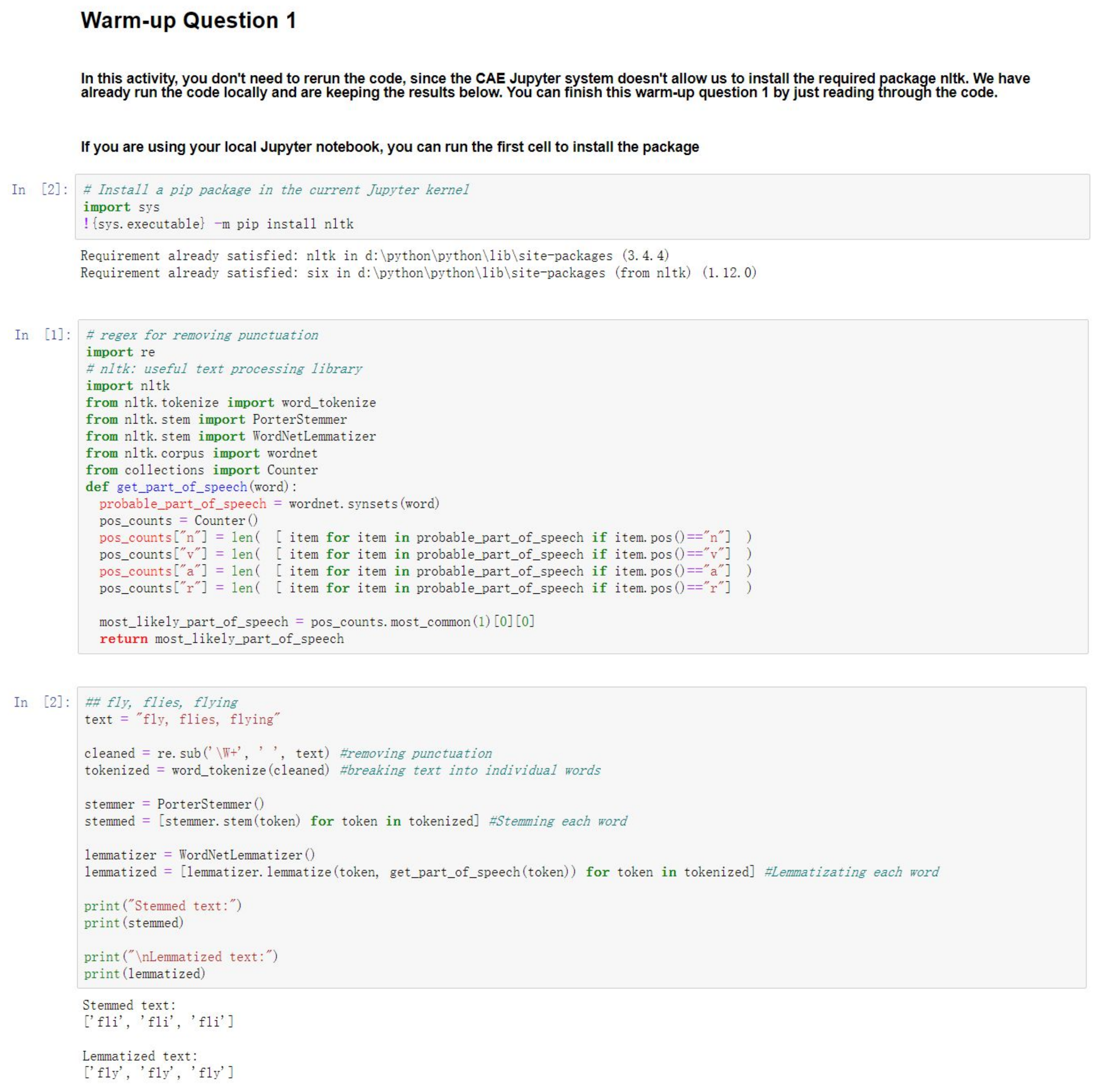


Main Activity 2

1. For the resulting topic-encoded matrix using SVD, a positive entry means the corresponding document is “strong” in the corresponding topic if the entry value is large. While a negative entry can be interpreted as the document is related to the corresponding topic negatively, i.e., the document is unlikely to be classified as consisting of this topic.
   For the NMF topic-encoded matrix, there are only non-negative entries, and their interpretation is much more intuitive. This can be seen as an advantage of NMF compared to SVD.

2. If you try different k in NMF, you will see that the resulting topic-encoded matrices and the top-words of each topic varies with respect to k. Note that the result of SVD is fixed, the only thing we can control over SVD is where to truncate it. On the contrary, NMF provides us some flexibility and control over our result by allowing us to get different results using different k. For unsupervised learning problems, this flexibility might come in handy in many cases.

Warm-up Question 1 running results



Warm-up Question 1

In this activity, you don't need to rerun the code, since the CAE Jupyter system doesn't allow us to install the required package nltk. We have already run the code locally and are keeping the results below. You can finish this warm-up question 1 by just reading through the code.

If you are using your local Jupyter notebook, you can run the first cell to install the package

In [2]:
```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install nltk
```

```
Requirement already satisfied: nltk in d:\python\python\lib\site-packages (3.4.4)
Requirement already satisfied: six in d:\python\python\lib\site-packages (from nltk) (1.12.0)
```

In [1]:
```
# regex for removing punctuation
import re
# nltk: useful text processing library
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from collections import Counter
def get_part_of_speech(word):
  probable_part_of_speech = wordnet.synsets(word)
  pos_counts = Counter()
  pos_counts["n"] = len(  [ item for item in probable_part_of_speech if item.pos()=="n"]  )
  pos_counts["v"] = len(  [ item for item in probable_part_of_speech if item.pos()=="v"]  )
  pos_counts["a"] = len(  [ item for item in probable_part_of_speech if item.pos()=="a"]  )
  pos_counts["r"] = len(  [ item for item in probable_part_of_speech if item.pos()=="r"]  )

  most_likely_part_of_speech = pos_counts.most_common(1)[0][0]
  return most_likely_part_of_speech
```

In [2]:
```
## fly, flies, flying
text = "fly, flies, flying"

cleaned = re.sub('\W+', ' ', text) #removing punctuation
tokenized = word_tokenize(cleaned) #breaking text into individual words

stemmer = PorterStemmer()
stemmed = [stemmer.stem(token) for token in tokenized] #Stemming each word

lemmatizer = WordNetLemmatizer()
lemmatized = [lemmatizer.lemmatize(token, get_part_of_speech(token)) for token in tokenized] #Lemmatizating each word

print("Stemmed text:")
print(stemmed)

print("\nLemmatized text:")
print(lemmatized)
```

```
Stemmed text:
['fli', 'fli', 'fli']

Lemmatized text:
['fly', 'fly', 'fly']
```

```
In [3]: ### organize, organizes, organizing
        text = "organize, organizes, organizing"

        cleaned = re.sub('\W+', ' ', text) #removing punctuation
        tokenized = word_tokenize(cleaned) #breaking text into individual words

        stemmer = PorterStemmer()
        stemmed = [stemmer.stem(token) for token in tokenized] #Stemming each word

        lemmatizer = WordNetLemmatizer()
        lemmatized = [lemmatizer.lemmatize(token, get_part_of_speech(token)) for token in tokenized] #Lemmatizating each word

        print("Stemmed text:")
        print(stemmed)

        print("\nLemmatized text:")
        print(lemmatized)
```

```
Stemmed text:
['organ', 'organ', 'organ']

Lemmatized text:
['organize', 'organize', 'organize']
```

```
In [4]: ### universe, university
        text = "universe, university"

        cleaned = re.sub('\W+', ' ', text) #removing punctuation
        tokenized = word_tokenize(cleaned) #breaking text into individual words

        stemmer = PorterStemmer()
        stemmed = [stemmer.stem(token) for token in tokenized] #Stemming each word

        lemmatizer = WordNetLemmatizer()
        lemmatized = [lemmatizer.lemmatize(token, get_part_of_speech(token)) for token in tokenized] #Lemmatizating each word

        print("Stemmed text:")
        print(stemmed)

        print("\nLemmatized text:")
        print(lemmatized)
```

```
Stemmed text:
['univers', 'univers']

Lemmatized text:
['universe', 'university']
```

Did you find anything interesting? Did the meaning of the individual word change a lot after lemmatization and stemming?