# 03_document_term_matrix

September 29, 2021

# 1 From tokens to numbers: the document-term matrix

The bag of words model represents a document based on the frequency of the terms or tokens it contains. Each document becomes a vector with one entry for each token in the vocabulary that reflects the token's relevance to the document.

The document-term matrix is straightforward to compute given the vocabulary. However, it is also a crude simplification because it abstracts from word order and grammatical relationships. Nonetheless, it often achieves good results in text classification quickly and, thus, a very useful starting point.

There are several ways to weigh a token's vector entry to capture its relevance to the document. We will illustrate below how to use sklearn to use binary flags that indicate presence or absence, counts, and weighted counts that account for differences in term frequencies across all documents, i.e., in the corpus.

## 1.1 Imports & Settings

[11]:
```
%matplotlib inline
import warnings
from collections import Counter, OrderedDict
from pathlib import Path

import numpy as np
import pandas as pd
from scipy import sparse
from scipy.spatial.distance import pdist, squareform

# Visualization
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter, ScalarFormatter
import seaborn as sns
import ipywidgets as widgets
from ipywidgets import interact, FloatRangeSlider

# spacy for language processing
import spacy

# sklearn for feature extraction & modeling
```

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer,␣
↪TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib
```

[12]:
```
plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (14.0, 8.7)
warnings.filterwarnings('ignore')
pd.options.display.float_format = '{:,.2f}'.format
```

## 1.2 Load BBC data

[13]:
```
path = Path('data', 'bbc')
files = path.glob('**/*.txt')
doc_list = []
for i, file in enumerate(files):
    topic = file.parts[-2]
    article = file.read_text(encoding='latin1').split('\n')
    heading = article[0].strip()
    body = ' '.join([l.strip() for l in article[1:]]).strip()
    doc_list.append([topic, heading, body])
```

### 1.2.1 Convert to DataFrame

[14]:
```
docs = pd.DataFrame(doc_list, columns=['topic', 'heading', 'body'])
docs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2225 entries, 0 to 2224
Data columns (total 3 columns):
topic      2225 non-null object
heading    2225 non-null object
body       2225 non-null object
dtypes: object(3)
memory usage: 52.2+ KB
```

### 1.2.2 Inspect results

[15]:
```
docs.sample(10)
```

[15]:
```
              topic                                  heading  \
53             tech        Hotspot users gain free net calls
578           sport                   A November to remember
808           sport        Barcelona title hopes hit by loss
58             tech        Ban hits Half-Life 2 pirates hard
1548       business  Business fears over sluggish EU economy
1884  entertainment         Famed music director Viotti dies
```

```
681           sport       Johnson accuses British sprinters
1609       business       Japanese mogul arrested for fraud
807           sport        Sydney return for Henin-Hardenne
1434       business        'Post-Christmas lull' in lending

                                                      body
53    People using wireless net hotspots will soon b…
578   Last Saturday, one newspaper proclaimed that E…
808   Barcelona's pursuit of the Spanish title took …
58    About 20,000 people have been banned from play…
1548  As European leaders gather in Rome on Friday t…
1884  Conductor Marcello Viotti, director of Venice'…
681   Former Olympic champion Michael Johnson has ac…
1609  One of Japan's best-known businessmen was arre…
807   Olympic champion Justine Henin-Hardenne will r…
1434  UK mortgage lending showed a "post-Christmas l…
```

### 1.2.3 Data drawn from 5 different categories

[16]:
```
docs.topic.value_counts(normalize=True).to_frame('count').style.format({'count':
 ↪ '{:,.2%}'.format})
```

[16]:
```
<pandas.io.formats.style.Styler at 0x7fa88e2d8e48>
```

## 1.3 Explore Corpus

### 1.3.1 Token Count via Counter()

[17]:
```
# word count
word_count = docs.body.str.split().str.len().sum()
print(f'Total word count: {word_count:,d} | per article: {word_count/len(docs):
 ↪,.0f}')
```

```
Total word count: 842,910 | per article: 379
```

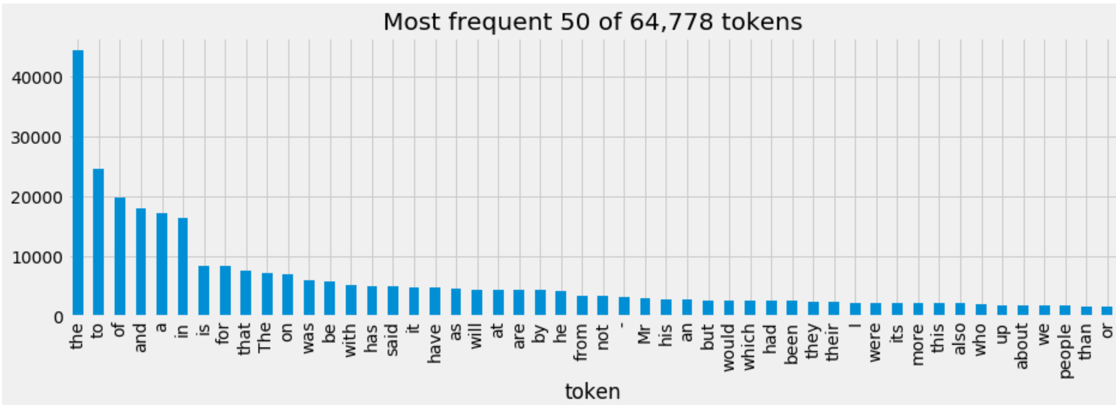
[18]:
```
token_count = Counter()
for i, doc in enumerate(docs.body.tolist(), 1):
    if i % 500 == 0:
        print(i, end=' ', flush=True)
    token_count.update([t.strip() for t in doc.split()])
```

```
500 1000 1500 2000
```

[19]:
```
tokens = (pd.DataFrame(token_count.most_common(), columns=['token', 'count'])
          .set_index('token')
          .squeeze())
```

[20]:

```
n = 50
(tokens
 .iloc[:50]
 .plot
 .bar(figsize=(14, 4), title=f'Most frequent {n} of {len(tokens):,d} tokens'));
```




## 1.4 Document-Term Matrix with `CountVectorizer`

The scikit-learn preprocessing module offers two tools to create a document-term matrix. The CountVectorizer uses binary or absolute counts to measure the term frequency tf(d, t) for each document d and token t.

The TfIDFVectorizer, in contrast, weighs the (absolute) term frequency by the inverse document frequency (idf). As a result, a term that appears in more documents will receive a lower weight than a token with the same frequency for a given document but lower frequency across all documents.

The resulting tf-idf vectors for each document are normalized with respect to their absolute or squared totals (see the sklearn documentation for details). The tf-idf measure was originally used in information retrieval to rank search engine results and has subsequently proven useful for text classification or clustering.

Both tools use the same interface and perform tokenization and further optional preprocessing of a list of documents before vectorizing the text by generating token counts to populate the document-term matrix.

Key parameters that affect the size of the vocabulary include:

- `stop_words`: use a built-in or provide a list of (frequent) words to exclude
- `ngram_range`: include n-grams in a range for n defined by a tuple of (nmin, nmax)
- `lowercase`: convert characters accordingly (default is True)
- `min_df`/ max_df: ignore words that appear in less / more (int) or a smaller / larger share of documents (if float [0.0,1.0])
- `max_features`: limit number of tokens in vocabulary accordingly
- `binary`: set non-zero counts to 1 True

### 1.4.1 Key parameters

[21]:
```python
print(CountVectorizer().__doc__)
```

```
Convert a collection of text documents to a matrix of token counts

    This implementation produces a sparse representation of the counts using
    scipy.sparse.csr_matrix.

    If you do not provide an a-priori dictionary and you do not use an analyzer
    that does some kind of feature selection then the number of features will
    be equal to the vocabulary size found by analyzing the data.

    Read more in the :ref:`User Guide <text_feature_extraction>`.

    Parameters
    ----------
    input : string {'filename', 'file', 'content'}
        If 'filename', the sequence passed as an argument to fit is
        expected to be a list of filenames that need reading to fetch
        the raw content to analyze.

        If 'file', the sequence items must have a 'read' method (file-like
        object) that is called to fetch the bytes in memory.

        Otherwise the input is expected to be the sequence strings or
        bytes items are expected to be analyzed directly.

    encoding : string, 'utf-8' by default.
        If bytes or files are given to analyze, this encoding is used to
        decode.

    decode_error : {'strict', 'ignore', 'replace'}
        Instruction on what to do if a byte sequence is given to analyze that
        contains characters not of the given `encoding`. By default, it is
        'strict', meaning that a UnicodeDecodeError will be raised. Other
        values are 'ignore' and 'replace'.

    strip_accents : {'ascii', 'unicode', None}
        Remove accents and perform other character normalization
        during the preprocessing step.
        'ascii' is a fast method that only works on characters that have
        an direct ASCII mapping.
        'unicode' is a slightly slower method that works on any characters.
        None (default) does nothing.

        Both 'ascii' and 'unicode' use NFKD normalization from
        :func:`unicodedata.normalize`.
```

```
lowercase : boolean, True by default
    Convert all characters to lowercase before tokenizing.

preprocessor : callable or None (default)
    Override the preprocessing (string transformation) stage while
    preserving the tokenizing and n-grams generation steps.

tokenizer : callable or None (default)
    Override the string tokenization step while preserving the
    preprocessing and n-grams generation steps.
    Only applies if ``analyzer == 'word'``.

stop_words : string {'english'}, list, or None (default)
    If 'english', a built-in stop word list for English is used.
    There are several known issues with 'english' and you should
    consider an alternative (see :ref:`stop_words`).

    If a list, that list is assumed to contain stop words, all of which
    will be removed from the resulting tokens.
    Only applies if ``analyzer == 'word'``.

    If None, no stop words will be used. max_df can be set to a value
    in the range [0.7, 1.0) to automatically detect and filter stop
    words based on intra corpus document frequency of terms.

token_pattern : string
    Regular expression denoting what constitutes a "token", only used
    if ``analyzer == 'word'``. The default regexp select tokens of 2
    or more alphanumeric characters (punctuation is completely ignored
    and always treated as a token separator).

ngram_range : tuple (min_n, max_n)
    The lower and upper boundary of the range of n-values for different
    n-grams to be extracted. All values of n such that min_n <= n <= max_n
    will be used.

analyzer : string, {'word', 'char', 'char_wb'} or callable
    Whether the feature should be made of word or character n-grams.
    Option 'char_wb' creates character n-grams only from text inside
    word boundaries; n-grams at the edges of words are padded with space.

    If a callable is passed it is used to extract the sequence of features
    out of the raw, unprocessed input.

max_df : float in range [0.0, 1.0] or int, default=1.0
    When building the vocabulary ignore terms that have a document
    frequency strictly higher than the given threshold (corpus-specific
```

```
    stop words).
    If float, the parameter represents a proportion of documents, integer
    absolute counts.
    This parameter is ignored if vocabulary is not None.

min_df : float in range [0.0, 1.0] or int, default=1
    When building the vocabulary ignore terms that have a document
    frequency strictly lower than the given threshold. This value is also
    called cut-off in the literature.
    If float, the parameter represents a proportion of documents, integer
    absolute counts.
    This parameter is ignored if vocabulary is not None.

max_features : int or None, default=None
    If not None, build a vocabulary that only consider the top
    max_features ordered by term frequency across the corpus.

    This parameter is ignored if vocabulary is not None.

vocabulary : Mapping or iterable, optional
    Either a Mapping (e.g., a dict) where keys are terms and values are
    indices in the feature matrix, or an iterable over terms. If not
    given, a vocabulary is determined from the input documents. Indices
    in the mapping should not be repeated and should not have any gap
    between 0 and the largest index.

binary : boolean, default=False
    If True, all non zero counts are set to 1. This is useful for discrete
    probabilistic models that model binary events rather than integer
    counts.

dtype : type, optional
    Type of the matrix returned by fit_transform() or transform().

Attributes
----------
vocabulary_ : dict
    A mapping of terms to feature indices.

stop_words_ : set
    Terms that were ignored because they either:

      - occurred in too many documents (`max_df`)
      - occurred in too few documents (`min_df`)
      - were cut off by feature selection (`max_features`).

    This is only available if no vocabulary was given.
```

```
Examples
--------
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> corpus = [
…     'This is the first document.',
…     'This document is the second document.',
…     'And this is the third one.',
…     'Is this the first document?',
… ]
>>> vectorizer = CountVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> print(vectorizer.get_feature_names())
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
>>> print(X.toarray())  # doctest: +NORMALIZE_WHITESPACE
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]

See also
--------
HashingVectorizer, TfidfVectorizer

Notes
-----
The ``stop_words_`` attribute can get large and increase the model size
when pickling. This attribute is provided only for introspection and can
be safely removed using delattr or set to None before pickling.
```

### 1.4.2 Document Frequency Distribution

[22]:
```python
binary_vectorizer = CountVectorizer(max_df=1.0,
                                    min_df=1,
                                    binary=True)

binary_dtm = binary_vectorizer.fit_transform(docs.body)
```

[23]:
```python
binary_dtm
```

[23]:
```
<2225x29275 sparse matrix of type '<class 'numpy.int64'>'
        with 445870 stored elements in Compressed Sparse Row format>
```

[24]:
```python
n_docs, n_tokens = binary_dtm.shape
```

[25]:
```python
tokens_dtm = binary_vectorizer.get_feature_names()
```

**CountVectorizer skips certain tokens by default**

[26]:
```
tokens.index.difference(pd.Index(tokens_dtm))
```

[26]:
```
Index(['!', '"', '""unconscionable,', '"'I', '"'Oh', '"'We', '"'You', '"(When',
       '"…it', '"100%',
       …
       'Â£900m', 'Â£910m).', 'Â£93.6bn)', 'Â£933m', 'Â£947m', 'Â£960m',
       'Â£98)', 'Â£99', 'Â£9m', 'Â£9m,'],
      dtype='object', length=47927)
```

**Persist Result**

[27]:
```
dtm_path = Path('data/binary_dtm.npz')
if not dtm_path.exists():
    sparse.save_npz(dtm_path, binary_dtm)
else:
    binary_dtm = sparse.load_npz(dtm_path)
```

[28]:
```
token_path = Path('data/tokens.csv')
if not token_path.exists():
    pd.Series(tokens_dtm).to_csv(token_path, index=False)
else:
    tokens = pd.read_csv(token_path, header=None, squeeze=True)
```

[29]:
```
doc_freq = pd.Series(np.array(binary_dtm.sum(axis=0)).squeeze()).div(n_docs)
max_unique_tokens = np.array(binary_dtm.sum(axis=1)).squeeze().max()
```

### 1.4.3 `min_df` vs `max_df`: Interactive Visualization

The notebook contains an interactive visualization that explores the impact of the min_df and max_df settings on the size of the vocabulary. We read the articles into a DataFrame, set the CountVectorizer to produce binary flags and use all tokens, and call its .fit_transform() method to produce a document-term matrix:

The visualization shows that requiring tokens to appear in at least 1% and less than 50% of documents restricts the vocabulary to around 10% of the almost 30K tokens. This leaves a mode of slightly over 100 unique tokens per document (left panel), and the right panel shows the document frequency histogram for the remaining tokens.

[30]:
```
df_range = FloatRangeSlider(value=[0.0, 1.0],
                            min=0,
                            max=1,
                            step=0.0001,
                            description='Doc. Freq.',
                            disabled=False,
                            continuous_update=True,
                            orientation='horizontal',
                            readout=True,
```

```python
                            readout_format='.1%',
                            layout={'width': '800px'})

@interact(df_range=df_range)
def document_frequency_simulator(df_range):
    min_df, max_df = df_range
    keep = doc_freq.between(left=min_df, right=max_df)
    left = keep.sum()

    fig, axes = plt.subplots(ncols=2, figsize=(14, 6))

    updated_dtm = binary_dtm.tocsc()[:, np.flatnonzero(keep)]
    unique_tokens_per_doc = np.array(updated_dtm.sum(axis=1)).squeeze()
    sns.distplot(unique_tokens_per_doc, ax=axes[0], kde=False, norm_hist=False)
    axes[0].set_title('Unique Tokens per Doc')
    axes[0].set_yscale('log')
    axes[0].set_xlabel('# Unique Tokens')
    axes[0].set_ylabel('# Documents (log scale)')
    axes[0].set_xlim(0, max_unique_tokens)
    axes[0].yaxis.set_major_formatter(ScalarFormatter())

    term_freq = pd.Series(np.array(updated_dtm.sum(axis=0)).squeeze())
    sns.distplot(term_freq, ax=axes[1], kde=False, norm_hist=False)
    axes[1].set_title('Document Frequency')
    axes[1].set_ylabel('# Tokens')
    axes[1].set_xlabel('# Documents')
    axes[1].set_yscale('log')
    axes[1].set_xlim(0, n_docs)
    axes[1].yaxis.set_major_formatter(ScalarFormatter())

    title = f'Document/Term Frequency Distribution | # Tokens: {left:,d} ({left/
 ↪n_tokens:.2%})'
    fig.suptitle(title, fontsize=14)
    fig.tight_layout()
    fig.subplots_adjust(top=.9)
```

```
interactive(children=(FloatRangeSlider(value=(0.0, 1.0), description='Doc. Freq.
 ↪', layout=Layout(width='800px'…
```

### 1.4.4 Most similar documents

The CountVectorizer result lets us find the most similar documents using the `pdist()` function for pairwise distances provided by the `scipy.spatial.distance` module.

It returns a condensed distance matrix with entries corresponding to the upper triangle of a square matrix.

We use `np.triu_indices()` to translate the index that minimizes the distance to the row and column indices that in turn correspond to the closest token vectors.

[84]:
```
m = binary_dtm.todense()
pairwise_distances = pdist(m, metric='cosine')
```

[86]:
```
closest = np.argmin(pairwise_distances)
```

[91]:
```
rows, cols = np.triu_indices(n_docs)
rows[closest], cols[closest]
```

[91]:
```
(11, 75)
```

[125]:
```
docs.iloc[11].to_frame(11).join(docs.iloc[75].to_frame(75)).to_csv('data/
↪most_similar.csv')
```

[113]:
```
docs.iloc[75]
```

[113]:
```
topic                                                        tech
heading                          BT program to beat dialler scams
body        BT is introducing two initiatives to help bea…
Name: 75, dtype: object
```

[114]:
```
pd.DataFrame(binary_dtm[[11,75], :].todense()).sum(0).value_counts()
```

[114]:
```
0    28873
1      344
2       58
dtype: int64
```

### 1.4.5 Baseline document-term matrix

[21]:
```
# Baseline: number of unique tokens
vectorizer = CountVectorizer() # default: binary=False
doc_term_matrix = vectorizer.fit_transform(docs.body)
doc_term_matrix
```

[21]:
```
<2225x29275 sparse matrix of type '<class 'numpy.int64'>'
        with 445870 stored elements in Compressed Sparse Row format>
```

[22]:
```
doc_term_matrix.shape
```

[22]:
```
(2225, 29275)
```

### 1.4.6 Inspect tokens

[23]:
```
# vectorizer keeps words
words = vectorizer.get_feature_names()
words[:10]
```

[23]:
```
['00',
 '000',
 '0001',
 '000bn',
 '000m',
 '000s',
 '000th',
 '001',
 '001and',
 '001st']
```

### 1.4.7 Inspect doc-term matrix

[24]:
```python
# from scipy compressed sparse row matrix to sparse DataFrame
doc_term_matrix_df = pd.SparseDataFrame(doc_term_matrix, columns=words)
doc_term_matrix_df.head()
```

[24]:
```
    00   000  0001  000bn  000m  000s  000th  001  001and  001st     …      \
0 nan 1.00   nan    nan   nan   nan    nan  nan     nan    nan     …
1 nan 1.00   nan    nan   nan   nan    nan  nan     nan    nan     …
2 nan  nan   nan    nan   nan   nan    nan  nan     nan    nan     …
3 nan  nan   nan    nan   nan   nan    nan  nan     nan    nan     …
4 nan  nan   nan    nan   nan   nan    nan  nan     nan    nan     …

   zooms  zooropa  zornotza  zorro  zubair  zuluaga  zurich  zutons  \
0    nan      nan       nan    nan     nan      nan     nan     nan
1    nan      nan       nan    nan     nan      nan     nan     nan
2   1.00      nan       nan    nan     nan      nan     nan     nan
3    nan      nan       nan    nan     nan      nan     nan     nan
4    nan      nan       nan    nan     nan      nan     nan     nan

   zvonareva  zvyagintsev
0        nan          nan
1        nan          nan
2        nan          nan
3        nan          nan
4        nan          nan

[5 rows x 29275 columns]
```

### 1.4.8 Most frequent terms

[25]:
```python
word_freq = doc_term_matrix_df.sum(axis=0).astype(int)
word_freq.sort_values(ascending=False).head()
```

[25]:
```
the     52574
to      24767
of      19930
and     18574
in      17553
dtype: int64
```

### 1.4.9 Compute relative term frequency

[27]:
```python
vectorizer = CountVectorizer(binary=True)
doc_term_matrix = vectorizer.fit_transform(docs.body)
doc_term_matrix.shape
```

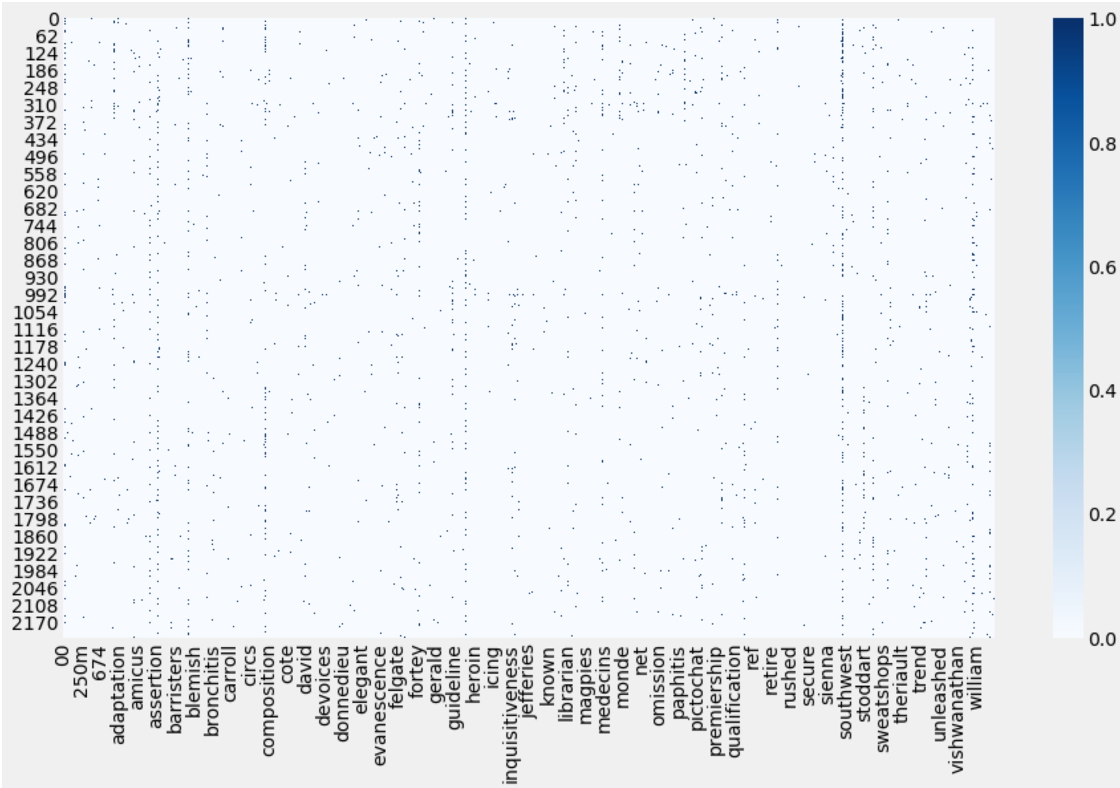
[27]:
```
(2225, 29275)
```

[28]:
```python
words = vectorizer.get_feature_names()
word_freq = doc_term_matrix.sum(axis=0)

# reduce to 1D array
word_freq_1d = np.squeeze(np.asarray(word_freq))

pd.Series(word_freq_1d, index=words).div(
    docs.shape[0]).sort_values(ascending=False).head(10)
```

[28]:
```
the     1.00
to      1.00
of      0.99
and     0.99
in      0.99
for     0.93
on      0.91
is      0.86
it      0.86
said    0.85
dtype: float64
```

### 1.4.10 Visualize Doc-Term Matrix

[29]:
```python
sns.heatmap(pd.DataFrame(doc_term_matrix.todense(), columns=words),␣
 ↪cmap='Blues')
plt.gcf().set_size_inches(14, 8);
```




### 1.4.11 Using thresholds to reduce the number of tokens

[42]:
```
vectorizer = CountVectorizer(max_df=.2, min_df=3, stop_words='english')
doc_term_matrix = vectorizer.fit_transform(docs.body)
doc_term_matrix.shape
```

[42]: (2225, 12789)

### 1.4.12 Use CountVectorizer with Lemmatization

**Building a custom tokenizer for Lemmatization with spacy**

[ ]:
```
nlp = spacy.load('en')
def tokenizer(doc):
    return [w.lemma_ for w in nlp(doc)
                if not w.is_punct | w.is_space]
```

[ ]:
```
vectorizer = CountVectorizer(tokenizer=tokenizer, binary=True)
doc_term_matrix = vectorizer.fit_transform(docs.body)
doc_term_matrix.shape
```

```
[ ]: lemmatized_words = vectorizer.get_feature_names()
word_freq = doc_term_matrix.sum(axis=0)
word_freq_1d = np.squeeze(np.asarray(word_freq))
word_freq_1d = pd.Series(word_freq_1d, index=lemmatized_words).div(docs.
 ↪shape[0])
word_freq_1d.sort_values().tail(20)
```

Unlike verbs and common nouns, there's no clear base form of a personal pronoun. Should the lemma of "me" be "I", or should we normalize person as well, giving "it" — or maybe "he"? spaCy's solution is to introduce a novel symbol, -PRON-, which is used as the lemma for all personal pronouns.

## 1.5 Document-Term Matrix with `TfIDFVectorizer`

The TfIDFTransfomer computes the tf-idf weights from a document-term matrix of token counts like the one produced by the CountVectorizer.

The TfIDFVectorizer performs both computations in a single step. It adds a few parameters to the CountVectorizer API that controls the smoothing behavior.

### 1.5.1 Key Parameters

The `TfIDFTransformer` builds on the `CountVectorizer` output; the `TfIDFVectorizer` integrates both

```
[43]: print(TfidfTransformer().__doc__)
```

```
Transform a count matrix to a normalized tf or tf-idf representation

    Tf means term-frequency while tf-idf means term-frequency times inverse
    document-frequency. This is a common term weighting scheme in information
    retrieval, that has also found good use in document classification.

    The goal of using tf-idf instead of the raw frequencies of occurrence of a
    token in a given document is to scale down the impact of tokens that occur
    very frequently in a given corpus and that are hence empirically less
    informative than features that occur in a small fraction of the training
    corpus.

    The formula that is used to compute the tf-idf of term t is
    tf-idf(d, t) = tf(t) * idf(d, t), and the idf is computed as
    idf(d, t) = log [ n / df(d, t) ] + 1 (if ``smooth_idf=False``),
    where n is the total number of documents and df(d, t) is the
    document frequency; the document frequency is the number of documents d
    that contain term t. The effect of adding "1" to the idf in the equation
    above is that terms with zero idf, i.e., terms  that occur in all documents
    in a training set, will not be entirely ignored.
    (Note that the idf formula above differs from the standard
    textbook notation that defines the idf as
```

```
idf(d, t) = log [ n / (df(d, t) + 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(d, t) = log [ (1 + n) / (1 + df(d, t)) ] + 1.

Furthermore, the formulas used to compute tf and idf depend
on parameter settings that correspond to the SMART notation used in IR
as follows:

Tf is "n" (natural) by default, "l" (logarithmic) when
``sublinear_tf=True``.
Idf is "t" when use_idf is given, "n" (none) otherwise.
Normalization is "c" (cosine) when ``norm='l2'``, "n" (none)
when ``norm=None``.

Read more in the :ref:`User Guide <text_feature_extraction>`.

Parameters
----------
norm : 'l1', 'l2' or None, optional
    Norm used to normalize term vectors. None for no normalization.

use_idf : boolean, default=True
    Enable inverse-document-frequency reweighting.

smooth_idf : boolean, default=True
    Smooth idf weights by adding one to document frequencies, as if an
    extra document was seen containing every term in the collection
    exactly once. Prevents zero divisions.

sublinear_tf : boolean, default=False
    Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).

Attributes
----------
idf_ : array, shape (n_features)
    The inverse document frequency (IDF) vector; only defined
    if  ``use_idf`` is True.

References
----------

.. [Yates2011] `R. Baeza-Yates and B. Ribeiro-Neto (2011). Modern
               Information Retrieval. Addison Wesley, pp. 68-74.`

.. [MRS2008] `C.D. Manning, P. Raghavan and H. Schütze  (2008).
```

```
Introduction to Information Retrieval. Cambridge University
Press, pp. 118-120.`
```

### 1.5.2 How Term Frequency - Inverse Document Frequency works

The TFIDF computation works as follows for a small text sample

[7]:
```
sample_docs = ['call you tomorrow',
               'Call me a taxi',
               'please call me... PLEASE!']
```

#### Compute term frequency

[8]:
```
vectorizer = CountVectorizer()
tf_dtm = vectorizer.fit_transform(sample_docs).todense()
tokens = vectorizer.get_feature_names()
```

[9]:
```
term_frequency = pd.DataFrame(data=tf_dtm,
                              columns=tokens)
print(term_frequency)
```

```
   call  me  please  taxi  tomorrow  you
0     1   0       0     0         1    1
1     1   1       0     1         0    0
2     1   1       2     0         0    0
```

#### Compute document frequency

[10]:
```
vectorizer = CountVectorizer(binary=True)
df_dtm = vectorizer.fit_transform(sample_docs).todense().sum(axis=0)
```

[12]:
```
document_frequency = pd.DataFrame(data=df_dtm,
                                  columns=tokens)
print(document_frequency)
```

```
   call  me  please  taxi  tomorrow  you
0     3   2       1     1         1    1
```

#### Compute TfIDF

[13]:
```
tfidf = pd.DataFrame(data=tf_dtm/df_dtm, columns=tokens)
print(tfidf)
```

```
   call   me  please  taxi  tomorrow  you
0  0.33 0.00    0.00  0.00      1.00 1.00
1  0.33 0.50    0.00  1.00      0.00 0.00
2  0.33 0.50    2.00  0.00      0.00 0.00
```

**The effect of smoothing** The TfidfVectorizer uses smoothing for document and term frequencies: - `smooth_idf`: add one to document frequency, as if an extra document contained every token in the vocabulary once to prevents zero divisions - `sublinear_tf`: scale term Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf)

[14]:
```
vect = TfidfVectorizer(smooth_idf=True,
                       norm='l2',            # squared weights sum to 1 by␣
 ↪document
                       sublinear_tf=False,   # if True, use 1+log(tf)
                       binary=False)
print(pd.DataFrame(vect.fit_transform(sample_docs).todense(),
             columns=vect.get_feature_names()))
```

```
   call   me  please  taxi  tomorrow  you
0  0.39 0.00    0.00  0.00      0.65 0.65
1  0.43 0.55    0.00  0.72      0.00 0.00
2  0.27 0.34    0.90  0.00      0.00 0.00
```

### 1.5.3 TfIDF with new articles

Due to their ability to assign meaningful token weights, TFIDF vectors are also used to summarize text data. E.g., reddit's autotldr function is based on a similar algorithm.

[126]:
```
tfidf = TfidfVectorizer(stop_words='english')
dtm_tfidf = tfidf.fit_transform(docs.body)
tokens = tfidf.get_feature_names()
dtm_tfidf.shape
```

[126]:
```
(2225, 28980)
```

[127]:
```
token_freq = (pd.DataFrame({'tfidf': dtm_tfidf.sum(axis=0).A1,
                            'token': tokens})
              .sort_values('tfidf', ascending=False))
```

[128]:
```
token_freq.head(10).append(token_freq.tail(10)).set_index('token')
```

[128]:
```
            tfidf
token
said        87.25
mr          58.22
year        41.98
people      37.30
new         34.20
film        29.73
government  28.79
world       27.03
time        26.36
best        26.30
```

```
baked         0.01
pavlovian     0.01
buzzcocks     0.01
sisterhood    0.01
siouxsie      0.01
sioux         0.01
bane          0.01
biassed       0.01
duetted       0.01
speechless    0.01
```

### 1.5.4 Summarizing news articles using TfIDF weights

**Select random article**

[140]:
```
article = docs.sample(1).squeeze()
article_id = article.name
```

[141]:
```
print(f'Topic:\t{article.topic.capitalize()}\n\n{article.heading}\n')
print(article.body.strip())
```

```
Topic:  Politics

MPs issued with Blackberry threat

MPs will be thrown out of the Commons if they use Blackberries in the chamber
Speaker Michael Martin has ruled.  The Â£200 handheld computers can be used as a
phone, pager or to send e-mails. The devices gained new prominence this week
after Alastair Campbell used his to accidentally send an expletive-laden message
to a Newsnight journalist. Mr Martin revealed some MPs had been using their
Blackberries during debates and he also cautioned members against using hidden
earpieces.  The use of electronic devices in the Commons chamber has long been
frowned on. The sound of a mobile phone or a pager can result in a strong rebuke
from either the Speaker or his deputies. The Speaker chairs debates in the
Commons and is charged with ensuring order in the chamber and enforcing rules
and conventions of the House. He or she is always an MP chosen by colleagues
who, once nominated, gives up all party political allegiances.
```

**Select most relevant tokens by tfidf value**

[142]:
```
article_tfidf = dtm[article_id].todense().A1
article_tokens = pd.Series(article_tfidf, index=tokens)
article_tokens.sort_values(ascending=False).head(10)
```

[142]:
```
speaker         0.33
chamber         0.31
blackberries    0.27
pager           0.26
debates         0.23
```

```
commons         0.22
send            0.15
devices         0.15
mps             0.15
martin          0.14
dtype: float64
```

**Compare to random selection**

[144]:
```
pd.Series(article.body.split()).sample(10).tolist()
```

[144]:
```
['Campbell',
 'after',
 'in',
 'deputies.',
 'as',
 'strong',
 'using',
 'Speaker',
 'either',
 'be']
```

## 1.6 Create Train & Test Sets

### 1.6.1 Stratified train_test_split

[34]:
```
train_docs, test_docs = train_test_split(docs,
                                         stratify=docs.topic,
                                         test_size=50,
                                         random_state=42)
```

[35]:
```
train_docs.shape, test_docs.shape
```

[35]:
```
((2175, 3), (50, 3))
```

[36]:
```
pd.Series(test_docs.topic).value_counts()
```

[36]:
```
sport            12
business         11
entertainment     9
tech              9
politics          9
Name: topic, dtype: int64
```

### 1.6.2 Vectorize train & test sets

[38]:
```
vectorizer = CountVectorizer(max_df=.2,
                             min_df=3,
                             stop_words='english',
                             max_features=2000)

train_dtm = vectorizer.fit_transform(train_docs.body)
words = vectorizer.get_feature_names()
train_dtm
```

[38]:
```
<2175x2000 sparse matrix of type '<class 'numpy.int64'>'
        with 178572 stored elements in Compressed Sparse Row format>
```

[40]:
```
test_dtm = vectorizer.transform(test_docs.body)
test_dtm
```

[40]:
```
<50x2000 sparse matrix of type '<class 'numpy.int64'>'
        with 4160 stored elements in Compressed Sparse Row format>
```

[ ]:
```

```