HANDS-ON TUTORIALS

NMF — A visual explainer and Python Implementation

Anupama Garla Mar 18 · 14 min read ★

Gain an intuition for the unsupervised learning algorithm that allows data scientists to extract topics from texts, photos, and more, and build those handy recommendation systems. NMF explanation is followed by a Python Implementation on a toy example of topic modelling on Presidential Inauguration Speeches.

Origins of NMF

> "Is perception of the whole based on perception of its parts?"

Researchers Lee and Seung go on to lay out the mathematical basis of NMF in their 1999 paper in Nature — "**Here we demonstrate an algorithm for non-negative matrix factorization that is able to learn parts of faces and semantic features of text.**"

The beauty of data science is its ability to translate rather philosophical theory into mathematical algorithms that test the usefulness of such theory. NMF answers the question — to what extent a whole is a sum of its parts? And how do we use these parts? Companies typically use these parts to find patterns and associations between

In a business context, topics can be used to segment customers into types and target them differently. Additionally, the document-topic matrix can be used to plot each document in space and provide a system to give recommendations based on similarities and differences.

Let's start looking at NMF in the context of natural language processing where it is often used for topic modeling.

Topic Modeling: How do we extract topics from a large corpus of documents?

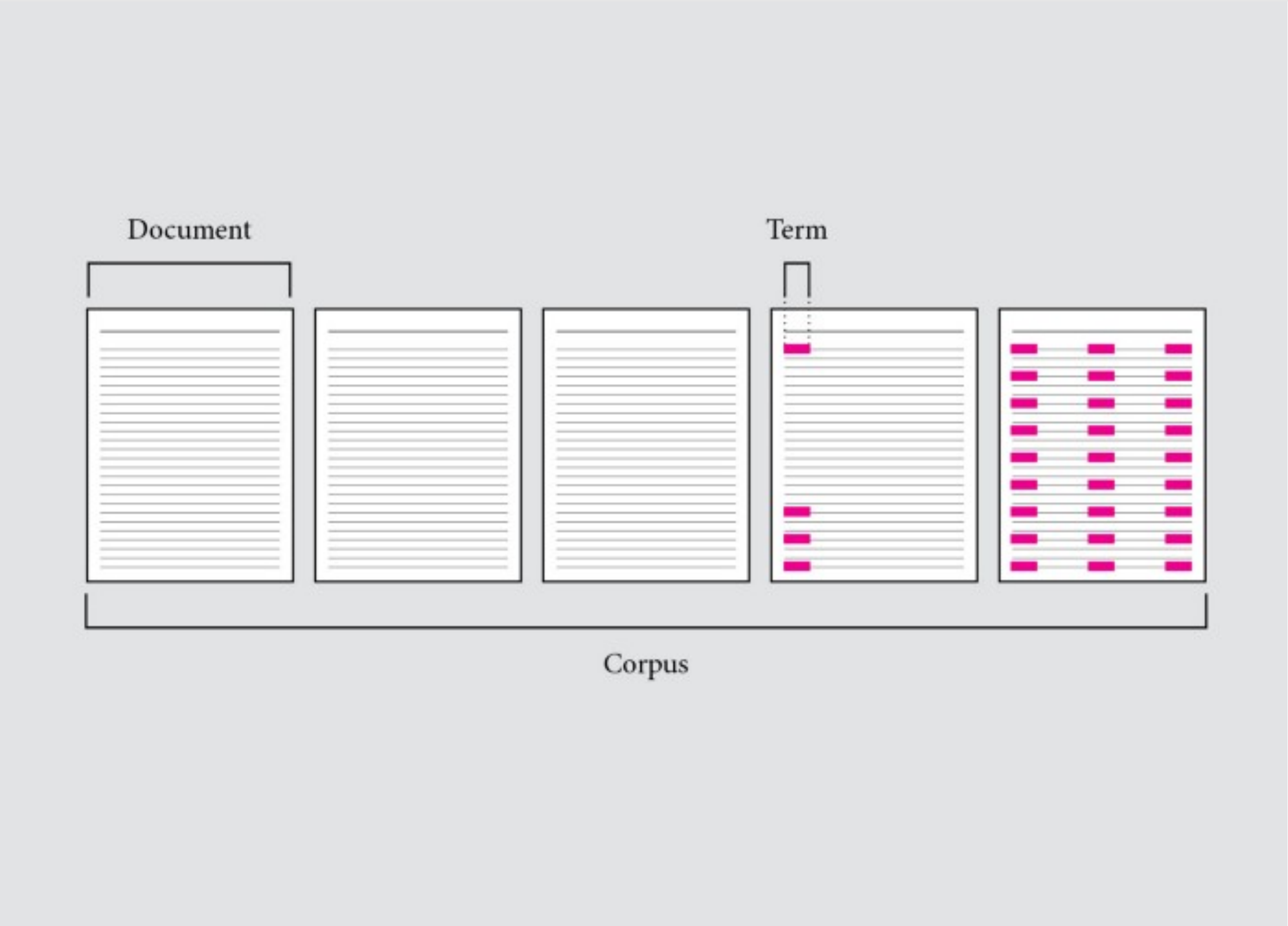
We convert a corpus of documents into numbers to answer questions like:

- What are the prevailing topics in this corpus?
- To what extent is each topic important in each document?
- How well do these extracted topics describe the original document?
- How similar is one document to another?

There are many more questions that can be answered with NMF and natural language processing, but these are the core questions. First, let's understand the terminology data scientists and a lot of our literature like to use.

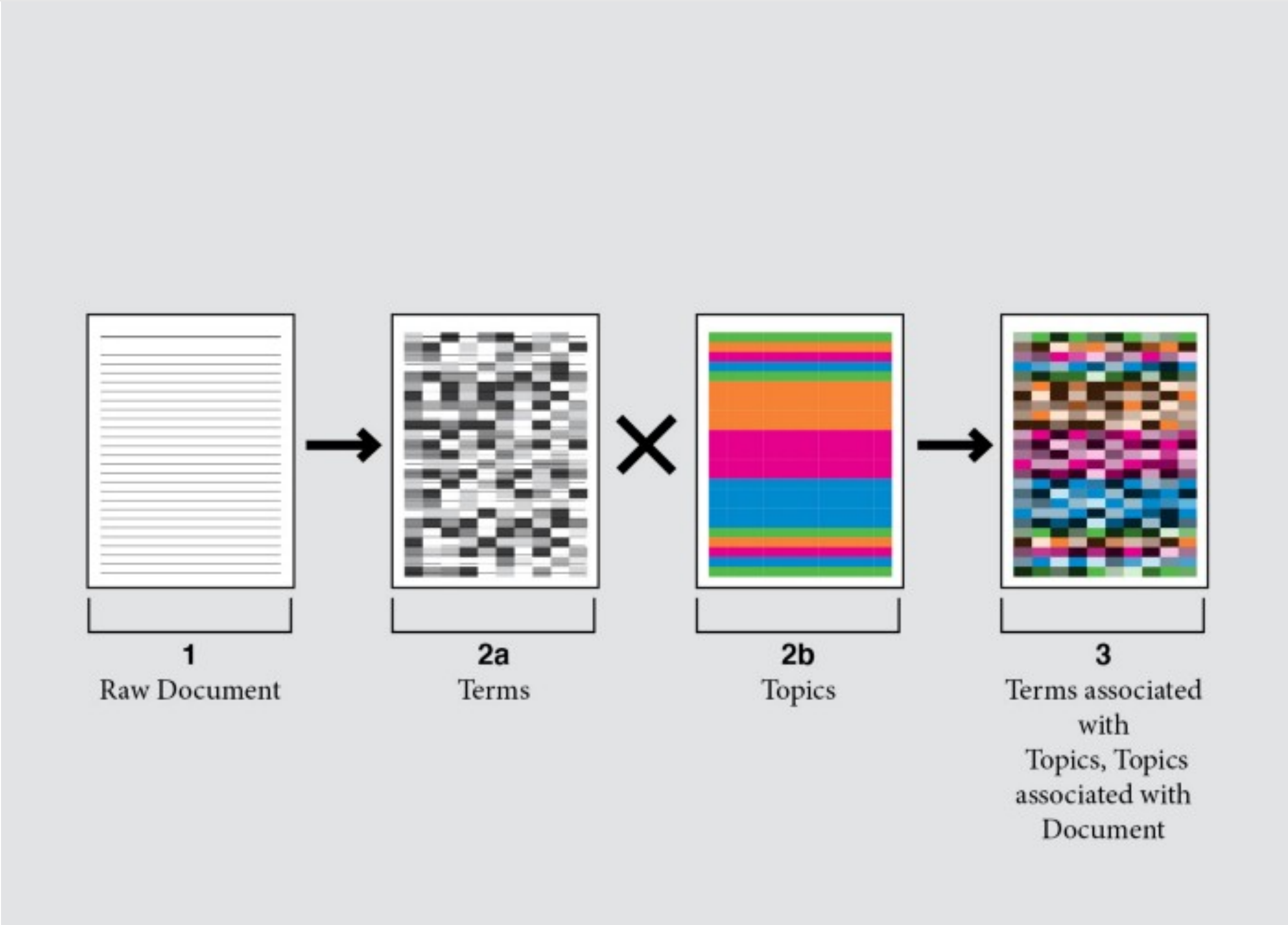
Terminology

We use **corpus** to refer to a group of similar media. For NMF, this could be a group of texts like articles, images like aerial photos or portraits, audio like songs, or youtube videos — anything that can be conceptualized as the addition of parts (rather than the addition and subtraction of parts). We use **document** to refer to one text, photo, song, or video. A **term** is a word.



Corpus, Document, Term — Image by Anupama Garla

NMF is a form of **Topic Modelling —** the art of extracting meaningful themes that recur through a corpus of documents. A corpus is composed of a set of topics embedded in its documents. A **document** is composed of a hierarchy of **topics.** A **topic** is composed of a hierarchy of **terms.**



Terms, Topics, Document — Image by Anupama Garla

NMF uses the logic of the 'the distributional hypothesis' discussed in detail in the Swedish linguist, Magnus Sahlgren's paper, The Distributional Hypothesis.

Sahlgren offers one often cited explanation,

> 'This hypothesis is often stated in terms like "words which are similar in meaning occur in similar contexts" (Rubenstein & Goodenough, 1965)'

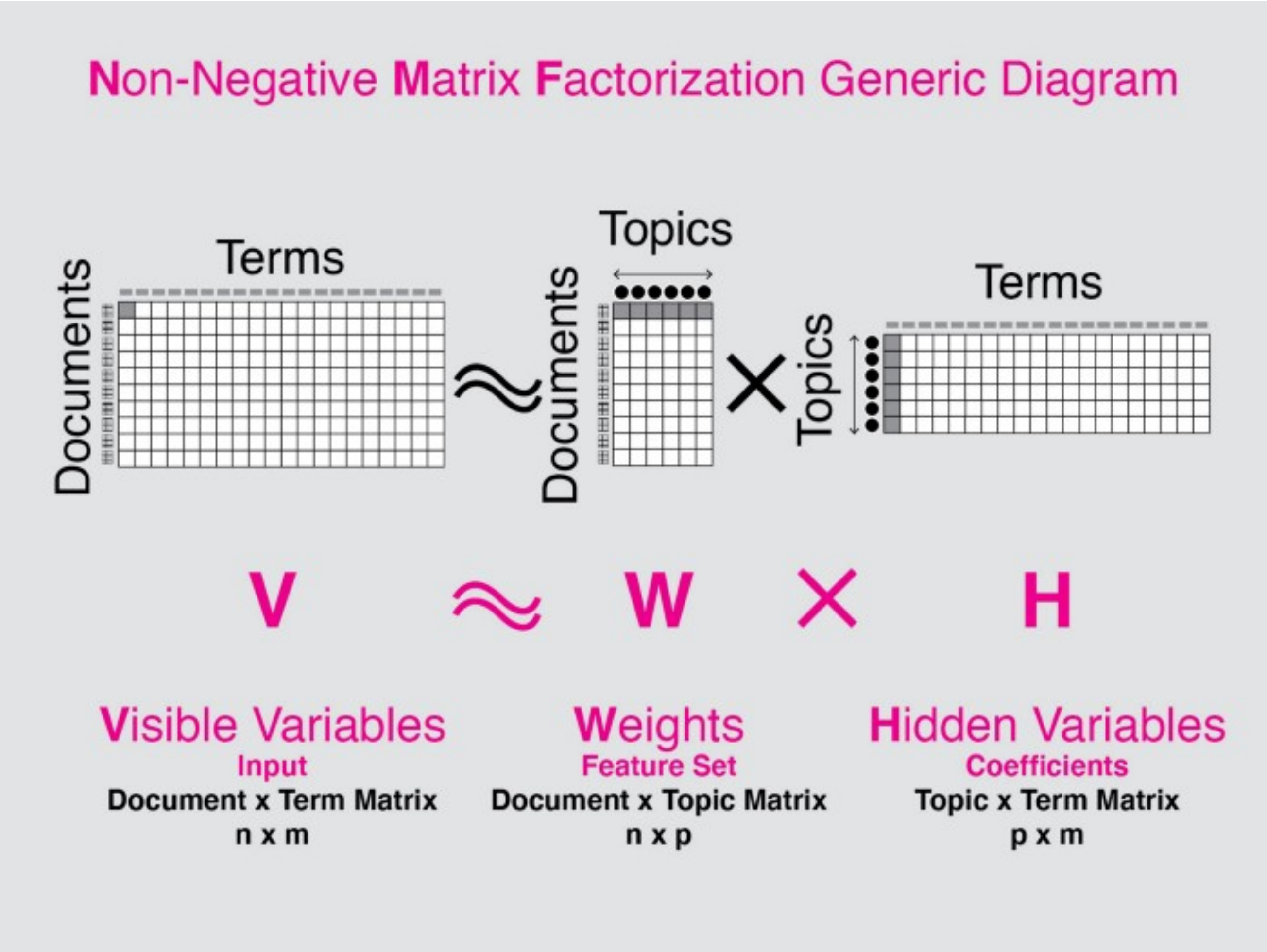
and continues — 'The general idea behind the distributional hypothesis seems clear enough: there is a correlation between distributional similarity and meaning similarity, which allows us to utilize the former in order to estimate the latter.'

For us, this means that documents can be treated as a bag of words. Similar documents have similar word frequency distributions — and similar topics. Two words that occur together are likely similar — belonging to a single topic. This algorithm ignores *syntactic*

Let's dig in!

NMF

NMF stands for Latent Semantic Analysis with the 'Non-negative Matrix-Factorization' method used to decompose the document-term matrix into two smaller matrices — the document-topic matrix (U) and the topic-term matrix (W) — each populated with unnormalized probabilities.



Matrix Decomposition in NMF Diagram by Anupama Garla

The values in V can be approximated through matrix multiplication of W and H. For instance the grey box in V can be found through matrix multiplication of the first row of W by the first column of H, also greyed in. The first row of V can be approximated through matrix multiplication of the first row of W with the entire matrix of H.

V is for Visible

In the *V matrix*, each row represents a *Document* that is composed of a *frequency of* excerpt of a *V Matrix* of Presidential Inauguration Speech Text:

Name	constitution	union	peace	freedom	principle	man	spirit	party	congress	justice	year	day	policy	president	service
Theodore Roosevelt	0.00	0.00	0.10	0.00	0.00	0.00	0.12	0.00	0.00	0.07	0.00	0.10	0.00	0.00	0.00
William Howard Taft	0.02	0.00	0.03	0.01	0.02	0.02	0.00	0.02	0.13	0.02	0.04	0.01	0.12	0.01	0.00
Woodrow Wilson	0.00	0.00	0.00	0.00	0.03	0.05	0.03	0.14	0.00	0.17	0.07	0.09	0.03	0.06	0.03
Warren G. Harding	0.00	0.02	0.09	0.07	0.00	0.03	0.05	0.01	0.02	0.06	0.00	0.00	0.05	0.00	0.08
Calvin Coolidge	0.07	0.00	0.15	0.09	0.08	0.05	0.00	0.17	0.03	0.09	0.04	0.01	0.10	0.02	0.03
Herbert Hoover	0.02	0.00	0.16	0.07	0.00	0.03	0.02	0.08	0.04	0.15	0.03	0.04	0.04	0.00	0.09
Franklin D. Roosevelt	0.02	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.10	0.00	0.02	0.08	0.06	0.00	0.00
Harry S. Truman	0.00	0.00	0.20	0.21	0.05	0.07	0.00	0.00	0.00	0.09	0.03	0.00	0.02	0.02	0.00
Dwight D. Eisenhower	0.02	0.00	0.19	0.18	0.10	0.11	0.02	0.00	0.00	0.00	0.00	0.05	0.02	0.00	0.03
John F. Kennedy	0.00	0.00	0.10	0.12	0.00	0.12	0.00	0.03	0.00	0.03	0.05	0.05	0.00	0.12	0.03
Lyndon Baines Johnson	0.00	0.24	0.00	0.05	0.00	0.25	0.02	0.00	0.00	0.13	0.06	0.08	0.00	0.00	0.00
Richard Milhous Nixon	0.02	0.00	0.22	0.02	0.00	0.21	0.12	0.00	0.00	0.02	0.11	0.04	0.00	0.07	0.00
Jimmy Carter	0.00	0.00	0.03	0.13	0.06	0.03	0.19	0.00	0.00	0.03	0.06	0.03	0.03	0.10	0.00
Ronald Reagan	0.02	0.00	0.03	0.15	0.02	0.11	0.00	0.02	0.02	0.02	0.05	0.10	0.00	0.10	0.00
George Bush	0.00	0.02	0.05	0.09	0.02	0.10	0.00	0.02	0.07	0.03	0.03	0.15	0.00	0.12	0.00
Bill Clinton	0.00	0.00	0.00	0.05	0.00	0.00	0.02	0.00	0.06	0.00	0.00	0.00	0.00	0.05	0.10
George W. Bush	0.00	0.02	0.02	0.08	0.06	0.00	0.06	0.00	0.00	0.06	0.04	0.04	0.00	0.07	0.06
Barack Obama	0.00	0.00	0.07	0.06	0.02	0.06	0.10	0.00	0.00	0.00	0.05	0.09	0.00	0.02	0.06
Donald J. Trump	0.00	0.00	0.00	0.03	0.00	0.00	0.03	0.05	0.00	0.02	0.05	0.09	0.00	0.14	0.00

Excerpt of V Matrix (Using TF-IDF Vectorizer) by Anupama Garla

W is for weights

In the *W matrix*, each row represents a *Document* that is composed of unnormalized probabilities of *Topics*. Each column represents a *semantic feature* that recurs throughout the corpus. For image processing, the *features* would be specific prototypical characteristics like 'mustache'. Here is an excerpt of a *W Matrix* of Presidential Inauguration Speech Text:

Name	Union + Constitution	Man + Freedom	Business + Policy	Principles + Improvement	Family + Jobs
George Washington	0.00	0.00	0.00	0.55	0.01
John Adams	0.03	0.00	0.02	0.53	0.03
Thomas Jefferson	0.05	0.19	0.00	0.37	0.00
James Madison	0.00	0.00	0.00	0.61	0.00
James Monroe	0.21	0.00	0.01	0.35	0.00
John Quincy Adams	0.31	0.10	0.00	0.18	0.00
Andrew Jackson	0.00	0.00	0.00	0.58	0.00
Martin Van Buren	0.15	0.01	0.02	0.38	0.02
William Henry Harrison	0.34	0.00	0.00	0.22	0.04
James Knox Polk	0.47	0.01	0.00	0.01	0.00
Zachary Taylor	0.15	0.00	0.02	0.32	0.01
Franklin Pierce	0.19	0.13	0.04	0.27	0.01
James Buchanan	0.47	0.00	0.00	0.00	0.00

Excerpt of W Matrix (1) by Anupama Garla

Abraham Lincoln	0.44	0.00	0.00	0.00	0.00
Ulysses S. Grant	0.14	0.00	0.22	0.00	0.00
Rutherford B. Hayes	0.21	0.00	0.24	0.09	0.00
James A. Garfield	0.35	0.04	0.08	0.00	0.03
Grover Cleveland	0.08	0.01	0.34	0.11	0.00
Benjamin Harrison	0.17	0.00	0.37	0.00	0.01
William McKinley	0.07	0.00	0.54	0.00	0.00
Theodore Roosevelt	0.00	0.06	0.15	0.07	0.20
William Howard Taft	0.05	0.00	0.52	0.00	0.00
Woodrow Wilson	0.00	0.00	0.25	0.00	0.30
Warren G. Harding	0.00	0.11	0.46	0.00	0.02
Calvin Coolidge	0.00	0.09	0.51	0.06	0.00
Herbert Hoover	0.00	0.06	0.56	0.00	0.02
Franklin D. Roosevelt	0.00	0.03	0.38	0.00	0.07

Excerpt of W Matrix (2) by Anupama Garla click to scroll output; double click to hide

Jimmy Carter	0.00	0.50	0.00	0.00	0.00
Ronald Reagan	0.00	0.39	0.00	0.00	0.19
George Bush	0.02	0.06	0.00	0.00	0.53
Bill Clinton	0.00	0.00	0.00	0.00	0.60
George W. Bush	0.00	0.04	0.00	0.00	0.48
Barack Obama	0.00	0.02	0.00	0.04	0.62
Donald J. Trump	0.00	0.00	0.00	0.00	0.54

Excerpt of W Matrix (3) by Anupama Garla

H is for hidden

In the *H matrix, each row represents a Topic* or *semantic feature* that is composed of *term frequencies. Each column represents a visible variable.* Two terms that occur frequently together form a topic, and each term gives more contextual meaning for the term it is grouped together with. If a term occurs frequently in two topics, then those topics are likely related. Here is an excerpt of an H *Matrix* of Presidential Inauguration Speech Text:

Topics	constitution	union	peace	freedom	principle	man	spirit	party	congress	justice	year	day	policy	president	service	administration
Union + Constitution	0.55	0.61	0.09	0.05	0.17	0.07	0.08	0.16	0.16	0.05	0.11	0.04	0.10	0.08	0.07	0.15
Man + Freedom	0.00	0.05	0.23	0.24	0.07	0.26	0.10	0.00	0.00	0.07	0.09	0.08	0.00	0.10	0.01	0.03
Business + Policy	0.05	0.00	0.15	0.05	0.04	0.02	0.04	0.15	0.15	0.13	0.08	0.04	0.16	0.02	0.08	0.08
Principles + Improvement	0.09	0.06	0.11	0.03	0.15	0.01	0.09	0.05	0.04	0.05	0.03	0.04	0.06	0.02	0.10	0.06
Family + Jobs	0.00	0.01	0.02	0.09	0.02	0.05	0.08	0.03	0.03	0.06	0.06	0.15	0.00	0.13	0.07	0.00

Excerpt of H Matrix by Anupama Garla

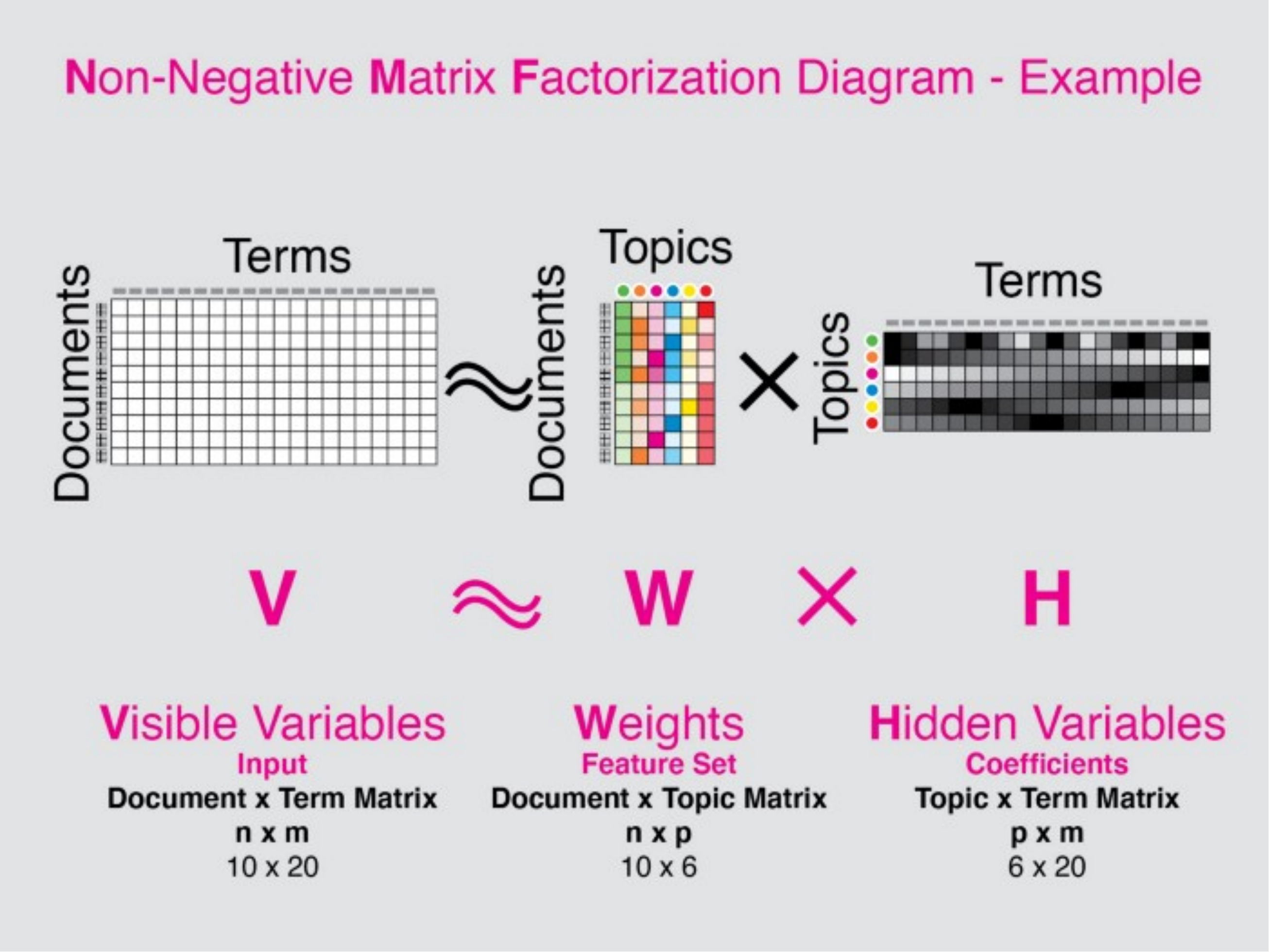
Topics	administration	liberty	hand	child	question	strength	generation	revenue	institution	purpose	condition	responsibility	faith	history	opinion
Union + Constitution	0.15	0.08	0.09	0.06	0.21	0.02	0.07	0.15	0.18	0.10	0.09	0.02	0.02	0.07	0.20
Man + Freedom	0.03	0.08	0.05	0.05	0.01	0.18	0.09	0.00	0.00	0.04	0.01	0.04	0.16	0.12	0.02
Business + Policy	0.08	0.02	0.03	0.00	0.07	0.01	0.00	0.12	0.04	0.07	0.12	0.10	0.04	0.02	0.01
Principles + Improvement	0.06	0.11	0.06	0.00	0.00	0.02	0.00	0.04	0.08	0.03	0.04	0.03	0.02	0.02	0.06
Family + Jobs	0.00	0.02	0.09	0.24	0.02	0.07	0.17	0.00	0.00	0.06	0.01	0.08	0.05	0.06	0.00

Excerpt of H Matrix (2) by Anupama Garla

The W and H matrix can be used to approximately reconstruct V through matrix multiplication.

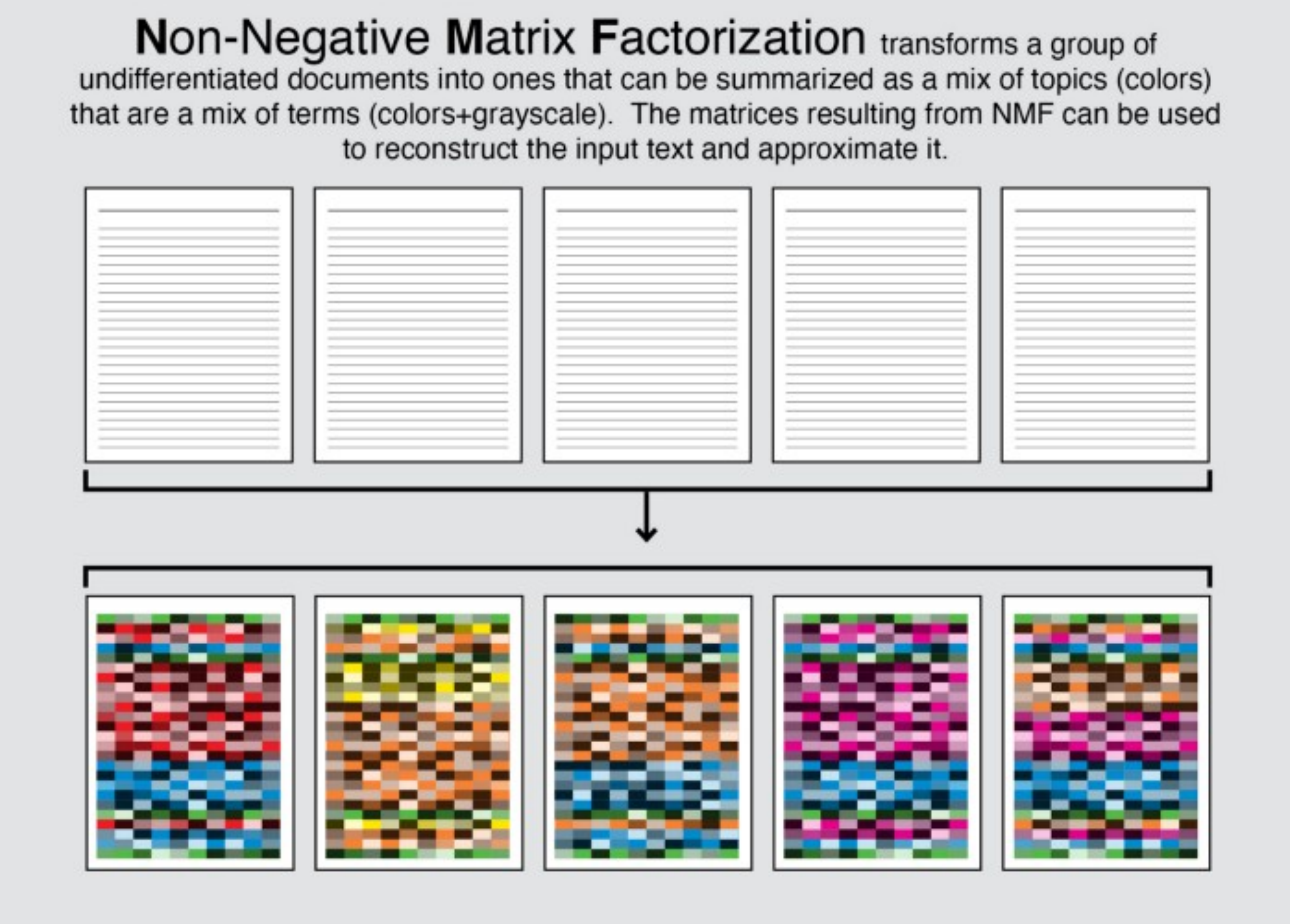
The Art of Topic Modeling

The output of NMF changes each time you run it, and the topics are not resolved — the data scientist must infer the topic from the highest word frequencies per topic, using the H matrix. This is where the art of choosing the correct number of topics comes into play. Too many, and you have topic repeats or topics composed of sparsely related words. Too few, and you have not very meaningful topics.



Matrix Decomposition in NMF Diagram Example by Anupama Garla

In the example above, the Topics (p) are set to 6. Each column of the W matrix represents a probability that the topic is in the document. Each row of the W matrix represents a distribution of topic frequencies in each Document. Each row of the H matrix represent the distribution of term frequencies in each topic, and can be seen as the degree to which each term is activated in each topic.



Conceptual Image of NMF Transformation of Corpus by Anupama Garla

NMF Math

Like most machine learning algorithms, NMF operates by starting with a guess of values for W and H, and iteratively minimizing the loss function. Typically it is implemented by updating one matrix (either W or H) for each iteration, and continuing to minimize the error function, $||V — WH|| = 0$, while W and H values remain non-negative, until W and H are stable. There are different loss functions used to update the matrices based on the specific programming package you use, but the one proposed in the original Lee and Seung paper is the *multiplicative update rule*. Honestly the math is fairly complex, and it would take me some study to wrap my head around it and explain it in a no-fuss style. I'm going to call it out-of-scope of this article. :-).

NMF caveats

NMF requires that all documents are fairly similar — for instance if you are comparing faces, you would look at faces from a similar angle. If you are comparing texts, you would look at texts of similar lengths. For more complex documents/images, you would

NMF and TF-IDF

The advantage of NMF, as opposed to TF-IDF is that NMF breaks down the V matrix into ***two smaller matrices***, W and H. The data scientist can set the number of Topics (p) to determine how small these matrices get. Data scientists often use the TF-IDF derived Document-Term Matrix as the Input Matrix, V, because it yields better results.

NMF vs. other matrix decomposition methods

NMF differs from other matrix decomposition methods like PCA and VQ in that it only uses non-negative numbers. This allows for each *Topic* or *feature* to be ***interpretable***. Additionally, W and H can be represented by sparse matrices where only the values > 0 are encoded, making for ***a smaller dataset***.

Python Implementation

This is generally the order of operations:

1. Import Relevant Libraries to Jupyter Notebook / Install Libraries in your Environment
2. Select Data and Import
3. Isolate Data to Topic Model
4. Clean Data
5. Create Function to Pre-process Data*
6. Create Document Term Matrix 'V'
7. Create Function to Display Topics*
8. Run NMF on Document Term Matrix 'V'
9. Iterate until you find useful Topics

*Star indicates optional steps

1. Import the Python Libraries you will need.

For **dataframe manipulation** and exporting you will need:

```
import pandas as pd
```

From scikit learn for **modeling** you will need:

- TfidfVectorizer
- NMF
- text

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF
from sklearn.feature_extraction import text
```

From nltk for **text processing** you may need:

- stopwords
- word_tokenize
- pos_tag

```
from nltk.corpus import stopwords
from nltk import word_tokenize, pos_tag
from nltk.stem import WordNetLemmatizer
```

For **text cleaning** you may need:

- re (regular expressions)
- string

```
import re
import string
```

Next, select your corpus of documents. I am using American President's Inauguration Speeches, available as a Kaggle Dataset. I did get an error when I tried to import the speeches without the engine clause but I just googled my error and took this 'engine' suggestion which worked for me.

```
# expand pandas df column display width to enable easy inspection
pd.set_option('max_colwidth', 150)

# read in csv to dataframe
df = pd.read_csv('inaug_speeches.csv', engine='python')

# visually inspect dataframe
df.head()
```

	Unnamed: 0	Name	Inaugural Address	Date	text
0	4	George Washington	First Inaugural Address	Thursday, April 30, 1789	Fellow-Citizens of the Senate and of the House of Representatives: ��AMONG the vicissitudes incident to life no event could have fille...
1	5	George Washington	Second Inaugural Address	Monday, March 4, 1793	Fellow Citizens: ��I AM again called upon by the voice of my country to execute the functions of its Chief Magistrate. When the occas...
2	6	John Adams	Inaugural Address	Saturday, March 4, 1797	��WHEN it was first perceived, in early times, that no middle course for America remained between unlimited submission to a foreign le...
3	7	Thomas Jefferson	First Inaugural Address	Wednesday, March 4, 1801	Friends and Fellow-Citizens: ��CALLED upon to undertake the duties of the first executive office of our country, I avail myself of th...
4	8	Thomas Jefferson	Second Inaugural Address	Monday, March 4, 1805	��PROCEEDING, fellow-citizens, to that qualification which the Constitution requires before my entrance on the charge again conferred ...

Raw DataFrame Image by Anupama Garla

3. Isolate Data to Topic Model

I'm going to make a dataframe of the President's names and speeches, and isolate to the first term inauguration speech, because some President's did not have a second term. This makes for a corpus of documents with similar lengths.

```
# Select Rows that are first term inaugural addresses
df = df.drop_duplicates(subset=['Name'], keep='first')

# Clean Up Index
df = df.reset_index()

# Select only President's Names and their Speeches
df = df[['Name', 'text']]

# Set Index to President's Names
df = df.set_index('Name')
```

Name	text
George Washington	Fellow-Citizens of the Senate and of the House of Representatives: ��AMONG the vicissitudes incident to life no event could have fille...
Thomas Jefferson	Friends and Fellow-Citizens: ��CALLED upon to undertake the duties of the first executive office of our country, I avail myself of th...
James Madison	��UNWILLING to depart from examples of the most revered authority, I avail myself of the occasion now presented to express the profoun...
James Monroe	��I SHOULD be destitute of feeling if I was not deeply affected by the strong proof which my fellow-citizens have given me of their co...
Andrew Jackson	Fellow-Citizens: ��ABOUT to undertake the arduous duties that I have been appointed to perform by the choice of a free people, I avai...

Isolated Text DataFrame Image by Anupama Garla

4. Clean Data

I want to make all of the text in the speeches as comparable as possible so I will create a cleaning function that removes punctuation, capitalization, numbers, and strange characters. I use regular expressions for this, which offers a lot of ways to 'substitute' text. There are tons of regular expression cheat sheets, like this one. I 'apply' this function to my speech column.

```
def clean_text_round1(text):
    '''Make text lowercase, remove text in square brackets,
    remove punctuation, remove read errors,
    and remove words containing numbers.'''

    text = text.lower()
    text = re.sub('\[.*?\]', ' ', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), ' ', text)
    text = re.sub('\w*\d\w*', ' ', text)
    text = re.sub('�', ' ', text)

    return text

round1 = lambda x: clean_text_round1(x)

# Clean Speech Text
df["text"] = df["text"].apply(round1)

# Visually Inspect
df.head()
```

Cleaned Text DataFrame Image by Anupama Garla

5. Create Function to Pre-process Data*

Here we will ***lemmatize*** all the words in the speeches so that different forms of a particular word will be reduced to a base form, for instance noun plurals become singular and all verb tenses become present. This simplifies the text so that repeated instances of slight variations of a word are interpreted as one word. One can stem or lemmatize, more info here. The other thing we are doing here is isolating the speech text to a particular part of speech — nouns. You can experiment with different parts of speech isolation but I have found most success with nouns in this particular dataset. More info on how to select parts of speech with ***'pos' (part of speech) tagging*** here. This is a step you can skip if you want to setup a basic pipeline first and then you can add it in when you iterate to find what works best with your dataset.

```
# Noun extract and lemmatize function

def nouns(text):
    '''Given a string of text, tokenize the text
    and pull out only the nouns.'''

    # create mask to isolate words that are nouns
    is_noun = lambda pos: pos[:2] == 'NN'

    # store function to split string of words
    # into a list of words (tokens)
    tokenized = word_tokenize(text)

    # store function to lemmatize each word
    wordnet_lemmatizer = WordNetLemmatizer()

    # use list comprehension to lemmatize all words
    # and create a list of all nouns
    all_nouns = [wordnet_lemmatizer.lemmatize(word) \
    for (word, pos) in pos_tag(tokenized) if is_noun(pos)]

    #return string of joined list of nouns
```

```
data_nouns = pd.DataFrame(df.text.apply(nouns))

# Visually Inspect
data_nouns.head()
```


Nouns DataFrame Image by Anupama Garla

6. Create Document Term Matrix 'V'

Here I added some stopwords to the stopword list so we don't get words like 'America' in the topics as that is not super meaningful in this context. I also use TF-IDF Vectorizer rather than a simple Count Vectorizer in order to give greater value to more unique terms. You can learn more about TF-IDF here.

```
# Add additional stop words since we are recreating the document-term
matrix
stop_noun = ["america", 'today', 'thing']
stop_words_noun_agg = text.ENGLISH_STOP_WORDS.union(stop_noun)

# Create a document-term matrix with only nouns

# Store TF-IDF Vectorizer
tv_noun = TfidfVectorizer(stop_words=stop_words_noun_agg, ngram_range
= (1,1), max_df = .8, min_df = .01)

# Fit and Transform speech noun text to a TF-IDF Doc-Term Matrix
data_tv_noun = tv_noun.fit_transform(data_nouns.text)

# Create data-frame of Doc-Term Matrix with nouns as column names
data_dtm_noun = pd.DataFrame(data_tv_noun.toarray(),
columns=tv_noun.get_feature_names())

# Set President's Names as Index
data_dtm_noun.index = df.index

# Visually inspect Document Term Matrix
data_dtm_noun.head()
```

TF-IDF Document Term Matrix DataFrame Image by Anupama Garla

7. Create Function to Display Topics*

To evaluate how useful the topics created by NMF are, we need to know what they are. Here I create a function to display the top words activated for each topic.

```
def display_topics(model, feature_names, num_top_words,\
topic_names=None):
'''Given an NMF model, feature_names, and number of top words, print
topic number and its top feature names, up to specified number of top
words.'''

    # iterate through topics in topic-term matrix, 'H' aka
    # model.components_
    for ix, topic in enumerate(model.components_):

        #print topic, topic number, and top words
        if not topic_names or not topic_names[ix]:
            print("\nTopic ", ix)
        else:
            print("\nTopic: '",topic_names[ix],"'")
        print(", ".join([feature_names[i] \
             for i in topic.argsort()[:-num_top_words - 1:-1]]))
```

8. Run NMF on Document Term Matrix 'V'

Perhaps the easiest step is running NMF on the document term matrix once you have one.

```
nmf_model = NMF(2)

# Learn an NMF model for given Document Term Matrix 'V'
# Extract the document-topic matrix 'W'
```

```
display_topics(nmf_model, tv_noun.get_feature_names(), 5)
```


2 Topics displayed with strongest terms — Image by Anupama Garla

9. Iterate until you find useful Topics

This is where the art comes in. As a basic dummy model, I usually run the text through a count vectorizer and then NMF to get an idea of what we are looking[...] through topic numbers, parts of speech, TF-IDF settings, adding stop[...] much more. Getting useful topics can be done by eye, and is definite[...]

```
nmf_model = NMF(8)
doc_topic = nmf_model.fit_transform(data_dtm_noun)

display_topics(nmf_model, tv_noun.get_feature_names(), 5)
```


8 Topics displayed with strongest terms — Image by Anupama Garla

Conclusions

Further Reading

On TF-IDF

TF-IDF : A visual explainer and Python implementation on Presidential Inauguration Speeches

Ever asked to explain TF-IDF to non-technical audiences? Here is a visual unpacking of TF-IDF (Term Frequency — Inverse…

towardsdatascience.com

On Creating a Recommendation System from NMF generated Topics

Building a Content-Based Children's Book Recommender for

- LDA : Latent Dirichlet Algorithm — probability based and decomposes the corpus into two matrices. This is a generative model where you can use these smaller

However, here we can see 8 inauguration themes which could be described as — laws, peac leadership, spending, justice, revenue, ideals, decision-making. This is just the beginning of project that would investigate inauguration speeches. Some questions we can ask of these results are:

- Which President focussed on which topics?
- Which President's speeches are most similar?
- Is there a clear divide between recent Democratic and Republican Presidents in terms of topic focus? Is there a clear progression of topics through time?

The opportunities are many and the process is iterative, depending on what questions you se to answer. The surprising results of this analysis so far is that recent Presidents, regardless o political party, discuss the same general subjects in relation to all Presidents in the past. Tha both surprising and expected as we are all living through similar challenges at this moment. wonder what topic modeling on adjectives would yield…

- SVD: Latent Semantic Analysis with Singular Value Decomposition — a way of decomposing the document-term matrix into three smaller matrices that can be reformatted into two matrices containing both ***negative and positive*** unnormalized probabilities which prevents a direct interpretation of the decomposed matrices.
- NMF : Latent Semantic Analysis with Non-Negative Matrix Factorization — a way of decomposing the document-term matrix into two smaller matrices that contain only positive values which allows ***direct interpretation*** of each matrix as unnormalized probabilities.

NMF Original Paper

Even though there are tons of resources online for learning NMF, I think it's always best to go to the source for a deeper understanding, so below is a link to the original Lee and Seung paper on NMF:

Lee, D., Seung, H. Learning the parts of objects by non-negative matrix factorization. *Nature* 401, 788–791 (1999). https://doi.org/10.1038/44565

NMF scikit learn Documentation

It's also best to get acquainted with the toggles on your NMF algorithm in scikit learn. Dig in here once you start iterating.

Topic Supervised NMF

This method is a supervised spin on NMF and allows more user control over the topics. I haven't dug into it yet, but would love to know if anyone has!

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

Get this newsletter

Natural Language Process   Python   Machine Learning   Hands On Tutorials   Deep Dives