Deep Learning for Search (summer 2024)

Albert Nasybullin

tlg: @levshaazz
email: levshaazz@gmail.com

- Masters in Robotics and Computer Vision, Innopolis (2021)
- PhD student, Deep Learning and Brain-Computer Interfaces
- Teaching: Probability & Statistics, Digital Signal Processing, Machine Learning, Deep Learning (Innopolis University, SPSU)
- Machine Learning for Banking
- Nowadays, MTS. Machine Learning Engineer (NLP & Search)

The Course Aims

- Natural Language Processing
- Similarity Search
- Deep Learning
- Optimization of Neural Nets and Data storage
- Data quality and augmentation
- Retrieval Augmented Generation
- Machine Learning System Design
- Metrics of Search
- Risk approach to Machine Learning

The Course Aims

- Natural Language Processing
- Similarity Search
- Deep Learning
- Optimization of Neural Nets and Data storage
- Data quality and augmentation
- Retrieval Augmented Generation
- Machine Learning System Design
- Metrics of Search
- Risk approach to Machine Learning

By the end of the course you will have to be able to design and develop Deep Learning-based search application

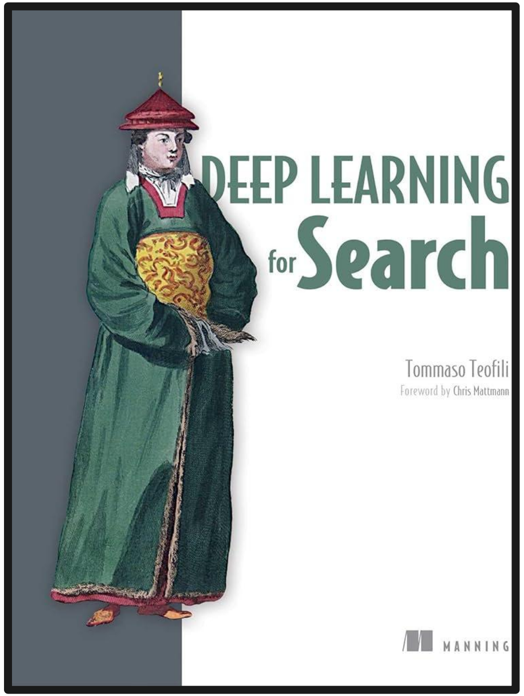
The Course Highlights

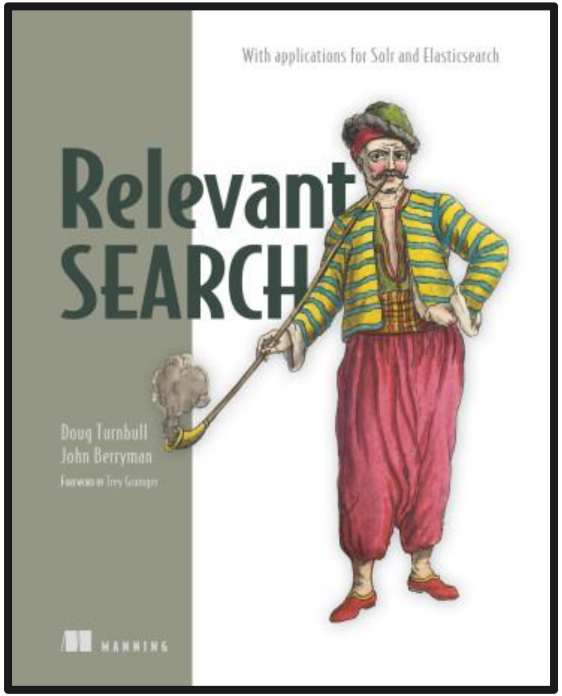
- Tuesdays, 17:40 – 20:50 (314)
- Start: June 4
- End: July 16
- “Lecture + Lab” or “Lecture + Lecture” format
- In class labs are to be graded (3 * 5%)
- Written home assignments (3 * 5%)
- The course long team projects (up to 3 people)

- Midterm exam: June 25 (20%)
- Final exam: July 16 (30%)
- Project defense: July 16 (20%)

I expect a lot of collaboration from your side!!!

Books

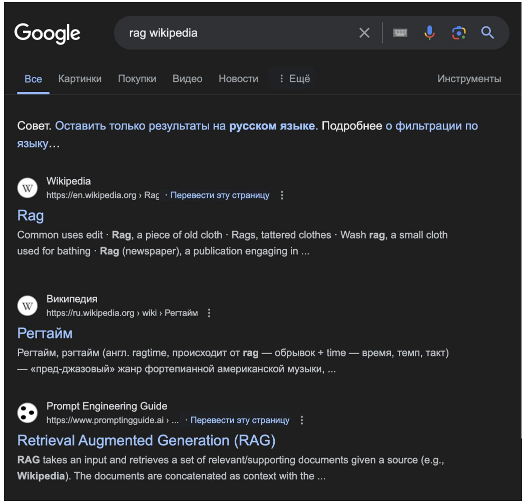

DEEP LEARNING for Search
Tommaso Teofili
Foreword by Chris Mattmann
MANNING

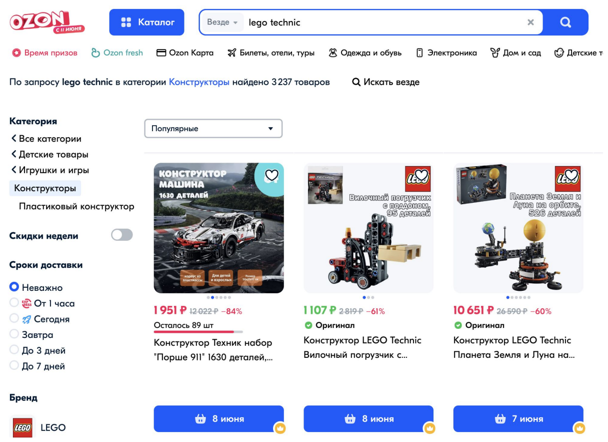

With applications for Solr and Elasticsearch
Relevant SEARCH
Doug Turnbull
John Berryman
MANNING

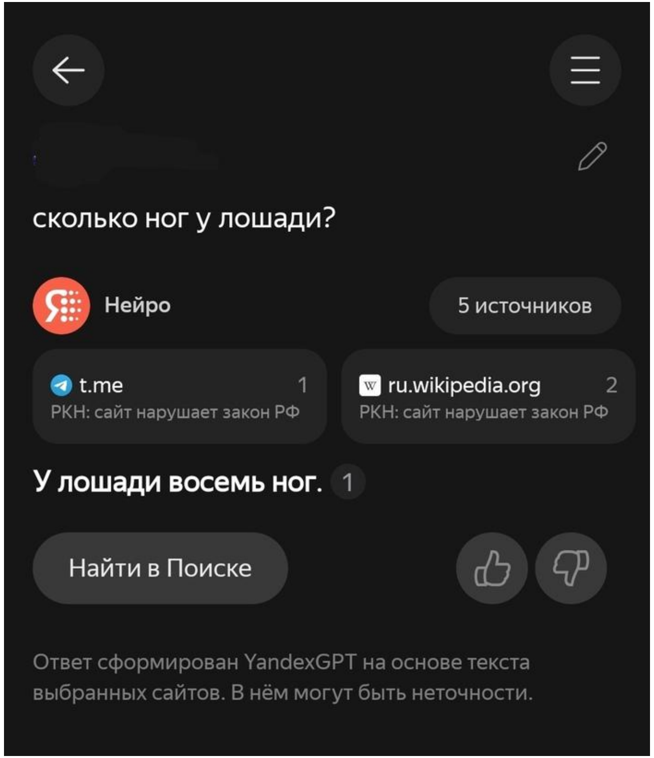
“I will make your life miserable”

Prof. Adil Khan

Introduction to Search (and a little recap)





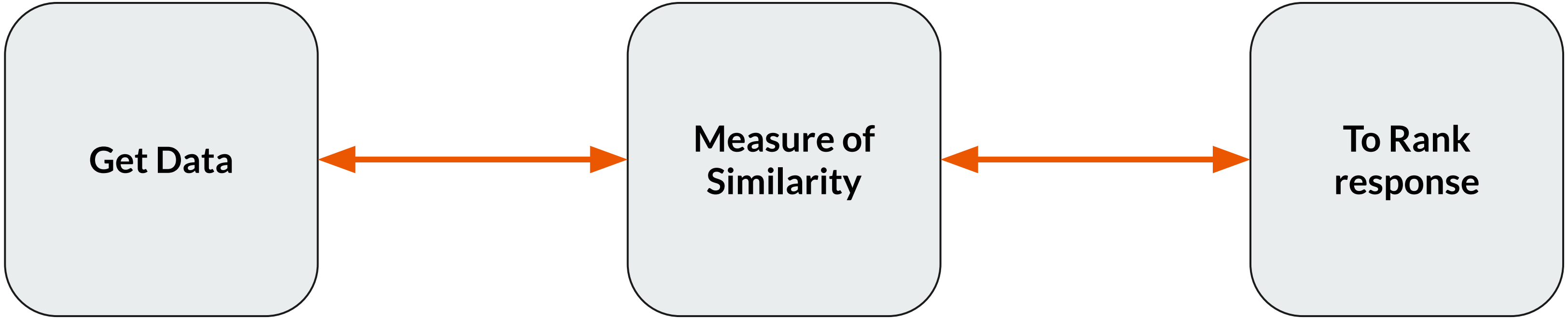


All of that is search (+ a little bit of RecSys)

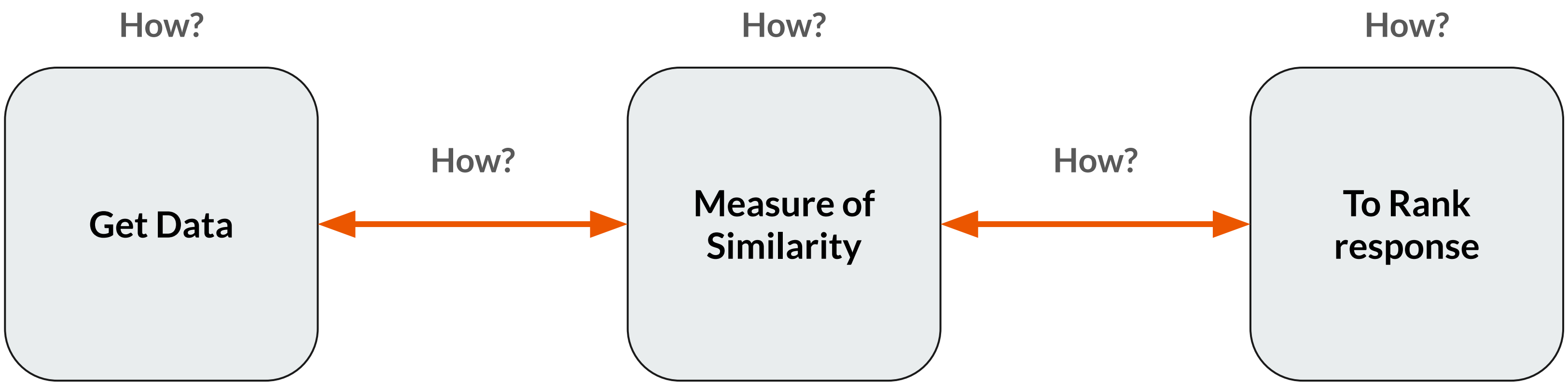
Why to make search better?

Why to make search better?

Simplest perspective



Simplest perspective



OpenSearch

elasticsearch

qdrant

Basics of Natural Language Processing

- Tokenization
- Stemming
- Lemmatization
- Ngrams
- Stop-words
- Regular expressions
- Bag of words

Tokenization

Core libs:

- **NLTK**
- TextBlob
- spaCy
- Gensim
- PyTorch
- Keras

Necklace	عُقْد
Decade	عِقد
Contract	عَقْد
Held	عَقَدَ
Complicated	عَقَّدَ
Knots	عُقّدْ

Challenges:

1 arabic word = 6 meanings in english

Tokenization

Sentence tokenization

```
sent_tokenize('Life is a matter of choices, and every choice you make makes you.')
```

```
['Life is a matter of choices, and every choice you make makes you.']
```

Word tokenization

```
word_tokenize("The sole meaning of life is to serve humanity")
```

```
['The', 'sole', 'meaning', 'of', 'life', 'is', 'to', 'serve', 'humanity']
```

Tokenization

```
import nltk
from nltk.tokenize import (word_tokenize,
                           sent_tokenize,
                           TreebankWordTokenizer,
                           wordpunct_tokenize,
                           TweetTokenizer,
                           MWETokenizer)
text="Hope, is the only thing stronger than fear! #Hope #Amal.M"
```

```
print(word_tokenize(text))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', '#', 'Hope', '#', 'Amal.M']
```

```
print(sent_tokenize(text))
```

```
['Hope, is the only thing stronger than fear!', '#Hope #Amal.M']
```

Tokenization

Punctuation tokenization

```
print(wordpunct_tokenize(text))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', '#', 'Hope', '#', 'Amal', '.', 'M']
```

Tokenization

Treebank Word tokenization

```
text="What you don't want to be done to yourself, don't do to others..."
tokenizer= TreebankWordTokenizer()
print(tokenizer.tokenize(text))
```

```
['What', 'you', 'do', "n't", 'want', 'to', 'be', 'done', 'to', 'yourself', ',', 'do', "n't", 'do', 'to', 'others', '...']
```

Tokenization

[Tweet tokenization]()

```
tweet= "Don't take cryptocurrency advice from people on Twitter 😅👌"
tokenizer = TweetTokenizer()
print(tokenizer.tokenize(tweet))
```

```
["Don't", 'take', 'cryptocurrency', 'advice', 'from', 'people', 'on', 'Twitter', '😅', '👌']
```

Tokenization

MWET tokenization (multiple word expressions)

```
text="Hope, is the only thing stronger than fear! Hunger Games #Hope"
tokenizer = MWETokenizer()
print(tokenizer.tokenize(word_tokenize(text)))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', 'Hunger', 'Games', '#', 'Hope']
```

```
text="Hope, is the only thing stronger than fear! Hunger Games #Hope"
tokenizer = MWETokenizer()
tokenizer.add_mwe(('Hunger', 'Games'))
print(tokenizer.tokenize(word_tokenize(text)))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', 'Hunger_Games', '#', 'Hope']
```

Stemming & Lemmatization

Both are normalization techniques in Natural Language Processing

Stemming:

a crude heuristic process that cuts off the “extra” from the root of words, often resulting in the loss of word-forming suffixes.

Lemmatization

a more subtle process that uses dictionary and morphological analysis to eventually bring a word to its canonical form, the lemma.

Stemming & Lemmatization

```python
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import wordnet

def compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word, pos):
    """
    Print the results of stemmind and lemmitization using the passed stemmer, lemmatizer,
    """
    print("Stemmer:", stemmer.stem(word))
    print("Lemmatizer:", lemmatizer.lemmatize(word, pos))
    print()

lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word = "seen", pos = wordnet.VERB)
compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word = "drove", pos = wordnet.VERB)
```

```
Stemmer: seen
Lemmatizer: see

Stemmer: drove
Lemmatizer: drive
```

Stemming & Lemmatization

```python
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import wordnet

def compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word, pos):
    """
    Print the results of stemmind and lemmitization using the passed stemmer, lemmatizer,
    """
    print("Stemmer:", stemmer.stem(word))
    print("Lemmatizer:", lemmatizer.lemmatize(word, pos))
    print()

lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word = "seen", pos = wordnet.VERB)
compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word = "drove", pos = wordnet.VERB)
```

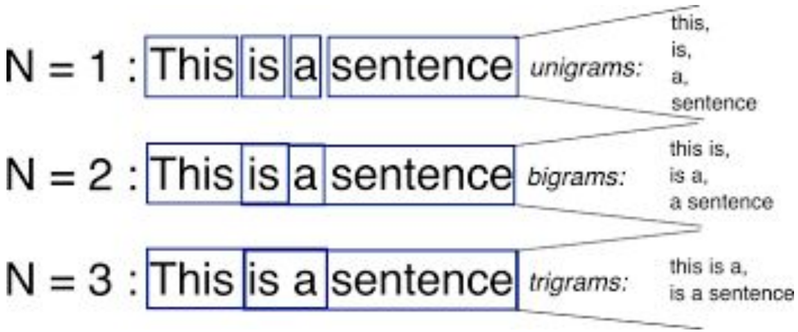
“seen” and “drove”

```
Stemmer: seen
Lemmatizer: see

Stemmer: drove
Lemmatizer: drive
```

Ngrams

Ngrams:

contiguous sequences of n words. For example "riverbank"," The three musketeers" etc.If the number of words is two, it is called bigram. For 3 words it is called a trigram and so on.



Ngrams

```
from nltk.util import ngrams
list(ngrams(['I' ,'went','to','the','river','bank'],2))
```

✓ 0.0s

```
[('I', 'went'),
 ('went', 'to'),
 ('to', 'the'),
 ('the', 'river'),
 ('river', 'bank')]
```

Stop-words

Stop-words:

```python
import nltk
from nltk.corpus import stopwords

stop_words = stopwords.words("english")
stop_words.append('nonsens')
stop_words = set(stop_words)
sentence = "Backgammon is one of the oldest and nonsens known board games."

words = nltk.word_tokenize(sentence)
without_stop_words = [word for word in words if not word in stop_words]
print(without_stop_words)
```

✓ 1.1s

```
['Backgammon', 'one', 'oldest', 'known', 'board', 'games', '.']
```

Stop-words

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
```

```
 'she',
 "she's",
 'her',
 'hers',
 'herself',
 'it',
 "it's",
 'its',
 'itself',
 'they',
 'them',
 'their',
 'theirs',
 'themselves',
 'what',
 'which',
 'who',
 'whom',
 'this',
 'that',
```

```
 "isn't",
 'ma',
 'mightn',
 "mightn't",
 'mustn',
 "mustn't",
 'needn',
 "needn't",
 'shan',
 "shan't",
 'shouldn',
 "shouldn't",
 'wasn',
 "wasn't",
 'weren',
 "weren't",
 'won',
 "won't",
 'wouldn',
 "wouldn't"]
```

Regular expressions

<u>**RegEx:**</u>

(regular, regexp, regex) is a sequence of characters that defines a search pattern

- . - any character except line feed;
- \w - one character;
- \d - one digit;
- \s - one space;
- \W - one non-character;
- \D - one non-digit;
- \S - one non-space;
- [abc] - finds any of the specified characters match any of a, b, or c;
- [^abc] - finds any character other than the specified ones;
- [a-g] - finds a character between a and g.

Regular expressions

The re module in Python represents regular expression operations. We can use the re.sub function to replace anything that fits the search pattern with the specified string. This is how to replace all non-words with spaces

```
import re
sentence = "The development of snowboarding was inspired by skateboarding, sledding, surfing and skiing."
pattern = r"[^\w]"
print(re.sub(pattern, " ", sentence))
```

✓ 0.0s

What will "print" on line 4 produce?

Regular expressions

The re module in Python represents regular expression operations. We can use the re.sub function to replace anything that fits the search pattern with the specified string. This is how to replace all non-words with spaces

```
import re
sentence = "The development of snowboarding was inspired by skateboarding, sledding, surfing and skiing."
pattern = r"[^\w]"
print(re.sub(pattern, " ", sentence))
```

✓ 0.0s

```
The development of snowboarding was inspired by skateboarding  sledding  surfing and skiing 
```

Bag of words

Bag of words:

- Machine learning algorithms cannot deal directly with raw text
- necessary to convert text into sets of numbers (vectors), aka “extract features”
- simple feature extraction technique for text mining
- describes the occurrences of each word in the text.

To use the model, we need to:

- Define a vocabulary of known words (tokens).
- Select the degree of occurrence of known words.

Intuition tells us that **similar documents** have **similar content**

Bag of words

```
initial_text = """I like this movie, it's funny.
I hate this movie.
This was awesome! I like it.
Nice one. I love it."""
```

```
initial_list = initial_text.split('\n')
initial_list
```

✓ 0.0s

```
["I like this movie, it's funny.",
 'I hate this movie.',
 'This was awesome! I like it.',
 'Nice one. I love it.']
```

Bag of words

```
["I like this movie, it's funny.",
 'I hate this movie.',
 'This was awesome! I like it.',
 'Nice one. I love it.']
```

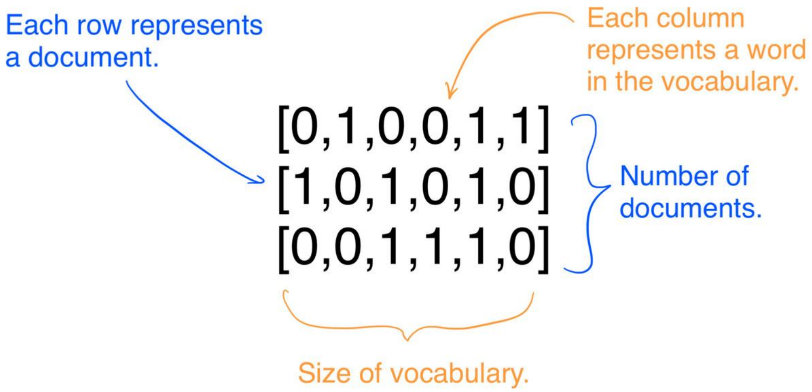
```python
# Import the libraries we need
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Step 2. Design the Vocabulary
# The default token pattern removes tokens of a single character
# That's why we don't have the "I" and "s" tokens in the output
count_vectorizer = CountVectorizer()

# Step 3. Create the Bag-of-Words Model
bag_of_words = count_vectorizer.fit_transform(initial_list)

# Show the Bag-of-Words Model as a pandas DataFrame
feature_names = count_vectorizer.get_feature_names_out()
pd.DataFrame(bag_of_words.toarray(), columns = feature_names)
```

✓ 0.7s

	awesome	funny	hate	it	like	love	movie	nice	one	this	was
0	0	1	0	1	1	0	1	0	0	1	0
1	0	0	1	0	0	0	1	0	0	1	0
2	1	0	0	1	1	0	0	0	0	1	1
3	0	0	0	1	0	1	0	1	1	0	0

Bag of words

	the	red	dog	cat	eats	food
1. the red dog	1	1	1	0	0	0
2. cat eats dog	0	0	1	1	1	0
3. dog eats food	0	0	1	0	1	1
4. red cat eats	0	1	0	1	1	0



Bag of words

Bag of words:

Sometimes “Bag of words” produces sparse matrices. **How to fix?**

Bag of words

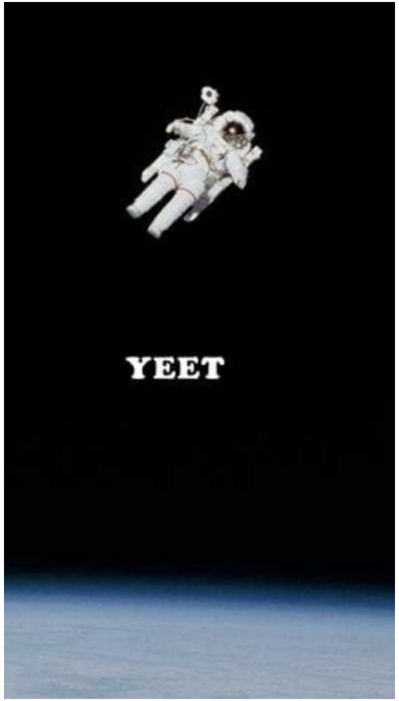
Bag of words:

Sometimes “Bag of words” produces sparse matrices. **How to fix?**

- ignore word case
- ignore punctuation
- throw out stop words
- bring words back to their basic forms (lemmatization and stemming)
- correct misspelled words

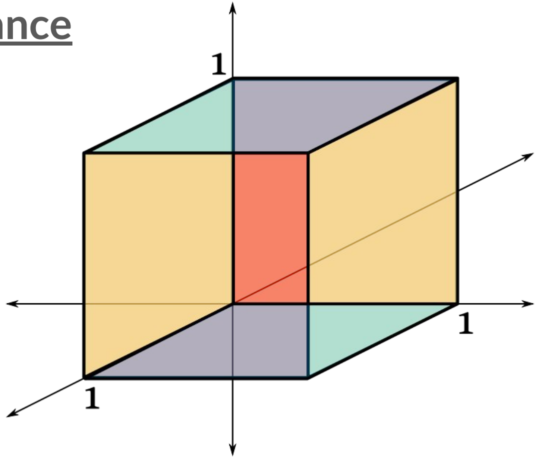
Bag of words & Similarity

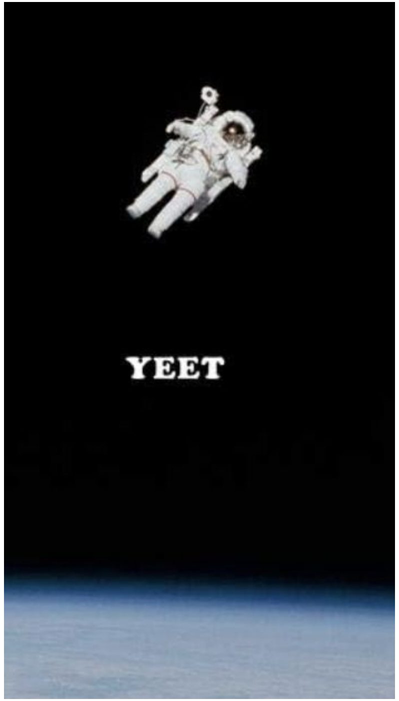
- documents are not a sequence of words
- documents are points in a **multi-dimensional vector space**
- each vector has the **same length**
- now we can measure **the distance**



Bag of words & Similarity

- documents are not a sequence of words
- documents are points in a **multi-dimensional vector space**
- each vector has the **same length**
- now we can measure **the distance**

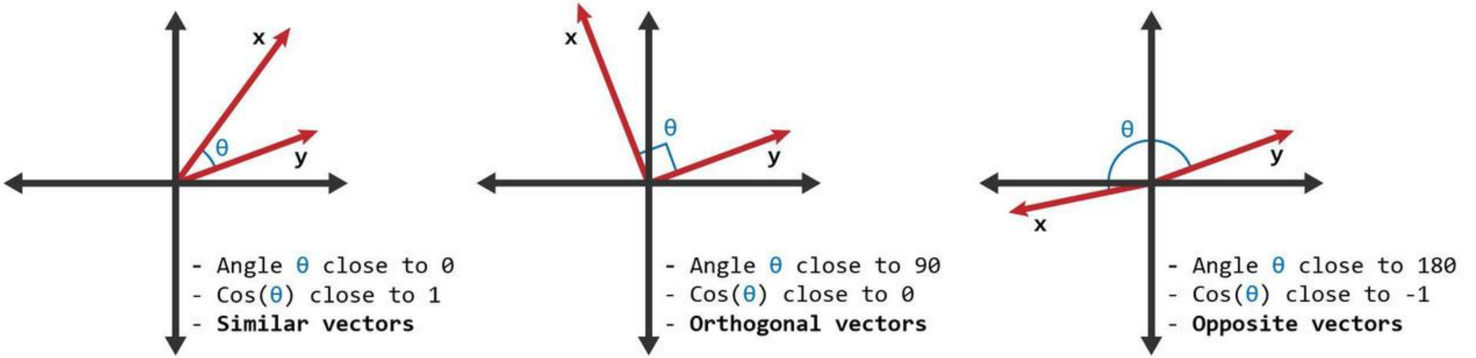




Bag of words & Similarity

$$\cos(\theta) = \frac{a \cdot b}{\|a\| \, \|b\|}$$

- cosine similarity
- cosine of the angle between any two points (more precisely their vectors starting from the origin)
- closer the score 1, the smaller the angle between the vectors and the more similar the documents are



Bag of words & Similarity

```
import numpy as np

def cosine_sim(a,b):
    return np.dot(a,b) / (np.linalg.norm(a)*np.linalg.norm(b))
```

Bag of words & Similarity

```
bow_array = bag_of_words.toarray()
bow_array
```

0.0s

```
array([[0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0],
       [0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0],
       [1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1],
       [0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0]])
```

Bag of words & Similarity

```python
print(initial_list[0])
print(initial_list[1])
print(f'Similarity score: {cosine_sim(bow_array[0], bow_array[1]):.3f}')
```

✓ 0.0s

```
I like this movie, it's funny.
I hate this movie.
Similarity score: 0.516
```

Home Assignment 1

- Deadline: June 10, 23:59
- Task:
  - Read the paper "Distributed Representations of Words and Phrases and their Compositionality" [link]
  - Summarize main concepts in essay (2 pages min)
  - Take example at slides 34-44 (Bag of Words & Similarity) and repeat it with your own hands (written form, no code)
  - Submit

Questions?

Brake (20 minutes)