Building tf-idf document vectors

FEATURE ENGINEERING FOR NLP IN PYTHON

Rounak Banik
Data Scientist

n-gram modeling

- Weight of dimension dependent on the frequency of the word corresponding to the dimension.
  - Document contains the word `human` in five places.
  - Dimension corresponding to `human` has weight `5`.

Motivation

- Some words occur very commonly across all documents
- Corpus of documents on the universe
  - One document has `jupiter` and `universe` occurring 20 times each.
  - `jupiter` rarely occurs in the other documents. `universe` is common.
  - Give more weight to `jupiter` on account of exclusivity.

Applications

- Automatically detect stopwords
- Search
- Recommender systems
- Better performance in predictive modeling for some cases

Term frequency-inverse document frequency

- Proportional to term frequency
- Inverse function of the number of documents in which it occurs

Mathematical formula

$$w_{i,j} = tf_{i,j} \cdot \log\left(\frac{N}{df_i}\right)$$

$w_{i,j} \rightarrow$ weight of term i in document j

Mathematical formula

$$w_{i,j} = tf_{i,j} \cdot \log\left(\frac{N}{df_i}\right)$$

$w_{i,j} \rightarrow$ weight of term i in document j

$tf_{i,j} \rightarrow$ term frequency of term i in document j

Mathematical formula

$$w_{i,j} = tf_{i,j} \cdot \log\left(\frac{N}{df_i}\right)$$

$w_{i,j} \rightarrow$ weight of term i in document j

$tf_{i,j} \rightarrow$ term frequency of term $i$in document j

$N \rightarrow$ number of documents in the corpus

$df_i \rightarrow$ number of documents containing term i

Mathematical formula

$$w_{i,j} = tf_{i,j} \cdot \log\left(\frac{N}{df_i}\right)$$

$$w_{i,j} \rightarrow \text{weight of term } i \text{ in document } j$$

$$tf_{i,j} \rightarrow term\ frequency\ of\ term\ i\ in\ document\ j$$

$$N \rightarrow number\ of\ documents\ in\ the\ corpus$$

$$df_i \rightarrow number\ of\ documents\ cotaining\ term\ i$$

Example:

$$w_{library,document} = 5 \cdot log(\tfrac{20}{8}) \approx 2$$

tf-idf using scikit-learn

```
# Import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
# Create TfidfVectorizer object
vectorizer = TfidfVectorizer()
# Generate matrix of word vectors
tfidf_matrix = vectorizer.fit_transform(corpus)
print(tfidf_matrix.toarray())
```

```
[[0.         0.         0.         0.         0.25434658 0.33443519
  0.33443519 0.         0.25434658 0.         0.25434658 0.
  0.76303975]
 [0.         0.46735098 0.         0.46735098 0.         0.
  0.         0.46735098 0.         0.46735098 0.35543247 0.
  0.        ]
...
```

Let's practice!

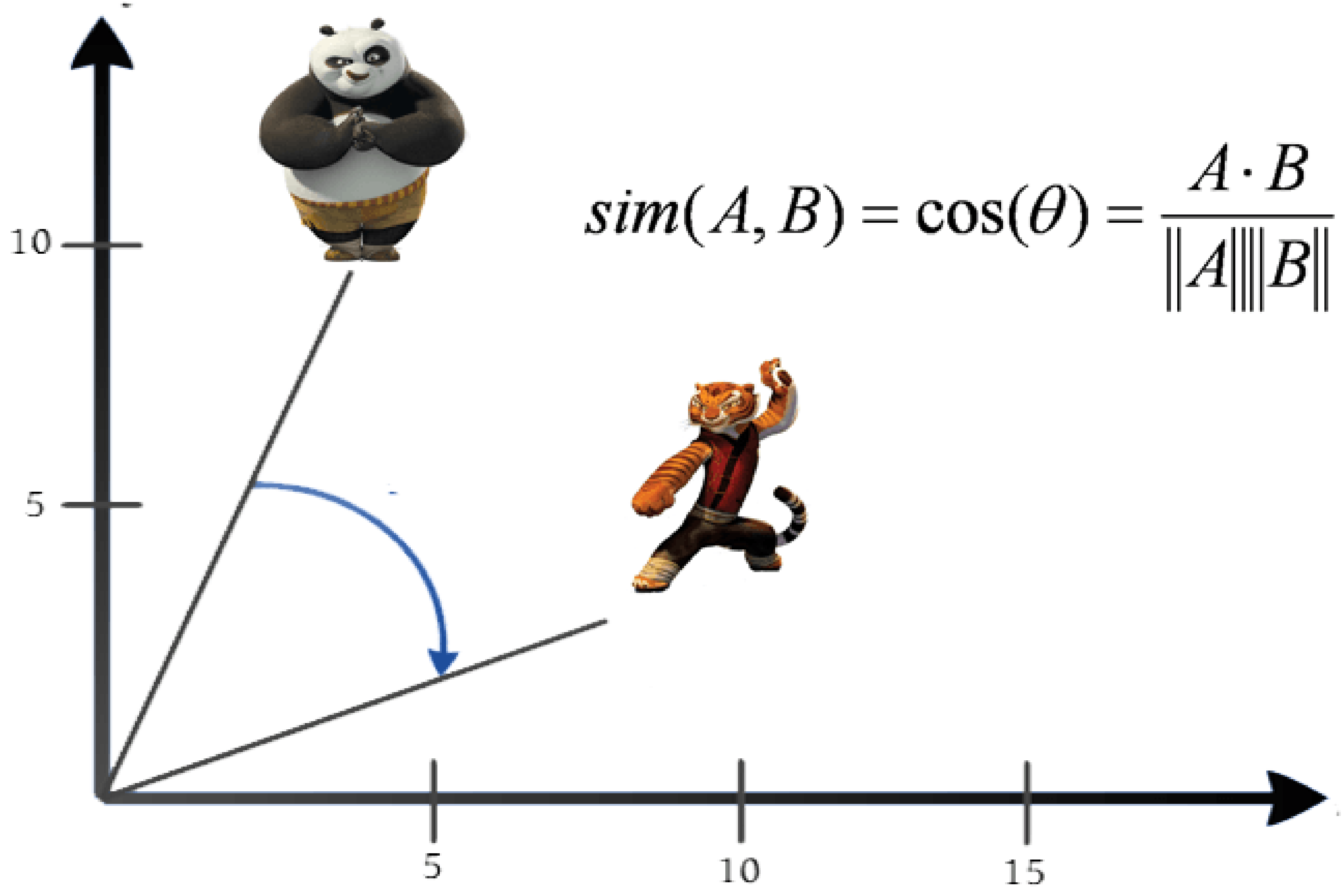
FEATURE ENGINEERING FOR NLP IN PYTHON

Cosine similarity

FEATURE ENGINEERING FOR NLP IN PYTHON

Rounak Banik
Data Scientist

Cosine Similarity



$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

[1] Image courtesy techninpink.com

The dot product

Consider two vectors,

$$V = (v_1, v_2, \cdots, v_n), W = (w_1, w_2, \cdots, w_n)$$

Then the dot product of V and W is,

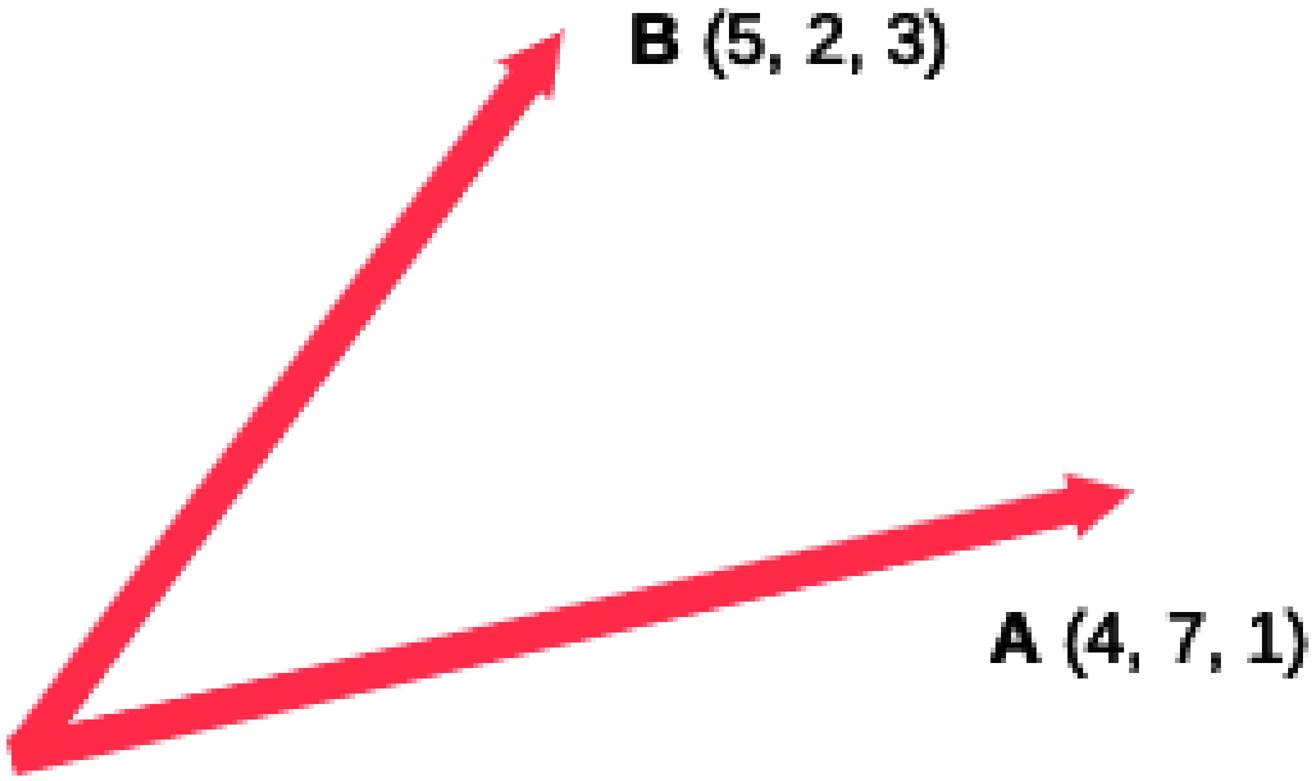
$$V \cdot W = (v_1 \times w_1) + (v_2 \times w_2) + \cdots + (v_n \times w_n)$$

Example:

$$A = (4, 7, 1) \ , \ B = (5, 2, 3)$$

$$A \cdot B = (4 \times 5) + (7 \times 2) + \cdots (1 \times 3)$$

$$= 20 + 14 + 3 = 37$$



Magnitude of a vector

For any vector,

$$V = (v_1, v_2, \cdots, v_n)$$

The magnitude is defined as,

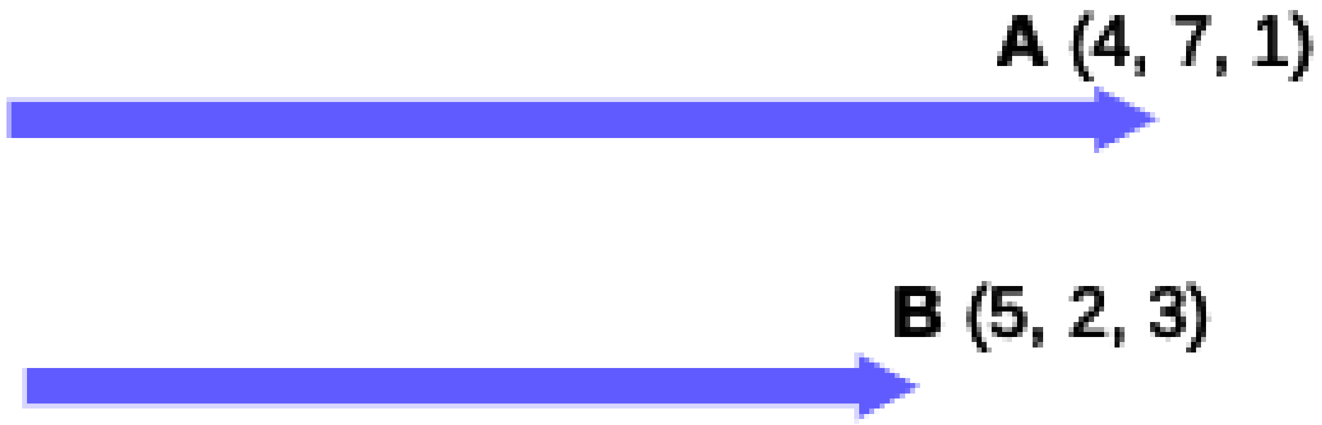
$$||\mathbf{V}|| = \sqrt{(v_1)^2 + (v_2)^2 + ... + (v_n)^2}$$

Example:

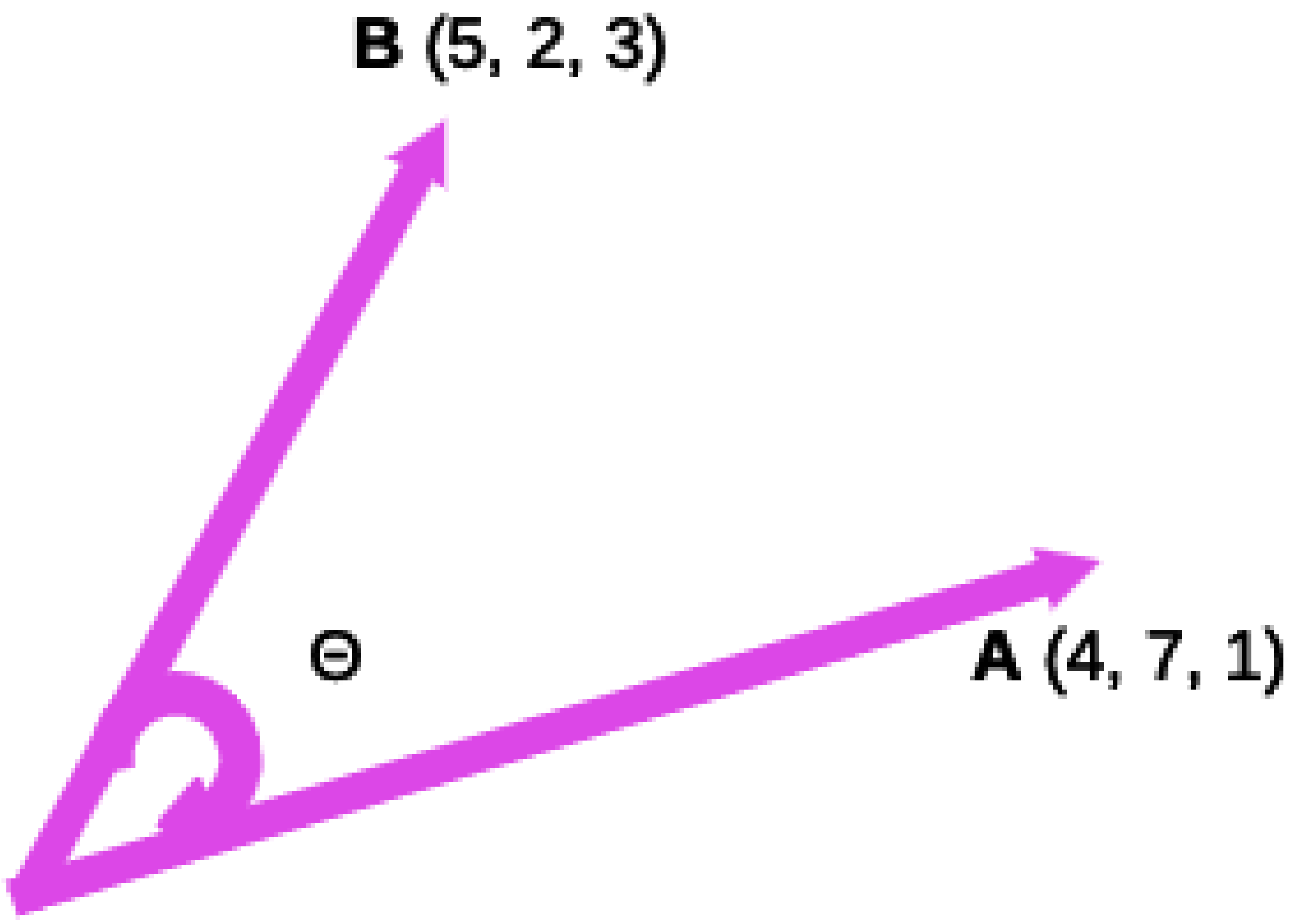
$$A = (4, 7, 1)\ ,\ B = (5, 2, 3)$$

$$||\mathbf{A}|| = \sqrt{(4)^2 + (7)^2 + (1)^2}$$

$$= \sqrt{16 + 49 + 1} = \sqrt{66}$$



The cosine score



$$A : (4, 7, 1)$$

$$B : (5, 2, 3)$$

The cosine score,

$$cos(A, B) = \frac{A \cdot B}{|A| \cdot |B|}$$

$$= \frac{37}{\sqrt{66} \times \sqrt{38}}$$

$$= 0.7388$$

Cosine Score: points to remember

- Value between -1 and 1.
- In NLP, value between 0 and 1.
- Robust to document length.

Implementation using scikit-learn

```
# Import the cosine_similarity
from sklearn.metrics.pairwise import cosine_similarity
```

```
# Define two 3-dimensional vectors A and B
A = (4,7,1)
B = (5,2,3)

# Compute the cosine score of A and B
score = cosine_similarity([A], [B])

# Print the cosine score
print(score)
```

```
array([[ 0.73881883]])
```

Let's practice!

FEATURE ENGINEERING FOR NLP IN PYTHON

Building a plot line based recommender

FEATURE ENGINEERING FOR NLP IN PYTHON

Rounak Banik
Data Scientist

Movie recommender

Title	Overview
Shanghai Triad	A provincial boy related to a Shanghai crime family is recruited by his uncle into cosmopolitan Shanghai in the 1930s to be a servant to a ganglord's mistress.
Cry, the Beloved Country	A South-African preacher goes to search for his wayward son who has committed a crime in the big city.

Movie recommender

```
get_recommendations("The Godfather")
```

```
1178                The Godfather: Part II
44030     The Godfather Trilogy: 1972-1990
1914               The Godfather: Part III
23126                          Blood Ties
11297                    Household Saints
34717                   Start Liquidation
10821                            Election
38030                          Goodfellas
17729                   Short Sharp Shock
26293                  Beck 28 - Familjen
Name: title, dtype: object
```

Steps

1. Text preprocessing
2. Generate tf-idf vectors
3. Generate cosine similarity matrix

The recommender function

1. Take a movie title, cosine similarity matrix and indices series as arguments.
2. Extract pairwise cosine similarity scores for the movie.
3. Sort the scores in descending order.
4. Output titles corresponding to the highest scores.
5. Ignore the highest similarity score (of 1).

Generating tf-idf vectors

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

# Create TfidfVectorizer object
vectorizer = TfidfVectorizer()

# Generate matrix of tf-idf vectors
tfidf_matrix = vectorizer.fit_transform(movie_plots)
```

Generating cosine similarity matrix

```
# Import cosine_similarity
from sklearn.metrics.pairwise import cosine_similarity

# Generate cosine similarity matrix
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

```
array([[1.        , 0.27435345, 0.23092036, ..., 0.        , 0.        ,
        0.00758112],
       [0.27435345, 1.        , 0.1246955 , ..., 0.        , 0.        ,
        0.00740494],
       ...,
       [0.00758112, 0.00740494, 0.        , ..., 0.        , 0.        ,
        1.        ]])
```

The linear_kernel function

- Magnitude of a tf-idf vector is 1
- Cosine score between two tf-idf vectors is their dot product.
- Can significantly improve computation time.
- Use `linear_kernel` instead of `cosine_similarity` .

Generating cosine similarity matrix

```python
# Import cosine_similarity
from sklearn.metrics.pairwise import linear_kernel

# Generate cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
array([[1.        , 0.27435345, 0.23092036, ..., 0.        , 0.        ,
        0.00758112],
       [0.27435345, 1.        , 0.1246955 , ..., 0.        , 0.        ,
        0.00740494],
       ...,
       [0.00758112, 0.00740494, 0.        , ..., 0.        , 0.        ,
        1.        ]])
```

The get_recommendations function

```
get_recommendations('The Lion King', cosine_sim, indices)
```

```
7782                         African Cats
5877     The Lion King 2: Simba's Pride
4524                            Born Free
2719                             The Bear
4770      Once Upon a Time in China III
7070                           Crows Zero
739                      The Wizard of Oz
8926                      The Jungle Book
1749                    Shadow of a Doubt
7993                         October Baby
Name: title, dtype: object
```

Let's practice!

FEATURE ENGINEERING FOR NLP IN PYTHON

Beyond n-grams: word embeddings

FEATURE ENGINEERING FOR NLP IN PYTHON

Rounak Banik
Data Scientist

The problem with BoW and tf-idf

```
'I am happy'
```

```
'I am joyous'
```

```
'I am sad'
```

Word embeddings

- Mapping words into an n-dimensional vector space
- Produced using deep learning and huge amounts of data
- Discern how similar two words are to each other
- Used to detect synonyms and antonyms
- Captures complex relationships
  - `King` - `Queen` → `Man` - `Woman`
  - `France` - `Paris` → `Russia` - `Moscow`
- Dependent on spacy model; independent of dataset you use

Word embeddings using spaCy

```
import spacy

# Load model and create Doc object
nlp = spacy.load('en_core_web_lg')
doc = nlp('I am happy')
```

```
# Generate word vectors for each token
for token in doc:
  print(token.vector)
```

```
[-1.0747459e+00  4.8677087e-02  5.6630421e+00  1.6680446e+00
 -1.3194644e+00 -1.5142369e+00  1.1940931e+00 -3.0168812e+00
 ...
```

Word similarities

```python
doc = nlp("happy joyous sad")
for token1 in doc:
  for token2 in doc:
    print(token1.text, token2.text, token1.similarity(token2))
```

```
happy happy 1.0
happy joyous 0.63244456
happy sad 0.37338886
joyous happy 0.63244456
joyous joyous 1.0
joyous sad 0.5340932
...
```

Document similarities

```
# Generate doc objects
sent1 = nlp("I am happy")
sent2 = nlp("I am sad")
sent3 = nlp("I am joyous")
```

```
# Compute similarity between sent1 and sent2
sent1.similarity(sent2)
```

```
0.9273363837282105
```

```
# Compute similarity between sent1 and sent3
sent1.similarity(sent3)
```

```
0.9403554938594568
```

Let's practice!

FEATURE ENGINEERING FOR NLP IN PYTHON

Congratulations!

FEATURE ENGINEERING FOR NLP IN PYTHON

Rounak Banik
Data Scientist

Review

- Basic features (characters, words, mentions, etc.)
- Readability scores
- Tokenization and lemmatization
- Text cleaning
- Part-of-speech tagging & named entity recognition
- n-gram modeling
- tf-idf
- Cosine similarity
- Word embeddings

Further resources

- Advanced NLP with spaCy
- Deep Learning in Python

Thank you!

FEATURE ENGINEERING FOR NLP IN PYTHON