Stop words

SENTIMENT ANALYSIS IN PYTHON

Violeta Misheva
Data Scientist

What are stop words and how to find them?

> Stop words: words that occur too frequently and not considered informative

- Lists of stop words in most languages

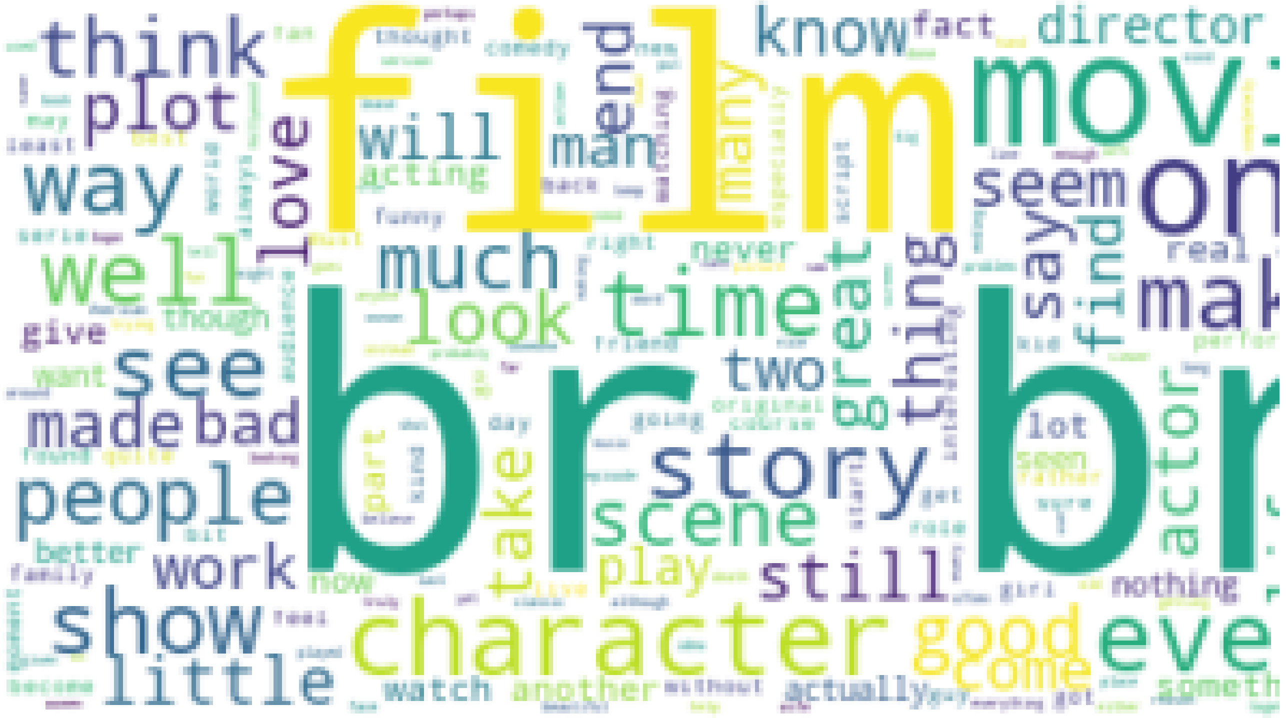
```
{'the', 'a', 'an', 'and', 'but', 'for', 'on', 'in', 'at' ...}
```

- Context matters

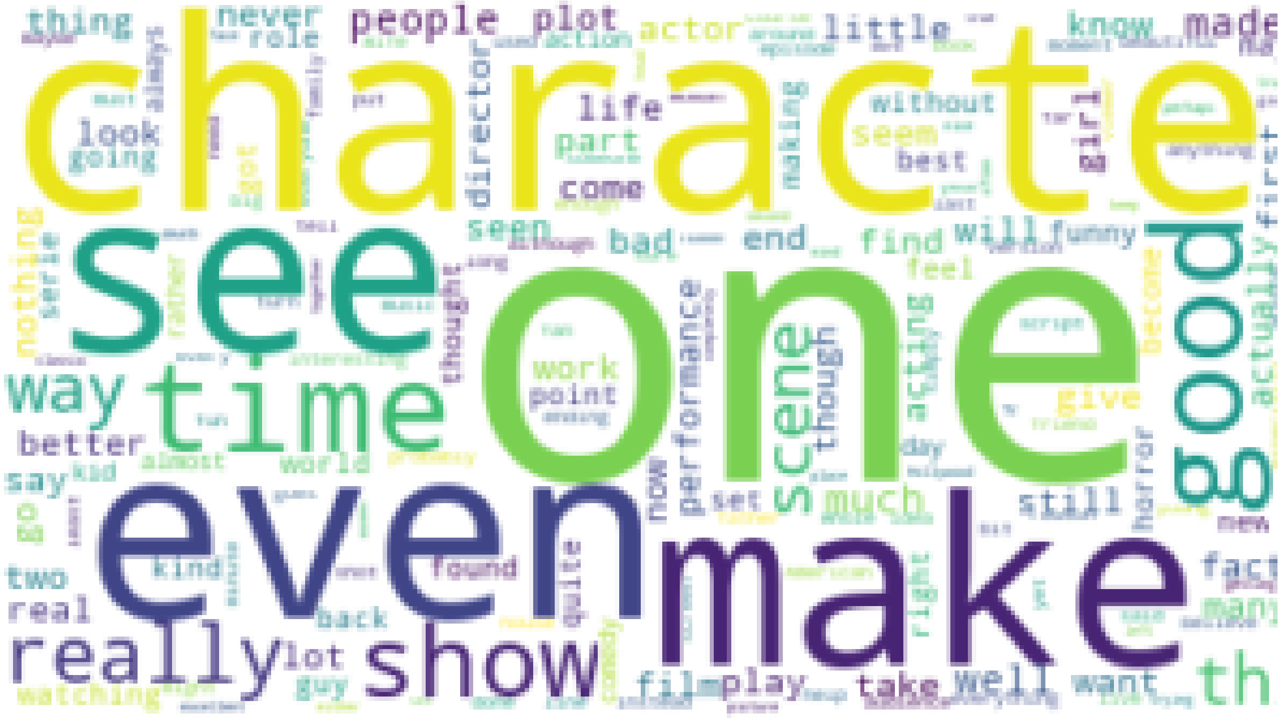
```
{'movie', 'movies', 'film', 'films', 'cinema'}
```

Stop words with word clouds

- Word cloud, not removing stop words



- Word cloud with stop words removed



Remove stop words from word clouds

```python
# Import libraries
from wordcloud import WordCloud, STOPWORDS
```

```python
# Define the stopwords list
my_stopwords = set(STOPWORDS)
my_stopwords.update(["movie", "movies", "film", "films", "watch", "br"])
```

```python
# Generate and show the word cloud
my_cloud = WordCloud(background_color='white', stopwords=my_stopwords).generate(name_string)
plt.imshow(my_cloud, interpolation='bilinear')
```

Stop words with BOW

```python
from sklearn.feature_extraction.text import CountVectorizer, ENGLISH_STOP_WORDS
```

```python
# Define the set of stop words
my_stop_words = ENGLISH_STOP_WORDS.union(['film', 'movie', 'cinema', 'theatre'])
```

```python
vect = CountVectorizer(stop_words=my_stop_words)
vect.fit(movies.review)
X = vect.transform(movies.review)
```

Let's practice!

SENTIMENT ANALYSIS IN PYTHON

Capturing a token pattern

SENTIMENT ANALYSIS IN PYTHON

Violeta Misheva
Data Scientist

String operators and comparisons

```
# Checks if a string is composed only of letters
my_string.isalpha()
```

```
# Checks if a string is composed only of digits
my_string.isdigit()
```

```
# Checks if a string is composed only of alphanumeric characters
my_string.isalnum()
```

String operators with list comprehension

```
# Original word tokenization
word_tokens = [word_tokenize(review) for review in reviews.review]
```

```
# Keeping only tokens composed of letters
cleaned_tokens = [[word for word in item if word.isalpha()] for item in word_tokens]
```

```
len(word_tokens[0])
```

```
87
```

```
len(cleaned_tokens[0])
```

```
78
```

Regular expressions

```python
import re
```

```python
my_string = '#Wonderfulday'
# Extract #, followed by any letter, small or capital
x = re.search('#[A-Za-z]', my_string)
```

```python
x
<re.Match object; span=(0, 2), match='#W'>
```

Token pattern with a BOW

```
# Default token pattern in CountVectorizer
'\b\w\w+\b'
```

```
# Specify a particular token pattern
CountVectorizer(token_pattern=r'\b[^\d\W][^\d\W]+\b')
```

Let's practice!

Stemming and lemmatization

SENTIMENT ANALYSIS IN PYTHON

Violeta Misheva
Data Scientist

What is stemming?

> Stemming is the process of transforming words to their root forms, even if the stem itself is not a valid word in the language.

```
staying, stays, stayed ----> stay
house, houses, housing ----> hous
```

What is lemmatization?

> Lemmatization is quite similar to stemming but unlike stemming, it reduces the words to roots that are valid words in the language.

```
stay, stays, staying, stayed ----> stay
house, houses, housing ----> house
```

Stemming vs. lemmatization

Stemming

- Produces roots of words
- Fast and efficient to compute

Lemmatization

- Produces actual words
- Slower than stemming and can depend on the part-of-speech

Stemming of strings

```python
from nltk.stem import PorterStemmer

porter = PorterStemmer()
```

```python
porter.stem('wonderful')
```

```
'wonder'
```

Non-English stemmers

> **Snowball Stemmer:** Danish, Dutch, English, Finnish, French, German, Hungarian,Italian, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish

```
from nltk.stem.snowball import SnowballStemmer

DutchStemmer = SnowballStemmer("dutch")
DutchStemmer.stem("beginnen")
```

```
'begin'
```

How to stem a sentence?

```
porter.stem('Today is a wonderful day!')
```

```
'today is a wonderful day!'
```

```
tokens = word_tokenize('Today is a wonderful day!')
stemmed_tokens = [porter.stem(token) for token in tokens]
stemmed_tokens
```

```
['today', 'is', 'a', 'wonder', 'day', '!']
```

Lemmatization of a string

```
from nltk.stem import WordNetLemmatizer

WNlemmatizer = WordNetLemmatizer()
```

```
WNlemmatizer.lemmatize('wonderful', pos='a')
```

```
'wonderful'
```

Let's practice!

SENTIMENT ANALYSIS IN PYTHON

TfIdf: More ways to transform text

SENTIMENT ANALYSIS IN PYTHON

Violeta Misheva
Data Scientist

What are the components of TfIdf?

- **TF: term frequency:** How often a given word appears within a document in the corpus
- **Inverse document frequency:** Log-ratio between the total number of documents and the number of documents that contain a specific word
  - Used to calculate the weight of words that do not occur frequently

TfIdf score of a word

- **TfIdf score:**

```
TfIdf = term frequency * inverse document frequency
```

- BOW does not account for length of a document, TfIdf does.
- TfIdf likely to capture words common within a document but not across documents.

How is TfIdf useful?

Twitter airline sentiment

- Low TfIdf scores: United, Virgin America
- High TfIdf scores: check-in process (if rare across documents)

More on TfIdf

- Since it penalizes frequent words, less need to deal with stop words explicitly.
- Quite useful in search queries and information retrieval to rank the relevance of returned results.

TfIdf in Python

```
# Import the TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

- **Arguments of TfidfVectorizer:** max_features, ngrams_range, stop_words, token_pattern, max_df, min_df

```
vect = TfidfVectorizer(max_features=100).fit(tweets.text)
X = vect.transform(tweets.text)
```

TfidfVectorizer

```
X
```

```
<14640x100 sparse matrix of type '<class 'numpy.float64'>'
    with 119182 stored elements in Compressed Sparse Row format>
```

```
X_df = pd.DataFrame(X_txt.toarray(), columns=vect.get_feature_names())
X_df.head()
```

	about	after	again	airline	all	am	americanair	amp	an	and	...	was	we	what	when	why	will	with	would	you	your
0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.0	0.0	0.668165	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000
1	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.32904	0.000000
2	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.000000	...	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000
3	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.431149	0.0	0.000000	...	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.00000	0.332355
4	0.494872	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.279754	...	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000

Let's practice!

SENTIMENT ANALYSIS IN PYTHON