TF-IDF with Scikit-Learn

Contents

Print to PDF

In the previous lesson, we learned about a text analysis method called *term frequency–inverse document frequency*, often abbreviated *tf-idf*. Tf-idf is a method that tries to identify the most distinctively frequent or significant words in a document. We specifically learned how to calculate tf-idf scores using word frequencies per page—or "extracted features"—made available by the HathiTrust Digital Library.

In this lesson, we're going to learn how to calculate tf-idf scores using a collection of plain text (.txt) files and the Python library scikit-learn, which has a quick and nifty module called TfidfVectorizer.

In this lesson, we will cover how to:

- Calculate and normalize tf-idf scores for U.S. Inaugural Addresses with scikit-learn

Dataset

U.S. Inaugural Addresses

> This is the meaning of our liberty and our creed; why men and women and children of every race and every faith can join in celebration across this magnificent Mall, and why a man whose father less than 60 years ago might not have been served at a local restaurant can now stand before you to take a most sacred oath. So let us mark this day with remembrance of who we are and how far we have traveled.
>
> —Barack Obama, Inaugural Presidential Address, January 2009

During Barack Obama's Inaugural Address in January 2009, he mentioned "women" four different times, including in the passage quoted above. How distinctive is Obama's inclusion of women in this address compared to all other U.S. Presidents? This is one of the questions that we're going to try to answer with tf-idf.

Breaking Down the TF-IDF Formula

But first, let's quickly discuss the tf-idf formula. The idea is pretty simple.

tf-idf = term_frequency * inverse_document_frequency

term_frequency = number of times a given term appears in document

inverse_document_frequency = log(total number of documents / number of documents with term) + 1*****

You take the number of times a term occurs in a document (term frequency). Then you take the number of documents in which the same term occurs at least once divided by the total number of documents (document frequency), and you flip that fraction on its head (inverse document frequency). Then you multiply the two numbers together (term_frequency * inverse_document_frequency).

The reason we take the *inverse*, or flipped fraction, of document frequency is to boost the rarer words that occur in relatively few documents. Think about the inverse document frequency for the word "said" vs the word "pigeon." The term "said" appears in 13 (document frequency) of 14 (total documents) *Lost in the City* stories (14 / 13 –> a smaller inverse document frequency) while the term "pigeons" only occurs in 2 (document frequency) of the 14 stories (total documents) (14 / 2 –> a bigger inverse document frequency, a bigger tf-idf boost).

*There are a bunch of slightly different ways that you can calculate inverse document frequency. The version of idf that we're going to use is the scikit-learn default, which uses "smoothing" aka it adds a "1" to the numerator and denominator:

inverse_document_frequency = log((1 + total_number_of_documents) / (number_of_documents_with_term +1)) + 1

If smooth_idf=True (the default), the constant "1" is added to the numerator and denominator of the idf as if an extra document was seen containing every term in the collection exactly once, which prevents zero divisions: idf(t) = log [(1 + n) / (1 + df(t))] + 1.
-scikit-learn documentation

TF-IDF with scikit-learn

scikit-learn, imported as `sklearn`, is a popular Python library for machine learning approaches such as clustering, classification, and regression. Though we're not doing any machine learning in this lesson, we're nevertheless going to use scikit-learn's `TfidfVectorizer` and `CountVectorizer`.

Install scikit-learn

```
!pip install sklearn
```

Import necessary modules and libraries

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
pd.set_option("max_rows", 600)
from pathlib import Path
import glob
```

🐼 Pandas

Do you need a refresher or introduction to the Python data analysis library Pandas? Be sure to check out Pandas Basics (1-3) in this textbook!

We're also going to import `pandas` and change its default display setting. And we're going to import two libraries that will help us work with files and the file system: `pathlib` and `glob`.

Set Directory Path

Below we're setting the directory filepath that contains all the text files that we want to analyze.

```
directory_path = "../texts/history/US_Inaugural_Addresses/"
```

Then we're going to use `glob` and `Path` to make a list of all the filepaths in that directory and a list of all the short story titles.

```
text_files = glob.glob(f"{directory_path}/*.txt")
```

```
text_files
```

```
 '../texts/history/US_Inaugural_Addresses/19_lincoln_1861.txt',
 '../texts/history/US_Inaugural_Addresses/01_washington_1789.txt',
 '../texts/history/US_Inaugural_Addresses/29_mckinley_1901.txt',
 '../texts/history/US_Inaugural_Addresses/04_jefferson_1801.txt',
 '../texts/history/US_Inaugural_Addresses/34_harding_1921.txt',
 '../texts/history/US_Inaugural_Addresses/52_clinton_1993.txt',
 '../texts/history/US_Inaugural_Addresses/35_coolidge_1925.txt',
 '../texts/history/US_Inaugural_Addresses/39_roosevelt_franklin_1941.txt',
 '../texts/history/US_Inaugural_Addresses/28_mckinley_1897.txt',
 '../texts/history/US_Inaugural_Addresses/24_garfield_1881.txt',
 '../texts/history/US_Inaugural_Addresses/22_grant_1873.txt',
 '../texts/history/US_Inaugural_Addresses/15_polk_1845.txt',
 '../texts/history/US_Inaugural_Addresses/54_bush_george_w_2001.txt',
 '../texts/history/US_Inaugural_Addresses/02_washington_1793.txt',
 '../texts/history/US_Inaugural_Addresses/38_roosevelt_franklin_1937.txt',
 '../texts/history/US_Inaugural_Addresses/37_roosevelt_franklin_1933.txt',
 '../texts/history/US_Inaugural_Addresses/18_buchanan_1857.txt',
 '../texts/history/US_Inaugural_Addresses/16_taylor_1849.txt',
 '../texts/history/US_Inaugural_Addresses/05_jefferson_1805.txt',
 '../texts/history/US_Inaugural_Addresses/26_harrison_1889.txt',
 '../texts/history/US_Inaugural_Addresses/44_kennedy_1961.txt',
 '../texts/history/US_Inaugural_Addresses/23_hayes_1877.txt',
```

```
text_titles = [Path(text).stem for text in text_files]
```

```
text_titles
```

```
['13_van_buren_1837',
 '47_nixon_1973',
 '50_reagan_1985',
 '53_clinton_1997',
 '17_pierce_1853',
 '14_harrison_1841',
 '56_obama_2009',
 '25_cleveland_1885',
 '03_adams_john_1797',
 '12_jackson_1833',
 '11_jackson_1829',
 '36_hoover_1929',
 '45_johnson_1965',
 '51_bush_george_h_w_1989',
 '21_grant_1869',
 '41_truman_1949',
 '33_wilson_1917',
 '49_reagan_1981',
 '30_roosevelt_theodore_1905',
 '07_madison_1813',
```

Calculate tf–idf

To calculate tf–idf scores for every word, we're going to use scikit-learn's `TfidfVectorizer`.

When you initialize TfidfVectorizer, you can choose to set it with different parameters. These parameters will change the way you calculate tf–idf.

The recommended way to run `TfidfVectorizer` is with smoothing (`smooth_idf = True`) and normalization (`norm='l2'`) turned on. These parameters will better account for differences in text length, and overall produce more meaningful tf–idf scores. Smoothing and L2 normalization are actually the default settings for `TfidfVectorizer`, so to turn them on, you don't need to include any extra code at all.

Initialize TfidfVectorizer with desired parameters (default smoothing and normalization)

```
tfidf_vectorizer = TfidfVectorizer(input='filename', stop_words='english')
```

Run TfidfVectorizer on our `text_files`

```
tfidf_vector = tfidf_vectorizer.fit_transform(text_files)
```

Make a DataFrame out of the resulting tf–idf vector, setting the "feature names" or words as columns and the titles as rows

```
tfidf_df = pd.DataFrame(tfidf_vector.toarray(), index=text_titles,
columns=tfidf_vectorizer.get_feature_names())
```

Add column for document frequency aka number of times word appears in all documents

```
tfidf_df.loc['00_Document Frequency'] = (tfidf_df > 0).sum()
```

```
tfidf_slice = tfidf_df[['government', 'borders', 'people', 'obama', 'war',
'honor','foreign', 'men', 'women', 'children']]
tfidf_slice.sort_index().round(decimals=2)
```

	government	borders	people	obama	war	honor	fore
00_Document Frequency	53.00	5.00	56.00	3.00	45.00	32.00	32
01_washington_1789	0.11	0.00	0.05	0.00	0.00	0.00	0
02_washington_1793	0.06	0.00	0.05	0.00	0.00	0.08	0
03_adams_john_1797	0.16	0.00	0.19	0.00	0.01	0.10	0
04_jefferson_1801	0.16	0.00	0.01	0.00	0.01	0.04	0
05_jefferson_1805	0.03	0.00	0.00	0.00	0.04	0.00	0
06_madison_1809	0.00	0.00	0.02	0.00	0.02	0.05	0
07_madison_1813	0.04	0.00	0.04	0.00	0.25	0.02	0
08_monroe_1817	0.17	0.00	0.11	0.00	0.09	0.01	0
09_monroe_1821	0.08	0.00	0.06	0.00	0.11	0.02	0
10_adams_john_quincy_1825	0.15	0.00	0.06	0.00	0.05	0.01	0
11_jackson_1829	0.10	0.00	0.06	0.00	0.02	0.02	0
12_jackson_1833	0.21	0.00	0.14	0.00	0.00	0.00	0
13_van_buren_1837	0.12	0.00	0.14	0.00	0.02	0.02	0
14_harrison_1841	0.14	0.00	0.14	0.00	0.01	0.02	0
15_polk_1845	0.26	0.00	0.08	0.00	0.03	0.01	0
16_taylor_1849	0.12	0.00	0.05	0.00	0.00	0.02	0
17_pierce_1853	0.08	0.00	0.05	0.00	0.00	0.02	0
18_buchanan_1857	0.12	0.00	0.11	0.00	0.08	0.01	0
19_lincoln_1861	0.12	0.00	0.13	0.00	0.02	0.00	0
20_lincoln_1865	0.02	0.00	0.00	0.00	0.27	0.00	0
21_grant_1869	0.05	0.00	0.03	0.00	0.02	0.05	0
22_grant_1873	0.06	0.00	0.10	0.00	0.05	0.02	0
23_hayes_1877	0.17	0.00	0.08	0.00	0.00	0.00	0
24_garfield_1881	0.19	0.00	0.16	0.00	0.05	0.00	0
25_cleveland_1885	0.21	0.00	0.21	0.00	0.00	0.00	0
26_harrison_1889	0.06	0.00	0.17	0.00	0.02	0.03	0
27_cleveland_1893	0.15	0.00	0.22	0.00	0.00	0.00	0
28_mckinley_1897	0.16	0.00	0.16	0.00	0.05	0.03	0
29_mckinley_1901	0.15	0.00	0.12	0.00	0.08	0.04	0
30_roosevelt_theodore_1905	0.05	0.00	0.10	0.00	0.00	0.00	0
31_taft_1909	0.12	0.00	0.03	0.00	0.03	0.01	0
32_wilson_1913	0.11	0.00	0.02	0.00	0.00	0.00	0
33_wilson_1917	0.00	0.00	0.08	0.00	0.07	0.00	0
34_harding_1921	0.08	0.00	0.05	0.00	0.12	0.00	0
35_coolidge_1925	0.10	0.00	0.10	0.00	0.02	0.01	0
36_hoover_1929	0.20	0.04	0.10	0.00	0.01	0.00	0
37_roosevelt_franklin_1933	0.03	0.00	0.08	0.00	0.02	0.02	0
38_roosevelt_franklin_1937	0.18	0.03	0.12	0.00	0.01	0.00	0
39_roosevelt_franklin_1941	0.05	0.00	0.08	0.00	0.00	0.00	0
40_roosevelt_franklin_1945	0.00	0.00	0.02	0.00	0.05	0.03	0
41_truman_1949	0.03	0.00	0.10	0.00	0.02	0.01	0
42_eisenhower_1953	0.01	0.00	0.10	0.00	0.04	0.03	0
43_eisenhower_1957	0.00	0.00	0.10	0.00	0.01	0.05	0
44_kennedy_1961	0.00	0.00	0.01	0.00	0.06	0.00	0
45_johnson_1965	0.01	0.00	0.11	0.00	0.01	0.00	0

	government	borders	people	obama	war	honor	fore
46_nixon_1969	0.05	0.00	0.13	0.00	0.03	0.03	C
47_nixon_1973	0.10	0.00	0.06	0.00	0.03	0.01	C
48_carter_1977	0.06	0.00	0.08	0.00	0.02	0.00	C
49_reagan_1981	0.16	0.00	0.08	0.00	0.01	0.00	C
50_reagan_1985	0.16	0.00	0.14	0.00	0.01	0.01	C
51_bush_george_h_w_1989	0.05	0.00	0.06	0.00	0.03	0.00	(
52_clinton_1993	0.05	0.00	0.13	0.00	0.03	0.00	C
53_clinton_1997	0.09	0.00	0.09	0.00	0.01	0.00	C
54_bush_george_w_2001	0.05	0.00	0.01	0.00	0.01	0.00	C
55_bush_george_w_2005	0.03	0.06	0.05	0.00	0.00	0.04	C
56_obama_2009	0.03	0.03	0.07	0.03	0.02	0.01	C
57_obama_2013	0.04	0.00	0.11	0.04	0.04	0.00	C
58_trump_2017	0.04	0.11	0.11	0.12	0.00	0.00	C

Let's drop "OO_Document Frequency" since we were just using it for illustration purposes.

```
tfidf_df = tfidf_df.drop('00_Document Frequency', errors='ignore')
```

Let's reorganize the DataFrame so that the words are in rows rather than columns.

```
tfidf_df.stack().reset_index()
```

	level_0	level_1	0
0	13_van_buren_1837	000	0.000000
1	13_van_buren_1837	03	0.011681
2	13_van_buren_1837	04	0.011924
3	13_van_buren_1837	05	0.000000
4	13_van_buren_1837	100	0.000000
...	...	...	...
521937	31_taft_1909	zachary	0.000000
521938	31_taft_1909	zeal	0.000000
521939	31_taft_1909	zealous	0.000000
521940	31_taft_1909	zealously	0.000000
521941	31_taft_1909	zone	0.000000

521942 rows × 3 columns

```
tfidf_df = tfidf_df.stack().reset_index()
```

```
tfidf_df = tfidf_df.rename(columns={0:'tfidf', 'level_0': 'document','level_1':
'term', 'level_2': 'term'})
```

To find out the top 10 words with the highest tf–idf for every story, we're going to sort by document and tfidf score and then groupby document and take the first 10 values.

```
tfidf_df.sort_values(by=['document','tfidf'], ascending=
[True,False]).groupby(['document']).head(10)
```

	document	term	tfidf
219683	01_washington_1789	government	0.113681
220084	01_washington_1789	immutable	0.103883
220151	01_washington_1789	impressions	0.103883
222313	01_washington_1789	providential	0.103883
221607	01_washington_1789	ought	0.103728
222327	01_washington_1789	public	0.103102
222093	01_washington_1789	present	0.097516
222365	01_washington_1789	qualifications	0.096372
221787	01_washington_1789	peculiarly	0.090546
216629	01_washington_1789	article	0.085786

```
top_tfidf = tfidf_df.sort_values(by=['document','tfidf'], ascending=
[True,False]).groupby(['document']).head(10)
```

We can zoom in on particular words and particular documents.

```
top_tfidf[top_tfidf['term'].str.contains('women')]
```

	document	term	tfidf
62910	56_obama_2009	women	0.084859

It turns out that the term “women” is very distinctive in Obama’s Inaugural Address.

```
top_tfidf[top_tfidf['document'].str.contains('obama')]
```

	document	term	tfidf
54455	56_obama_2009	america	0.148351
59347	56_obama_2009	nation	0.120229
59407	56_obama_2009	new	0.118002
62142	56_obama_2009	today	0.114792
57639	56_obama_2009	generation	0.100654
58811	56_obama_2009	let	0.091100
58627	56_obama_2009	jobs	0.090727
55960	56_obama_2009	crisis	0.087235
57828	56_obama_2009	hard	0.084859
62910	56_obama_2009	women	0.084859
418595	57_obama_2013	journey	0.167591
415909	57_obama_2013	creed	0.139659
417599	57_obama_2013	generation	0.127260
414415	57_obama_2013	america	0.125044
415519	57_obama_2013	complete	0.114891
420751	57_obama_2013	requires	0.114891
419777	57_obama_2013	people	0.110351
422088	57_obama_2013	time	0.105563
422102	57_obama_2013	today	0.103668
416980	57_obama_2013	evident	0.100896

```
top_tfidf[top_tfidf['document'].str.contains('trump')]
```

	document	term	tfidf
504405	58_trump_2017	america	0.350162
506586	58_trump_2017	dreams	0.156436
504406	58_trump_2017	american	0.149226
508577	58_trump_2017	jobs	0.142766
510263	58_trump_2017	protected	0.132439
509410	58_trump_2017	obama	0.120288
509767	58_trump_2017	people	0.112370
512002	58_trump_2017	thank	0.109171
504990	58_trump_2017	borders	0.107075
512597	58_trump_2017	ve	0.107075

```
top_tfidf[top_tfidf['document'].str.contains('kennedy')]
```

	document	term	tfidf
391774	44_kennedy_1961	let	0.267869
394306	44_kennedy_1961	sides	0.262849
392921	44_kennedy_1961	pledge	0.160960
387632	44_kennedy_1961	ask	0.107713
387864	44_kennedy_1961	begin	0.106495
388991	44_kennedy_1961	dare	0.106495
395895	44_kennedy_1961	world	0.103110
390313	44_kennedy_1961	final	0.102311
392370	44_kennedy_1961	new	0.096600
390120	44_kennedy_1961	explore	0.094223

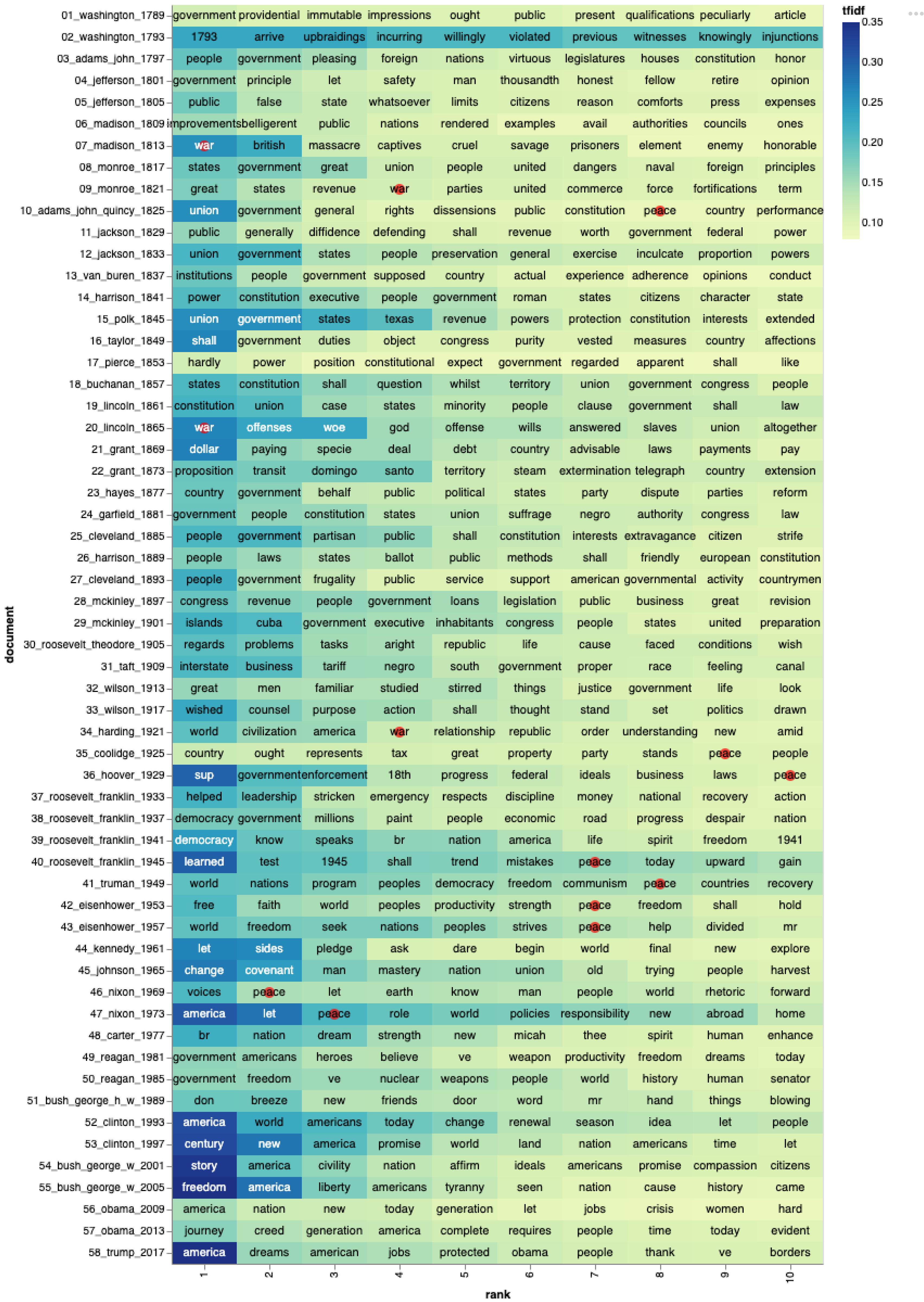
Visualize TF-IDF

We can also visualize our TF-IDF results with the data visualization library Altair.

```
!pip install altair
```

Let's make a heatmap that shows the highest TF-IDF scoring words for each president, and let's put a red dot next to two terms of interest: "war" and "peace":

The code below was contributed by [Eric Monson](). Thanks, Eric!



Your Turn!

Take a few minutes to explore the dataframe below and then answer the following questions.

1. What is the difference between a tf-idf score and raw word frequency?

2. Based on the dataframe above, what is one potential problem or limitation that you notice with tf-idf scores?

3. What's another collection of texts that you think might be interesting to analyze with tf-idf scores? Why?
