Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica

Lesson 3

Math with Words

Nicola Capuano and Antonio Greco

DIEM – University of Salerno

Outline

- Term Frequency
- Vector Space Model
- TF-IDF
- Building a Search Engine

Term Frequency

Bag of Words

A **vector space model** of text

- One-hot encode each word in a text and combine one-hot vectors
- **Binary BoW:** one-hot vectors are combined with the **OR** operation
- **Standard BoW:** one-hot vectors are summed
  - **Term Frequency (TF):** number of occurrences of each word in the text

Assumption:

- The **more times** a word occurs, the **more meaning** it contributes to that document

Calculating TF

```
# Extract tokens

sentence = "The faster Harry got to the store, the faster Harry, the faster, would get home."

import spacy
nlp = spacy.load("en_core_web_sm") # load the language model
doc = nlp(sentence)

tokens = [tok.lower_ for tok in doc if not tok.is_stop and not tok.is_punct]
tokens
```

```
['faster', 'harry', 'got', 'store', 'faster', 'harry', 'faster', 'home']
```

```
# Build BoW with word count

import collections
bag_of_words = collections.Counter(tokens) # counts the elements of a list
bag_of_words
```

```
Counter({'faster': 3, 'harry': 2, 'got': 1, 'store': 1, 'home': 1})
```

```
# Most common words

bag_of_words.most_common(2) # most common 2 words
```

```
[('faster', 3), ('harry', 2)]
```

Limits of TF

Example:

- In document A the word dog appears **3 times**
- In document B the word dog appears **100 times**

Is the word dog more important for document A or B?

Additional information:

- Document A is a 30-word email to a veterinarian
- Document B is the novel **War & Peace** (approx. 580,000 words)

Is the word dog more important for document A or B?

Normalized TF

Normalized (weighted) TF is the word count **normalized by the document length**

- TF (dog, $document_A$) = 3/30 = 0.1
- TF (dog, $document_B$) = 100/580000 = 0.00017

```
import pandas as pd

counts = pd.Series(bag_of_words) # from dict to Pandas Series
counts / counts.sum() # calculate TF
```

```
faster    0.375
harry     0.250
got       0.125
store     0.125
home      0.125
dtype: float64
```

Vector Space Model

NLTK Corpora

Natural Language Toolkit includes several **text corpora**

- They can be used to train and test NLP algorithms
- It has a package to easily access corpus data
- https://www.nltk.org/howto/corpus.html

Reuters 21578 corpus

- Widely used for NLP and text classification
- It contains **thousands of news** published by Reuters in 1986
- News are categorized into 90 different **topics**

Using Reuters 21578

```
import nltk

nltk.download('reuters') # download the reuters corpus
ids = nltk.corpus.reuters.fileids() # ids of the documents
sample = nltk.corpus.reuters.raw(ids[0]) # first document

print(len(ids), "samples.\n") # number of documents
print(sample)
```

```
     Taiwan had a trade trade surplus of 15.6 billion dlrs last
  year, 95 pct of it with the U.S.
     The surplus helped swell Taiwan's foreign exchange reserves
  to 53 billion dlrs, among the world's largest.
     "We must quickly open our markets, remove trade barriers and
  cut import tariffs to allow imports of U.S. Products, if we
  want to defuse problems from possible U.S. Retaliation," said
  Paul Sheen, chairman of textile exporters &lt;Taiwan Safe Group>.
     A senior official of South Korea's trade promotion
  association said the trade dispute between the U.S. And Japan
  might also lead to pressure on South Korea, whose chief exports
  are similar to those of Japan.
     Last year South Korea had a trade surplus of 7.1 billion
  dlrs with the U.S., Up from 4.9 billion dlrs in 1985.
     In Malaysia, trade officers and businessmen said tough
  curbs against Japan might allow hard-hit producers of
  semiconductors in third countries to expand their sales to the
  U.S.

  ...
```

Using Reuters 21578

```
doc = nlp(sample)

tokens = [tok.lower_ for tok in doc if not tok.is_stop and not tok.is_punct and not tok.is_space]

bag_of_words = collections.Counter(tokens)
counts = pd.Series(bag_of_words).sort_values(ascending=False) # sorted series
counts = counts / counts.sum() # calculate TF

print(counts.head(10))
```

```
u.s.           0.039627
said           0.037296
trade          0.034965
japan          0.027972
dlrs           0.013986
exports        0.013986
tariffs        0.011655
imports        0.011655
billion        0.011655
electronics    0.009324
dtype: float64
```

Most relevant words in the first news

Corpus Processing

```
ids_subset = ids[:100] # to speed-up we consider only the first 100 elements
counts_list = []

for i, id in enumerate(ids_subset):
    sample = nltk.corpus.reuters.raw(id)
    doc = nlp(sample)
    tokens = [tok.lower_ for tok in doc if not tok.is_stop and not tok.is_punct and not tok.is_space]
    bag_of_words = collections.Counter(tokens)
    counts = pd.Series(bag_of_words).sort_values(ascending=False)
    counts_list.append(counts / counts.sum())
    print("\rSample {} of {} processed.".format(i + 1, len(ids_subset)), end="") # print the state

df = pd.DataFrame(counts_list) # list of series to dataframe
df = df.fillna(0) # change NaNs to 0

df
```

Sample 100 of 100 processed.

	u.s.	said	trade	japan	dlrs	exports	tariffs	imports	billion	electronics	...
0	0.039627	0.037296	0.034965	0.027972	0.013986	0.013986	0.011655	0.011655	0.011655	0.009324	...
1	0.000000	0.042254	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
2	0.000000	0.033333	0.008333	0.016667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
3	0.000000	0.027523	0.018349	0.000000	0.000000	0.009174	0.000000	0.018349	0.055046	0.000000	...
4	0.000000	0.028302	0.000000	0.000000	0.018868	0.018868	0.000000	0.000000	0.000000	0.000000	...
...	...	...	...	...	...	...	...	...	...	...	...

100 rows × 3089 columns

Corpus Processing

```
ids_subset = ids[:100] # to speed-up we consider only the first 100 elements
counts_list = []

for i, id in enumerate(ids_subset):
  sample = nltk.corpus.
  doc = nlp(sample)
  tokens = [tok.lower_
  bag_of_words = collections.Counter(tokens)
  counts = pd.Series(bag_of_words).sort_values(ascending=False)
  counts_list.append(counts / counts.sum())
  print("\rSample {} of {} processed.".format(i + 1, len(ids_subset)), end="") # print the state

df = pd.DataFrame(counts_list) # list of series to dataframe
df = df.fillna(0) # change NaNs to 0

df
```

This in inefficient!

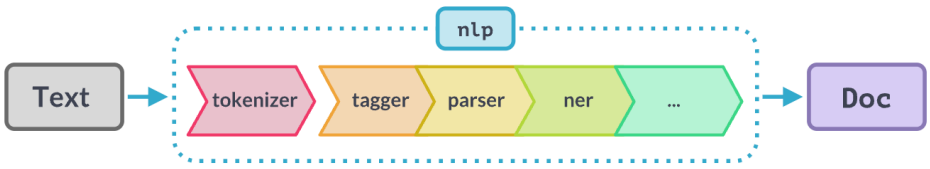
spaCy extract a lot of information from text

- **POS, lemmas, …**
- But we **only need tokenization** here!

Corpus Processing (Optimization)

When you call **`nlp`** on a text, spaCy:

- first **tokenizes** the text to produce a **Doc** object
- The **Doc** is then processed in **several steps (pipeline)**




```
# Show the current pipeline components
print (nlp.pipe_names)
```

```
['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']
```

We must remove these steps from the pipeline

Corpus Processing (Optimization)

```
ids_subset = ids # we now consoder all elements
counts_list = []

for i, id in enumerate(ids_subset):
  sample = nltk.corpus.reuters.raw(id)
  doc = nlp(sample, disable=["tok2vec", "tagger", "parser", "attribute_ruler", "lemmatizer", "ner"]) #
  tokens = [tok.lower_ for tok in doc if not tok.is_stop and not tok.is_punct and not tok.is_space]
  bag_of_words = collections.Counter(tokens)
  counts = pd.Series(bag_of_words).sort_values(ascending=False)
  counts_list.append(counts / counts.sum())
  print("\rSample {} of {} processed.".format(i + 1, len(ids_subset)), end="") # print the state

df = pd.DataFrame(counts_list) # list of series to dataframe
df = df.fillna(0) # change NaNs to 0
```

```
Sample 100 of 100 processed.
```

	u.s.	said	trade	japan	dlrs	exports	tariffs	imports	billion	electronics	...
0	0.039627	0.037296	0.034965	0.027972	0.013986	0.013986	0.011655	0.011655	0.011655	0.009324	...
1	0.000000	0.042254	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
2	0.000000	0.033333	0.008333	0.016667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
3	0.000000	0.027523	0.018349	0.000000	0.000000	0.009174	0.000000	0.018349	0.055046	0.000000	...
4	0.000000	0.028302	0.000000	0.000000	0.018868	0.018868	0.000000	0.000000	0.000000	0.000000	...
...	...	...	...	...	...	...	...	...	...	...	...

10788 rows × 49827 columns

Corpus Processing

Term-Document Matrix

- **Rows** represent **documents**
- **Columns** represent **terms** from the vocabulary
- **Elements** can be **TF**, **normalized TF**, … (other options later)

```
Sample 100 of 100 processed.
```

	u.s.	said	trade	japan	dlrs	exports	tariffs	imports	billion	electronics	...
0	0.039627	0.037296	0.034965	0.027972	0.013986	0.013986	0.011655	0.011655	0.011655	0.009324	...
1	0.000000	0.042254	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
2	0.000000	0.033333	0.008333	0.016667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
3	0.000000	0.027523	0.018349	0.000000	0.000000	0.009174	0.000000	0.018349	0.055046	0.000000	...
4	0.000000	0.028302	0.000000	0.000000	0.018868	0.018868	0.000000	0.000000	0.000000	0.000000	...
...	...	...	...	...	...	...	...	...	...	...	...
95	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
96	0.000000	0.008403	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
97	0.000000	0.000000	0.000000	0.000000	0.166667	0.000000	0.000000	0.000000	0.000000	0.000000	...
98	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
99	0.000000	0.050000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.008333	0.000000	...

100 rows × 3089 columns

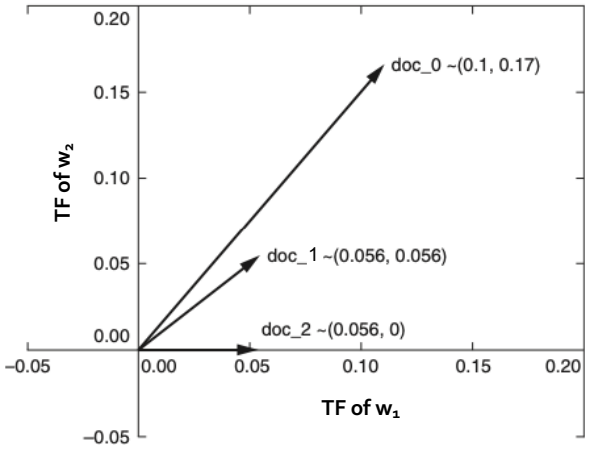
Vector Space Model

Mathematical representation of documents as **vectors** in a **multidimensional space**

- Each **dimension** of the space represents a **word**
- Built from a term-document matrix

2D Example:
with normalized TF

- We are considering a vocabulary of just **2 words** (w_1 and w_2)



Document Similarity

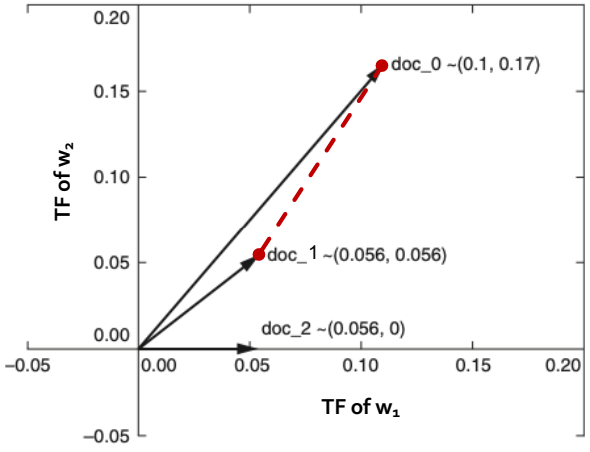
Euclidean Distance:

- Sensitive to the **magnitude** of the vectors
- The **direction** of the vectors captures the **relative importance of terms** and is **more informative than their magnitude**
- **Less commonly used** in NLP

$$\mathbf{A} = (a_1, a_2, ..., a_n)$$

$$\mathbf{B} = (b_1, b_2, ..., b_n)$$

$$d = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$



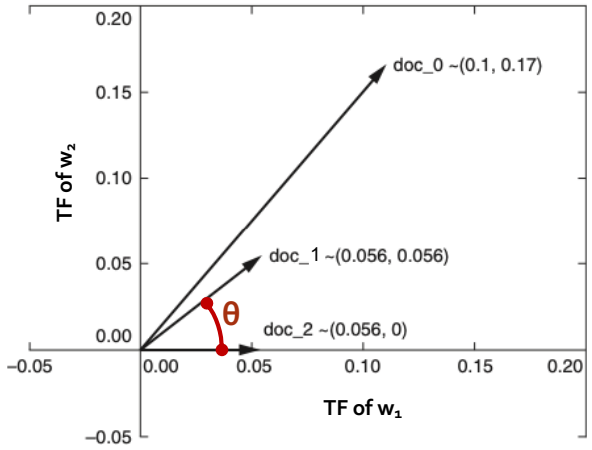
Document Similarity

Cosine Similarity:

- Measures the **cosine of the angle between two vectors**
- Focuses on the **direction** of the vectors, **ignoring their magnitude**
- Effective for **normalized text representations**
- **Widely used** in NLP

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

$$|A| = \sqrt{\sum_{i=1}^{n} a_i^2} \quad \text{(norm)}$$



Properties of Cosine Similarity

Cosine similarity is a value **between -1 and 1**

1: The vectors point in the **same direction** across all dimensions

- Documents use the **same words in similar proportion**, likely talking about the **same thing**

0: The vectors are **orthogonal** in all dimensions

- Documents **share no words**, likely discussing **completely different topics**

-1: The vectors point in **opposite directions** across all dimensions

- **Impossible with TF** (counts of words cannot be negative)
- Can happen with **other word representations** (discussed later)

Calculate Cosine Similarity

```python
import numpy as np

def sim(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    return dot_product / (norm_vec1 * norm_vec2)

# Example:
print(sim(df.iloc[0], df.iloc[1]))
print(sim(df.iloc[0], df.iloc[2]))
print(sim(df.iloc[3], df.iloc[3]))
```

```
0.14261893769917347
0.2365347461078053
1.0
```

Try also with other documents

```python
# Compare TF matrix subset (documents 0 and 1)
df.loc[[0, 1], (df.loc[0] > 0) & (df.loc[1] > 0)]
```

	said	mln	year	pct	30	government
0	0.037296	0.002331	0.009324	0.004662	0.002331	0.002331
1	0.042254	0.028169	0.014085	0.056338	0.014085	0.014085

TF-IDF

Inverse Document Frequency

TF does not consider the **uniqueness** of the word across the corpus

- **Common words** (the, is, and …) appear frequently in many documents, **contributing little to distinguishing one document from another**
- **Specific terms** relevant to the subject matter of the corpus may also be frequent but still not useful for differentiating documents
  - In a corpus about **astronomy**, terms like **planet** or **star** would be common and **non-discriminatory**

Inverse Document Frequency (IDF) addresses this by measuring **the importance of each word in relation to the entire corpus**

Inverse Document Frequency

IDF **increases** the weight of words that are **rare** across documents and **decreases** the weight of words that are **common**

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d : d \in D \text{ and } t \in d\}|}$$

$f_{t,d}$ is the *raw count* of a term in a document, i.e., the number of times that term t occurs in document d.

N: total number of documents in the corpus $N = |D|$

$|\{d \in D : t \in d\}|$: number of documents where the term t appears (i.e., $\mathrm{tf}(t,d) \neq 0$).

Note: If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the numerator $1 + N$ and denominator to $1 + |\{d \in D : t \in d\}|$.

TF-IDF

Term Frequency – Inverse Document Frequency is the product of TF and IDF

- **TF** gives more importance to the words that are **more frequent in the document**
- **IDF** gives more weightage to the words that are **less frequent in the corpus**

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

- **High TF-IDF** indicates a term that is **frequent in a document but rare in the corpus**, making it **significant** for that document
- **Low TF-IDF** Indicates a term that is **infrequent in the document or common in the corpus**, making it **less significant** for that document

TF-IDF: Example

- **Document A:** Jupiter is the largest planet
- **Document B:** Mars is the fourth planet from the sum

Words	TF (for A)	TF (for B)	IDF	TFIDF (A)	TFIDF (B)
Jupiter	1/5	0	ln(2/1) = 0.69	0.138	0
Is	1/5	1/8	ln(2/2) = 0	0	0
The	1/5	2/8	ln(2/2) = 0	0	0
largest	1/5	0	ln(2/1) = 0.69	0.138	0
Planet	1/5	1/8	ln(2/2) = 0	0.138	0
Mars	0	1/8	ln(2/1) = 0.69	0	0.086
Fourth	0	1/8	ln(2/1) = 0.69	0	0.086
From	0	1/8	ln(2/1) = 0.69	0	0.086
Sun	0	1/8	ln(2/1) = 0.69	0	0.086

Zipf's Law

Observes patterns in **word frequency distribution in natural languages**

- Named after linguist George Kingsley Zipf
- Frequency of a word is inversely proportional to its rank in a frequency table

$$f(r) = \frac{K}{r^a}$$

Where:

- *f(r)* is the frequency of the word at rank r
- *K* is a constant
- *r* is the rank of the word
- *α* determines the shape of the distribution which is approximately equals to 1

Zipf's Law

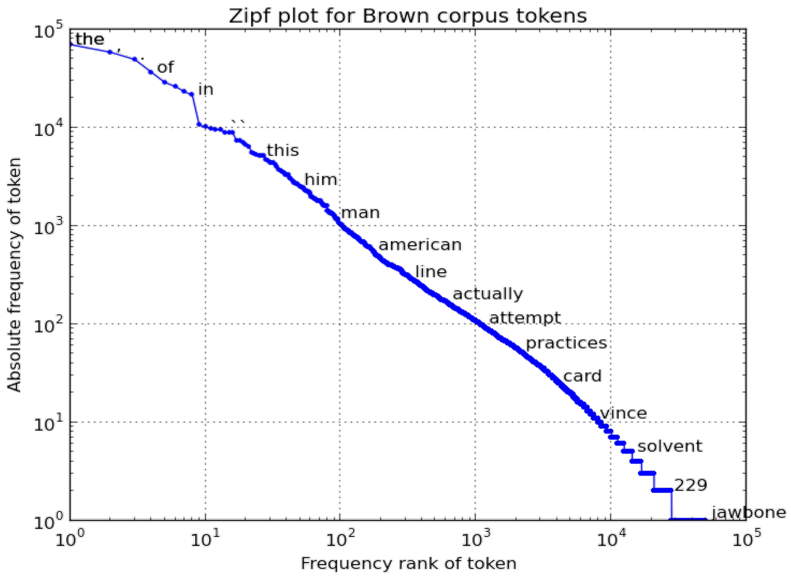
Meaning:

- The **most common word** appears approximately:
  - **Twice** as often as the second most common word
  - **Three times** as often as the third most common word
  - And so on **…**

Implications:

- A small set of highly frequent words dominate word usage
- Majority of words are relatively rare with low frequencies
- Using the **logarithm in IDF** mitigates the influence of rare words
- This results in a **more uniformly distributed TF-IDF score**

Zipf's Law: Example



The Brown corpus consists of 1 million words (500 samples of 2000+ words each) of running text of edited English prose printed in the United States during the year 1961 and it was revised and amplified in 1979.

Calculating TF-IDF

```
# Calculate IDF vector

n_docs = df.shape[0]
count_docs = (df > 0).sum()
idf = np.log10(n_docs / count_docs)

idf.sort_values(ascending=False)
```

```
asian          2.000000
exaggerated    2.000000
refute         2.000000
man            2.000000
laying         2.000000
                 ...
pct            0.494850
year           0.408935
mln            0.366532
>              0.292430
said           0.130768
Length: 3089, dtype: float64
```

```
# Calculate TF-IDF matrix

tfidf = df.mul(idf.values, axis='columns')
tfidf
```

	u.s.	said	trade	japan	dlrs	exports	tariffs	...
0	0.029511	0.004877	0.027828	0.023885	0.007114	0.013986	0.014241	...
1	0.000000	0.005525	0.000000	0.000000	0.000000	0.000000	0.000000	...
2	0.000000	0.004359	0.006632	0.014231	0.000000	0.000000	0.000000	...
3	0.000000	0.003599	0.014603	0.000000	0.000000	0.009174	0.000000	...
4	0.000000	0.003701	0.000000	0.000000	0.009597	0.018868	0.000000	...
...	...	...	...	...	...	...	...	...

Compare with TF matrix

	u.s.	said	trade	japan	dlrs	exports	tariffs	...
0	0.039627	0.037296	0.034965	0.027972	0.013986	0.013986	0.011655	...
1	0.000000	0.042254	0.000000	0.000000	0.000000	0.000000	0.000000	...
2	0.000000	0.033333	0.008333	0.016667	0.000000	0.000000	0.000000	...
3	0.000000	0.027523	0.018349	0.000000	0.000000	0.009174	0.000000	...
4	0.000000	0.028302	0.000000	0.000000	0.018868	0.018868	0.000000	...
...	...	...	...	...	...	...	...	...

Using Scikit-Learn

Scikit-learn is a popular library for machine learning

- `pip install scikit-learn`
- The `TfidfVectorizer` class does tokenization, omits punctuation, and computes the tf-idf scores all in one

```
# Calculating TF-IDF matrix with scikit-learn

# pip install scikit-learn

from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [nltk.corpus.reuters.raw(id) for id in nltk.corpus.reuters.fileids()]

vectorizer = TfidfVectorizer(min_df = 1) # min_df = 1 to consider all words
vectorizer = vectorizer.fit(corpus) # fit the vectorizer on the corpus
vectors = vectorizer.transform(corpus) # generates the TF-IDF matrix
print(vectors.todense())
```

```
[[0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.18232755 0.         ... 0.         0.         0.        ]]
```

TF-IDF Alternatives

None	$w_{ij} = f_{ij}$
TF-IDF	$w_{ij} = \log(f_{ij}) \times \log(\frac{N}{n_j})$
TF-ICF	$w_{ij} = \log(f_{ij}) \times \log(\frac{N}{f_j})$
Okapi BM25	$w_{ij} = \frac{f_{ij}}{0.5+1.5\times\frac{f_j}{\frac{f_j}{j}}+f_{ij}} \log\frac{N-n_j+0.5}{f_{ij}+0.5}$
ATC	$w_{ij} = \frac{(0.5+0.5\times\frac{f_{ij}}{max_f})\log(\frac{N}{n_j})}{\sqrt{\Sigma_{i=1}^{N}[(0.5+0.5\times\frac{f_{ij}}{max_f})\log(\frac{N}{n_j})]^2}}$
LTU	$w_{ij} = \frac{(\log(f_{ij})+1.0)\log(\frac{N}{n_j})}{0.8+0.2\times f_j\times\frac{j}{f_j}}$

TF-IDF Alternatives

MI	$w_{ij} = \log \frac{P(t_{ij}\|c_j)}{P(t_{ij})\,P(c_j)}$
PosMI	$w_{ij} = \max(0, \mathrm{MI})$
T-Test	$w_{ij} = \frac{P(t_{ij}\|c_j) - P(t_{ij})\,P(c_j)}{\sqrt{P(t_{ij})\,P(c_j)}}$
Lin98a	$w_{ij} = \frac{f_{ij} \times f}{f_i \times f_j}$
Lin98b	$w_{ij} = -1 \times \log \frac{n_j}{N}$
Gref94	$w_{ij} = \frac{\log f_{ij} + 1}{\log n_j + 1}$

Building a Search Engines

Building a Search Engine

TF-IDF matrices have been the **mainstay of information retrieval** (search) for decades

Process:

- **Tokenize** all documents and create a **TF-IDF matrix**
- Prompt the user to input a **query**
- Treat the **query as a document** and calculate its TF-IDF vector
  - **Note:** Ensure the query's vocabulary matches that of the TF-IDF matrix. Any additional words in the query are ignored
- Calculate the **cosine similarity** between the query vector and **all rows (documents)** in the TF-IDF matrix
- Identify and the document(s) with the **highest similarity** to the query

Example

```
from sklearn.metrics.pairwise import cosine_similarity

query = "OPEC oil production"

# Transform the query to a TF-IDF vector
q_vect = vectorizer.transform([query])

# Calculate cosine similarity between the query
# and all documents
similarities = cosine_similarity(q_vect, vectors).flatten()

# Get the indices of the most similar documents
# in descending order
similar_docs_indices = similarities.argsort()[::-1]

# Print the top 5 most similar documents
for i in similar_docs_indices[:5]:
  print("Document: {}, Similarity: {:.4f}\n".\
        format(i, similarities[i]))
  print(corpus[i][:500], "\n")
  print("-" * 70, "\n")

# query = "South China economy"
# query = "Credit card price"
# query = "New York stock exchange"
```

```
Document: 5252, Similarity: 0.5781

OPEC MAY HAVE TO MEET TO FIRM PRICES - ANALYSTS
  OPEC may be forced to meet before a
  scheduled June session to readdress its production cutting
  agreement if the organization wants to halt the current slide
  in oil prices, oil industry analysts said.
      "The movement to higher oil prices was never to be as easy
  as OPEC thought. They may need an emergency meeting to sort out
  the problems," said Daniel Yergin, director of Cambridge Energy
  Research Associates, CERA.
      Analysts and

----------------------------------------------------------------------

Document: 7661, Similarity: 0.5140

SAUDI OIL MINISTER SEES NO NEED TO ALTER OPEC PACT
  Saudi Arabian Oil Minister Hisham Nazer
  said OPEC's December agreement to stabilise oil prices at 18
  dlrs a barrel was being implemented satisfactorily and there
  was no immediate need to change it.
      Nazer, in an interview with Reuters and the television news
  agency Visnews, said Saudi Arabia was producing around three
  mln barrels per day (bpd) of crude oil, well below its OPEC
  quota.
      Saudi Arabia, the world's largest oil

----------------------------------------------------------------------

Document: 7736, Similarity: 0.5037

SAUDI OIL MINISTER SEES NO NEED TO ALTER PACT
  Saudi Arabian Oil Minister Hisham Nazer
  said OPEC's December agreement to stabilize oil prices at 18
  dlrs a barrel was being implemented satisfactorily and there
  was no immediate need to change it.
      Nazer, in an interview with Reuters and the television news
  agency Visnews, said Saudi Arabia was producing around three
  mln barrels per day (bpd) of crude oil, well below its OPEC
  quota.
      Saudi Arabia, the world's largest oil expo

----------------------------------------------------------------------

Document: 1010, Similarity: 0.4831

KUWAITI DAILY SAYS OPEC CREDIBILITY AT STAKE
  OPEC's credibility faces fresh scrutiny
  in coming weeks amid signs of a significant rise in supplies of
  oil to international oil markets, the Kuwait daily al-Qabas
  said.
      In an article headlined, "Gulf oil sources say Middle East
  production up 1.4 mln bpd," it warned OPEC's official prices
  could face fresh pressure from international oil companies
  seeking cheaper supplies.
      It did not say whether only OPEC or both OPEC and othe
```

More on Search Engines

Note: real search engines **do not compare the query with all indexed documents**

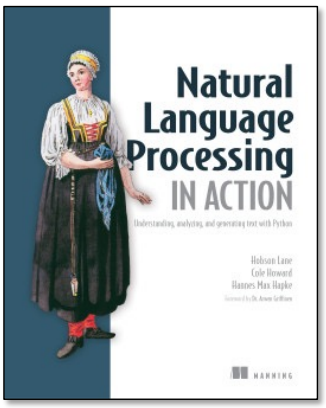
- An **inverted index** is used to map each word in the vocabulary to all documents containing that word
- The documents that contain **at least one word from the query** are extracted from the index
- The TF-IDF vector of the query is compared **only with that of the extracted documents**

Term	Document #1	Document #2
best	X	
carbonara		X
delicious		X
pasta	X	X
pesto	X	
recipe	X	X
the	X	
with	X	

References

Natural Language Processing IN ACTION
Understanding, analyzing, and generating text with Python

Chapter 3



Further Readings…

- **Scikit-Learn Documentation**
  https://scikit-learn.org/stable/user_guide.html
- **NLTK Corpora How-To**
  https://www.nltk.org/howto/corpus.html

Natural Language Processing and Large Language Models

Corso di Laurea Magistrale in Ingegneria Informatica

Lesson 3

Math with Words

Nicola Capuano and Antonio Greco

DIEM – University of Salerno