Natural Language Processing with Python

PIKAKSHI MANCHANDA

POST DOCTORAL RESEARCH FELLOW, VISTA AR

@itsPikakshi

p.Manchanda@Exeter.ac.uk

OUTLINE

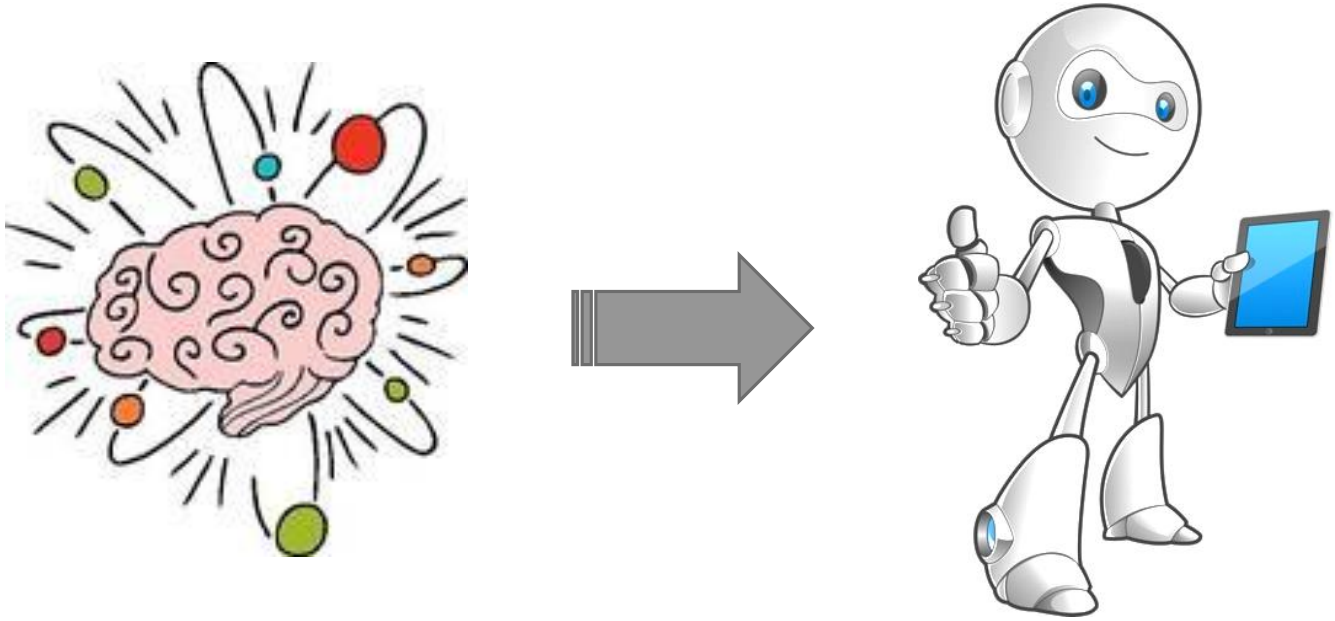
- Natural Languages
- What is Natural Language Processing?
- Unstructured Data
  - Why is NLP important?
  - Defining Text Analytics
- Libraries in Python
- The NLP Pipeline: hands-on using Python libraries
- Advanced Text Processing
  - Information Extraction
  - Vector Space Model: TFIDF
- Text Similarity Measures
- Why is NLP difficult?
- Significance of NLP: Modern Applications

NATURAL LANGUAGES

- Dream
- Plan
- Make decisions
- Communicate
- Conceive the world around us – in natural languages --words

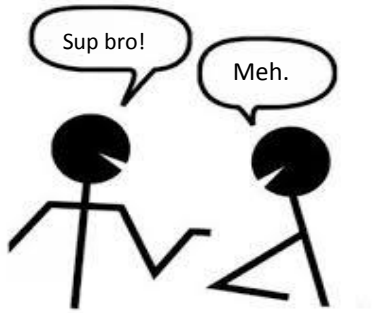
The Key...

The key to understanding how a human brain works (to understand, interpret and communicate) is by understanding how language works!



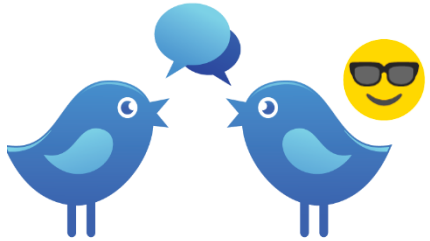
Cue: **Natural Language Processing**

WHAT IS NLP?



Speech communication

Speech-to-text communication



Text communication

- Natural Language Processing: teaching computers to understand (and generate) natural language for a range of applications by drawing insights.
- An umbrella term that describes the ability to break down the unstructured language to understand, process and generate a comprehensible unstructured output for humans.
- Draws from many disciplines including Computer Science, Computational Linguistics, Mathematics, Statistics, Artificial Intelligence, Psychology, …

NLU

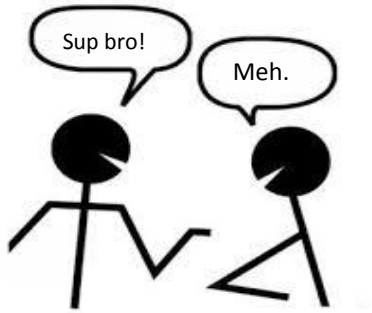
Express → Understand → Computer → Generate → Output

Understanding the (unstructured) language – human generated content.

- Syntax – understanding grammar of text
- Semantics – understanding meaning of text
- Pragmatics – understanding what the text is trying to achieve

- ✓ Profanity detection (eg, does a post contain any profanity?)
- ✓ Sentiment detection (eg, did a customer provide a positive or negative review)
- ✓ Topic identification (eg, what is this email about?)
- ✓ Entity detection (eg, what locations are referenced in this text message?)

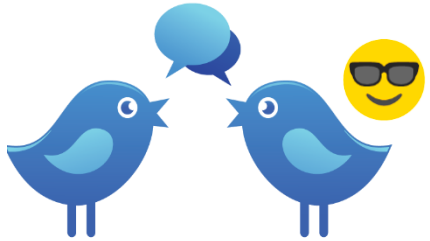
WHAT IS NLP?



Speech communication

Ok Google! What's the local weather?

Speech-to-text communication



Text communication

- Natural Language Processing: teaching computers to understand (and generate) natural language for a range of applications by drawing insights.
- An umbrella term that describes the ability to break down the unstructured language to understand, process and generate a comprehensible unstructured output for humans.
- Draws from many disciplines including Computer Science, Computational Linguistics, Mathematics, Statistics, Artificial Intelligence, Psychology, …

Express → Understand → Computer → Generate → Output

NLG

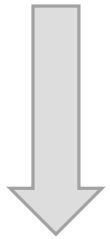
Generating natural language after analysis and interpretation of structured data.

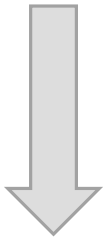
- ✓ Written analysis for business intelligence and analytics platforms
- ✓ Personalised customer communications via email, chatbots etc

UNSTRUCTURED DATA

- Un-understandable by machines.
- According to a study conducted in 2015, 80% of the world's data is unstructured.
- Customer feedback, medical records, emails and text messages, social media posts and comments, image/audio/video exchanges, discussion forums, press releases, literary texts…

➢ Huge portion: human-generated content (using natural languages) : primarily text





NATURAL LANGUAGE PROCESSING TEXT ANALYTICS

Why Is NLP Important?

TEXT ANALYTICS

- Process of examining unstructured text data to extract useful information (key concepts and themes) to uncover meaningful insights.
- Helpful in tasks such as understanding customer experience, product reviews, sentiment analysis, document summarization and so on which aid in decision making processes.
- ***Customer experience analysis***: analysing textual feedback from customers to find useful insights in the form of patterns and topics of interest about which, for instance, the customers might have expressed an opinion!

Examples of unstructured text and information that can be extracted

clairejobby
Braintree, United Kingdom
20 6

Reviewed 2 weeks ago

Sumptuous food in a relaxed atmosphere

We had a fabulous evening meal with fresh vegetables, gorgeous tasting fish and steak. Which was followed by a trio of lemon infused deserts. Heavenly!

Date of visit: January 2019

Ask clairejobby about The Dining Room Restaurant at Washingborough Hall Hotel

scothernite
Lincoln, United Kingdom
353 99

Reviewed 10 December 2018

Christmas Dinner

We were a group of twenty two, and the service was very good, The food was good too, except that my duck terrine was a little dry,

Date of visit: December 2018

Value

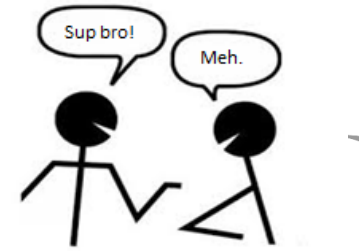
Service
Food

Ask scothernite about The Dining Room Restaurant at Washingborough Hall Hotel

Libraries available in Python

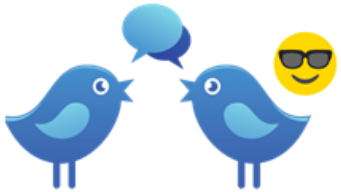
- Natural Language Toolkit (NLTK): Python Library for all NLP techniques.
- TextBlob – Easy to use NLP tools API, built on top of NLTK and Pattern.
- spaCy – Industrial strength NLP with Python and Cython.
- Gensim – Python library specialising in vector space modelling and topic modelling.
- Stanford Core NLP – A suite of NLP tools that provide POS tagging, entity recognition, sentiment analysis and many other services.
- Apache OpenNLP: Machine Learning toolkit that provides tokenizers, sentence segmentation, POS tagging, and more.
- scikit-learn: Machine learning in Python

The NLP Pipeline



Speech communication

Speech-to-text communication


Text communication

Express/Data Collection → Preprocessing → Lexical Analysis: Text Representation: understanding words → Syntactic Analysis/ Parsing: Understanding language: grammar → Semantic Analysis: Understanding meaning → Pragmatic Analysis: Understanding context and source of text

- The first step in the pipeline depends on the source of data and the application of the system.
  - Audio data → translated to text using HMM models (Speech to Text)

- Different NLP systems may use different techniques, but overall, data processing steps are fairly similar.

Text Preprocessing

Noise Removal

- Stopwords, URLs, special characters, punctuation
- Case normalization

```
set(stopwords.words('english'))
s.translate(str.maketrans("", "", string.punctuation))
word =  word.lower()
```

Segmentation & Tokenization

- Paragraph/sentence segmentation
- Tokenization

```
tokenize.sent_tokenize(line)
word_tokenize(line)
```

Normalization

- Stemming
- Lemmatization

```
porter = PorterStemmer()
lem = WordNetLemmatizer()
```

Available on github.com/Pikakshi/NLP_Introduction

Performing text pre-processing using Python NLTK (Natural Language Toolkit)

1. Stop Word Removal: NLTK has a list of stopwords in 16 different languages.

stopwords: Noisy words in a language, i.e., words with higher frequency, that a search engine has been programmed to ignore.

```
import nltk
nltk.download('stopwords')
```

```
#print stopwords in English language
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
print(stop_words)
```

Stop word removal from a file

```
from nltk.corpus import stopwords
#word_tokenize accepts a string as an input, not a file.
from nltk.tokenize import word_tokenize
#selecting stopwords language
stop_words = set(stopwords.words('english'))
#read from a file as a stream
filename = open('preprocessing_data.txt')
line = filename.read()
words = line.split()
for r in words:
    if r not in stop_words:
        outputfile = open('nostopwords_data.txt','a')
        outputfile.write(" "+r)
        outputfile.close()
```

Stop word removal from a string

```
example_line = 'something... is! wrong() with.,; this :: sentence.'
stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example_line)

filtered_sentence = [w for w in word_tokens if not w in stop_words]

filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
print(filtered_sentence)
```

```
['something', '...', '!', 'wrong', '(', ')', 'with.', ',', ';', ':', ':', 'sentence', '.']
```

2. Converting text to lower case

```
#read from file as a stream
with open('preprocessing_data.txt', 'r') as fileinput:
    for line in fileinput:
        #convert to lower case
        line = line.lower()
        #write to file
        with open('lowercase_data.txt','a') as outputFile:
            outputFile.writelines(line)
```

3. Removal of Punctuations, URLs and special characters

```
In [29]: import string
         tr = str.maketrans("", "", string.punctuation)
         s = "{{Here is more stuff in single curly braces.}}"
         s.translate(tr)
```

```
Out[29]: 'Here is more stuff in single curly braces'
```

```
import string
tr = str.maketrans("", "", string.punctuation)
with open('preprocessing_data.txt', 'r') as fileinput:
    for line in fileinput:
        stripped = line.translate(tr)
        print(stripped)
```

4. Tokenization

Sentence segmentation

```
from nltk import tokenize
with open('preprocessing_data.txt', 'r') as fileinput:
    for line in fileinput:
        tokenized = tokenize.sent_tokenize(line)
        print(tokenized)
```

Text tokenization

```
from nltk.tokenize import word_tokenize
with open('preprocessing_data.txt', 'r') as fileinput:
    for line in fileinput:
        tokens = word_tokenize(line)
        print(tokens)
```

5. Stemming – reducing each word to its root or base.

- A rudimentary rule-based process of stripping the suffixes (“ing”, “ly”, “es”, “s” etc) from a word.
- For example “*fishing*,” “*fished*,” “*fisher*” all reduce to the stem “*fish*.”
- “studies” → “studi”, “studying” → “study”
- Most common algorithm for stemming English is *Porter's algorithm.*

```
filename = 'preprocessing_data.txt'
file = open(filename, 'rt')
text = file.read()
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()
stemmed = [porter.stem(word) for word in tokens]
print(stemmed[:100])
```

6. Lemmatization - usually refers to the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word known as the *lemma*.

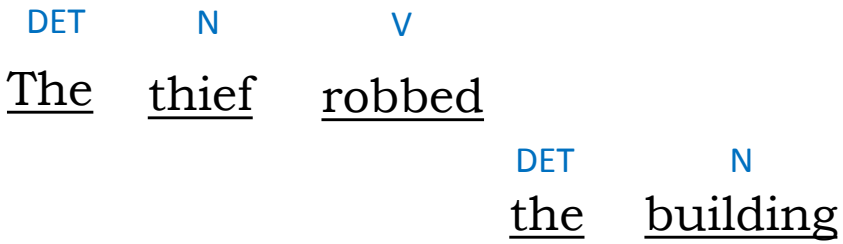
- “studies, studying” → “study”

```
filename = 'preprocessing_data.txt'
file = open(filename, 'rt')
text = file.read()
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()
lemmatized = [lem.lemmatize(word,'v') for word in tokens]
print(lemmatized)
```

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

```python
from nltk.tokenize import word_tokenize
with open('preprocessing_data.txt', 'r') as fileinput:
    for line in fileinput:
        tokens = word_tokenize(line)
        tags = nltk.pos_tag(tokens)
        print(tags)
```

```
[('The', 'DT'),
 ('thief', 'NN'),
 ('robbed', 'VBD'),
 ('the', 'DT'),
 ('building', 'NN')]
```



The thief robbed the building.

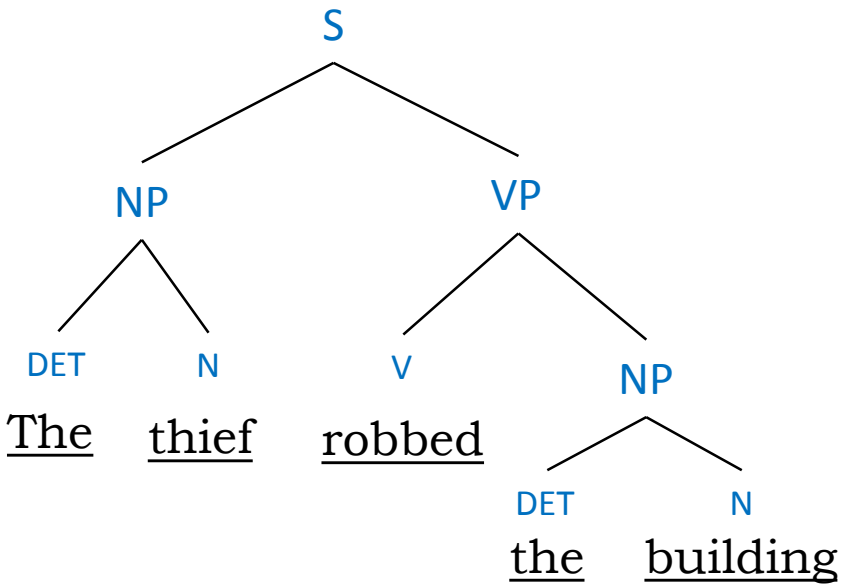
N: Noun
V: Verb
DET: Determiner
NP: Noun Phrase
VP: Verb Phrase
S: Sentence

1. Lexical Analysis: Use of part-of-speech tagging: Identifying words (i.e., lexical units) within text

NLP Pipeline contd..

```
from nltk.tokenize import word_tokenize
with open('preprocessing_data.txt', 'r') as fileinput:
    for line in fileinput:
        tokens = word_tokenize(line)
        tags = nltk.pos_tag(tokens)
        print(tags)
```

```
[('The', 'DT'),
 ('thief', 'NN'),
 ('robbed', 'VBD'),
 ('the', 'DT'),
 ('building', 'NN')]
```



The thief robbed the building.

N: Noun
V: Verb
DET: Determiner
NP: Noun Phrase
VP: Verb Phrase
S: Sentence

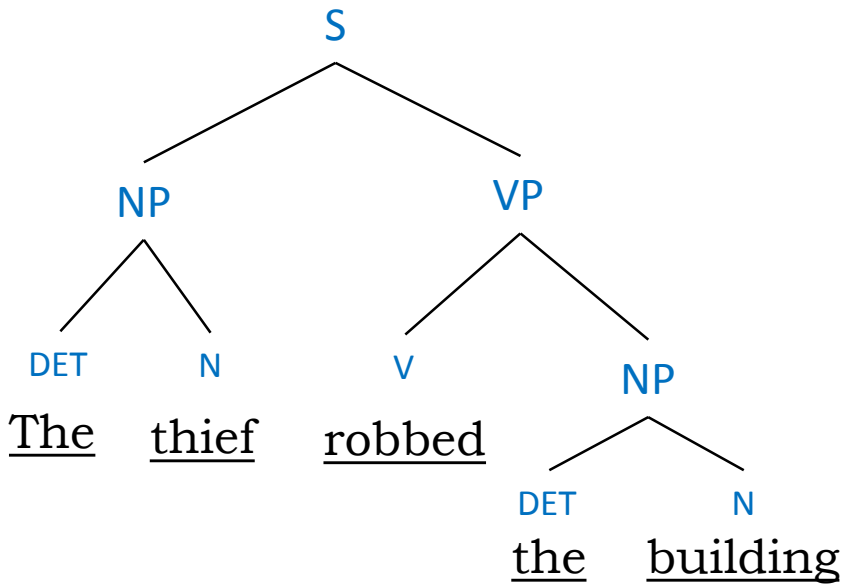
2. Syntactic Analysis: Parsing (Phrase Structure): understanding the grammar and word associations among lexical units by identifying phrases and their recursive structure.

1. Lexical Analysis: Use of part-of-speech tagging: Identifying words (i.e., lexical units) within text

NLP Pipeline contd..

```
from nltk.tokenize import word_tokenize
with open('preprocessing_data.txt', 'r') as fileinput:
    for line in fileinput:
        tokens = word_tokenize(line)
        tags = nltk.pos_tag(tokens)
        print(tags)
```

```
[('The', 'DT'),
 ('thief', 'NN'),
 ('robbed', 'VBD'),
 ('the', 'DT'),
 ('building', 'NN')]
```



The thief robbed the building.

N: Noun
V: Verb
DET: Determiner
NP: Noun Phrase
VP: Verb Phrase
S: Sentence

NLP Pipeline contd..

3. Semantic Analysis: understanding meaning of words, obtaining *logical forms* (context-independent meaning of a sentence).
Techniques include: NER, WSD, NLG

Building → verb, noun.

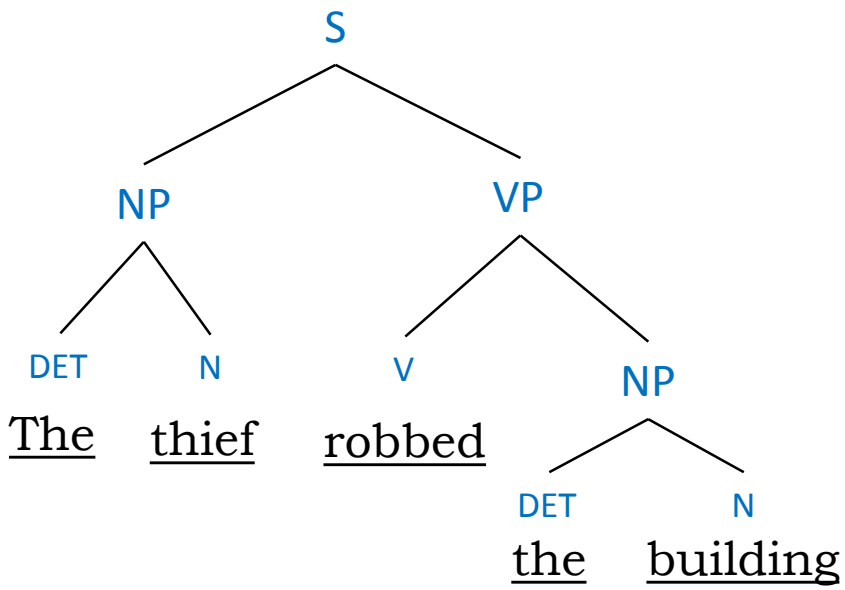
- '*thief*' and '*person*' are semantically related.
- '*thief*' and '*robbing*' are more closely related than '*thief*' and '*eating*'.
- helpful for WSD!

2. Syntactic Analysis: Parsing (Phrase Structure): understanding the grammar and word associations among lexical units by identifying phrases and their recursive structure.

1. Lexical Analysis: Use of part-of-speech tagging: Identifying words (i.e., lexical units) within text

```
from nltk.tokenize import word_tokenize
with open('preprocessing_data.txt', 'r') as fileinput:
    for line in fileinput:
        tokens = word_tokenize(line)
        tags = nltk.pos_tag(tokens)
        print(tags)
```

```
[('The', 'DT'),
 ('thief', 'NN'),
 ('robbed', 'VBD'),
 ('the', 'DT'),
 ('building', 'NN')]
```



The thief robbed the apartment.

4. Pragmatic Analysis: using context (knowledge of previous sentences, knowledge of the speaker,..) and world knowledge to elaborate the meaning of a sentence.

A person maybe saying that 'The thief robbed the apartment' because the building was unsafe, or needs to be protected.

N: Noun
V: Verb
DET: Determiner
NP: Noun Phrase
VP: Verb Phrase
S: Sentence

NLP Pipeline contd..

3. Semantic Analysis: understanding meaning of words, obtaining *logical forms* (context-independent meaning of a sentence).
Techniques include: NER, WSD, NLG

Building → verb, noun.

- '*thief*' and '*person*' are semantically related.
- '*thief*' and '*robbing*' are more closely related than '*thief*' and '*eating*'.
- helpful for WSD!

2. Syntactic Analysis: Parsing (Phrase Structure): understanding the grammar and word associations among lexical units by identifying phrases and their recursive structure.

1. Lexical Analysis: Use of part-of-speech tagging: Identifying words (i.e., lexical units) within text

Advanced Text Processing

- Information Extraction
  - Entity Extraction
  - Event Extraction
  - Relation Extraction
- Vector space models
  - Bag of Words
  - TF-IDF Model

INFORMATION EXTRACTION

Extracting relevant, structured information (named entities, events, relations,..) from unstructured data.

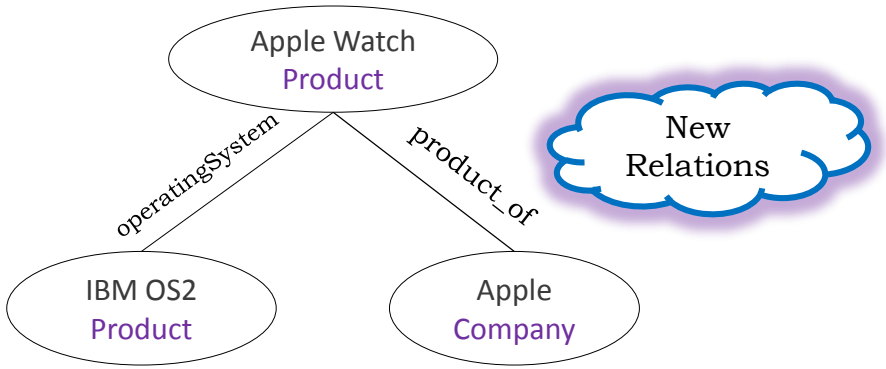
IE TASKS

- **Named Entity Identification:** Task of recognizing named entities in a piece of text
  - Named Entities: text fragments that refer to entities in the real world (proper nouns..)
- **Named Entity Classification:** Classifying recognized named entities into entity types such as PERSON, LOCATION, ORGANIZATION…
- **Relation Extraction:** Task of identifying and extracting semantic relationships between named entities from a piece of text.
- **Event Extraction:** Task of identifying and extracting events (an action of significance – *{actor,event phrase,time period,type of event}*), from a piece of text.

WHY THE NEED??

- ❖ Fresh
- ❖ Incomplete Knowledge Bases
- ❖ Real-time info
- ❖ Bridging the gap

Stunning announcement from Apple: an Apple Watch version of IBM OS2 apple.com/watchos-2/techland.time….



```python
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

doc = nlp("Next week I'll be in Madrid.") #to obtain IOB style tagging of sentences with the entity types
iob_tagged = [
    (
        token.text,
        token.tag_,
        "{0}-{1}".format(token.ent_iob_, token.ent_type_) if token.ent_iob_ != 'O' else token.ent_iob_
    ) for token in doc
]

print(iob_tagged)
```

- Use of spaCy.
- List of Entity types available here.

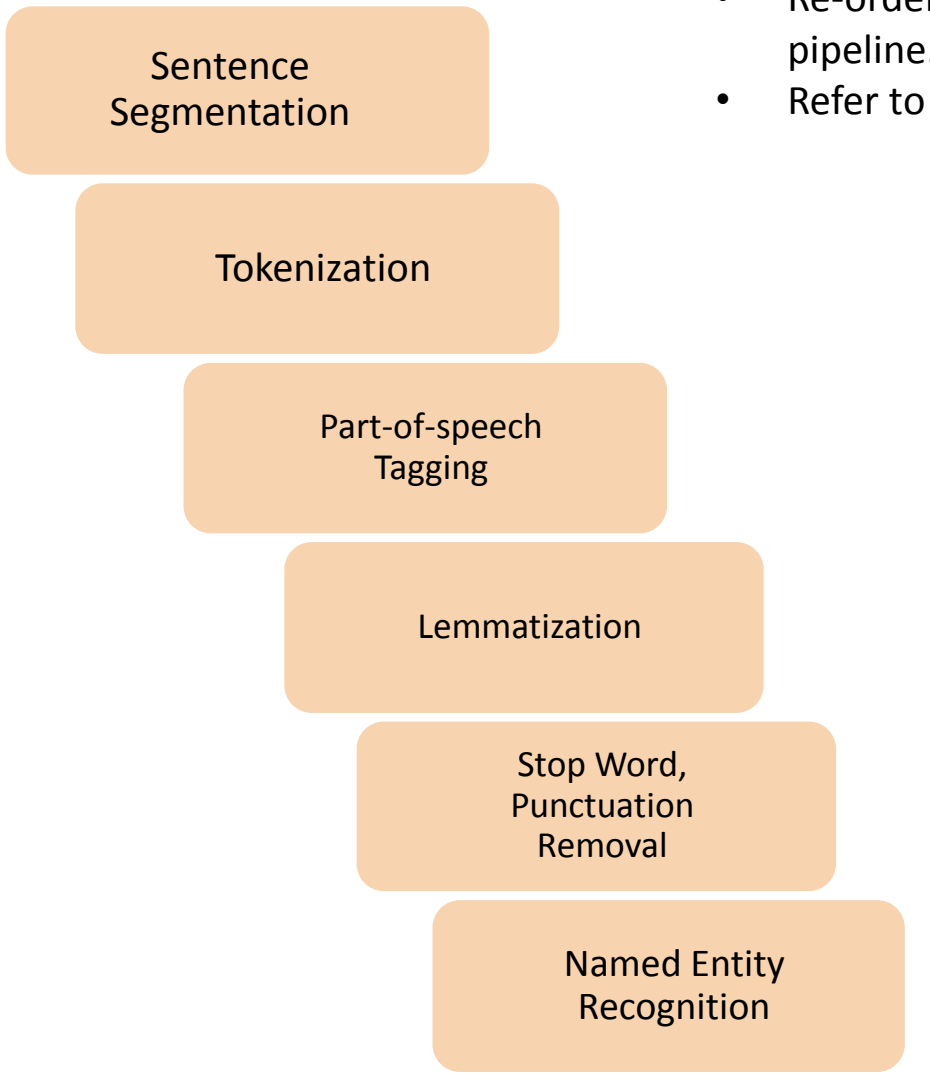
```python
#Obtaining frequencies of entity types
from spacy import displacy
from collections import Counter
nlp = spacy.load("en_core_web_sm")
doc = nlp("Monica, Mary and Oliver had lunch together and bought some Apple products.")

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

lables = [x.label_ for x in doc.ents]
Counter(lables) #count entity type occurrences
```

Monica PERSON , Mary PERSON and Oliver PERSON had lunch together and bought some Apple ORG products.

```python
#Visualising the document with identified named entities and entity types.
displacy.render(doc, jupyter=True, style='ent')
```

Now lets practice!

Putting together the NLP Pipeline

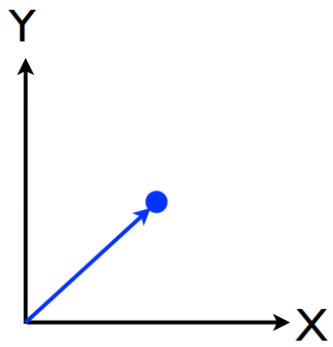


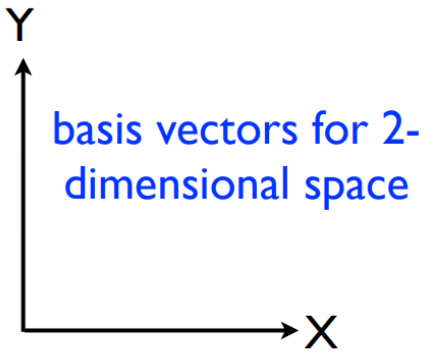
- Re-ordering the components and building the pipeline.
- Refer to *sampleText_Practice.txt*

Vectorizing the text

- Natural languages are ambiguous → harder for machines to understand.
- Machines can be trained over complex variants of mathematical models as interpretations of the underlying text if the textual data is transformed into numerical representations
- Turning text to numerical representations → '*vectorization*' or '*embedding*' techniques.
- Mapping words onto vectors of real numbers in the vector space which represents an algebraic model where all the rules of vector addition and measures of similarities apply.

- A vector is a point in a vector space and has length (from the origin to the point) and direction.
- A 2-dimensional vector can be written as [x,y].
- The basis vectors correspond to the dimensions or directions of the vector space.

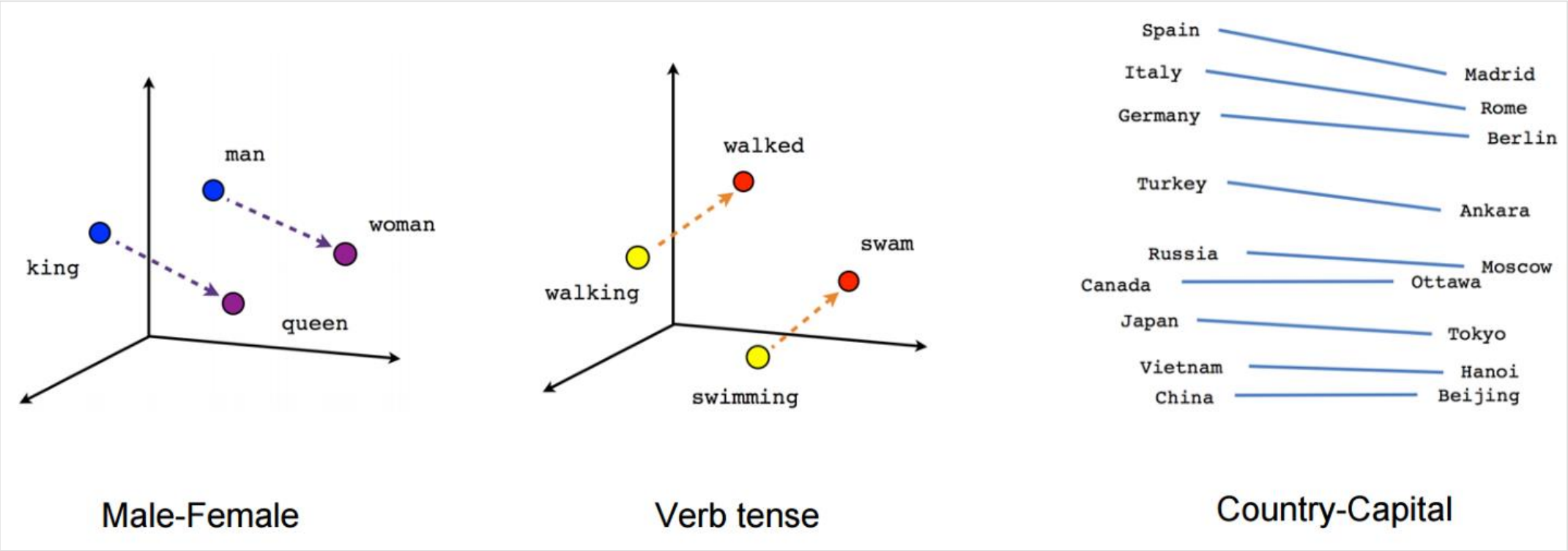




basis vectors for 2-dimensional space

Vector space is defined by a set of linearly independent basis vectors

- Similar words are found ‘closer’ in the vector space.
- Characters, groups of words or documents can be mapped to vectors as well.
- Useful for applications such as Text Classification.



➢Simplest method for embedding words to vectors is **Bag of Words** model.

➢Intuition behind this model is that two similar text fields will contain similar kind of words, and will therefore have a similar bag of words.

- Doc 1 = Sun shines bright.
- Doc2 = Sun is bright.
- Doc3 = It shines bright.

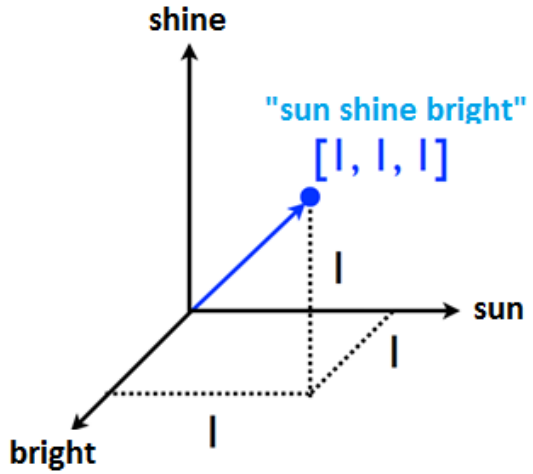
Vocab = {sun, shine, bright, is, it}.

	Doc 1	Doc 2	Doc 3
sun	1	1	0
shine	1	0	1
bright	1	1	1

Binary Weights

1 = the term appears at least once

0 = the term does not appear



Obtaining vectors from the term-document matrix:

sun = [1,1,0]

Doc3 = [0,1,1]

Problems with the model: not a good representation of language as it ignores word order, word relationships, meanings and context and produces sparse vectors that is largely filled with zeros.

Vector Space Model: TF-IDF Vectorizer

- **TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer** takes into account the importance of each term to document.
- TF-IDF vectorizes documents by calculating a TF-IDF statistic between the document and each term in the vocabulary.
- Term Document matrix represented by TF-IDF weights.
- TF-IDF accentuates terms that are frequent in the document, but not frequent in general.

Term Frequency

- Measures how frequently a term occurs in a document.
- A term might appear more times in long documents than shorter ones since every document's length is different.
- $tf(t, d)$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- Greater when a term is frequent in the document.

Inverse Document Frequency

- A word is not very informative if it occurs in all documents.
- Estimate the rarity of a term in the whole document collection.
- If a term occurs in all the documents of the collection, its IDF is zero.
- Greater when the term is rare in the collection

$$idf(t, D) = log(\frac{D}{df_t})$$

D = Number of documents in the collection, i.e. the Document space.
df_t = Number of documents in which term t appear, i.e., document frequency

TF-IDF

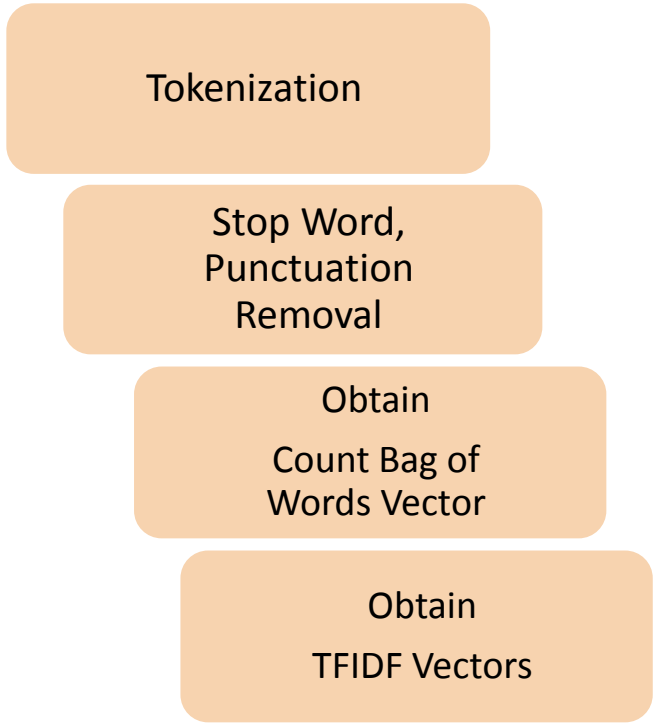
- The tf-idf weight of a term is the product of its tf weight and its idf weight, i.e.,

$$w(t) = tf(t, d) * \log(\frac{D}{df_t})$$

Now lets practice!

Vectorizing Text

- Refer to *textVectorization_Practice.txt*



Text Similarity Measures

- Computing similarity between two text pieces (terms/strings/documents/..)
- Example Applications:
  - Relevance of a document match for a query
  - Computing semantic relatedness between strings/terms
- Various measures available:
  - Edit Distance/Levenshtein Distance
  - Jaccard Distance
  - Cosine Similarity
  - …

Edit Distance

- Edit Distance between two strings is the minimum number of single character deletions, insertions, or substitutions required to transform one string into the other.
- The edit distance between ”good” and ”goodbye” is 3.
- Useful in spell checking applications.

```
#calculating edit distance between two terms

import nltk

w1 = 'mapping'
w2 = 'mappng'

nltk.edit_distance(w1, w2)
```

```
1
```

Edit Distance

- Edit Distance between two strings is the minimum number of single character deletions, insertions, or substitutions required to transform one string into the other.
- The edit distance between ”good” and ”goodbye” is 3.
- Useful in spell checking applications.

```
#calculating edit distance between two terms

import nltk

w1 = 'mapping'
w2 = 'mappng'

nltk.edit_distance(w1, w2)
```

```
1
```

Calculate the edit distance between the strings:
S1 = *‘It might help to re-install Python if possible.’*
S2= *‘I possibly love Python programming.’*

Would lemmatization have an effect on the Edit distance?

Jaccard Distance

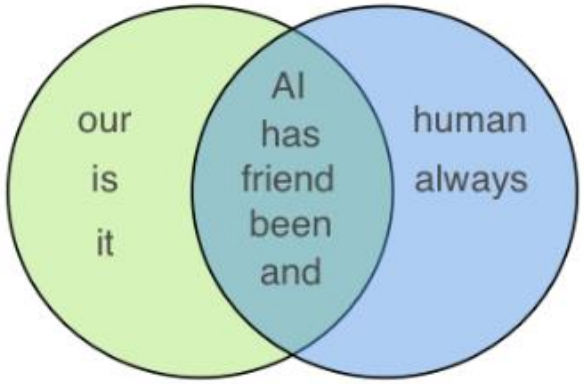
- Measure of how dissimilar two sets of strings are. The lower the distance, the stringer the string similarity.
- It is defined as the size of intersection divided by size of union of two sets.
- Perform lemmatization first in order to increase the number of size of intersection.

```
#calculating jaccard distance between two terms

import nltk

w1 = set('mapping')
w2 = set('mappng')

nltk.jaccard_distance(w1, w2)
```

```
0.16666666666666666
```

Jaccard Distance

- Measure of how dissimilar two sets of strings are. The lower the distance, the stringer the string similarity.
- It is defined as the size of intersection divided by size of union of two sets.
- Perform lemmatization first in order to increase the number of size of intersection.



Calculate the edit distance between the strings:
S1 = '*AI is our friend and it has been friendly.*'
S2= '*AI and humans have always been friendly.*'

Perform lemmatization to achieve better results.

Cosine Similarity

•Cosine similarity calculates similarity by measuring the **cosine of angle between two vectors**.

➔ Sentences should, therefore, first be converted to vectors using BOW or TFIDF methods.

cosine([1,0,1],[1,1,0]) = $\frac{(1*1)+(0*1)+(1*0)}{\sqrt{1^2+0^2+1^2}*\sqrt{1^2+1^2+0^2}}$ = 0.5

$$= \frac{inner\ product}{product\ of\ two\ vector\ lengths}$$

	Doc 1	Doc 2	Doc 3
sun	1	1	1
shine	1	0	1
bright	1	1	0

- Doc 1 = Sun shines bright.
- Doc2 = Sun is bright.
- Doc3 = Sun is shining.

Vocab = {sun, shine, bright, is, it}.

$$\frac{\sum_{i=1}^{V} x_i \times y_i}{\sqrt{\sum_{i=1}^{V} x_i^2} \times \sqrt{\sum_{i=1}^{V} y_i^2}}$$

length of vector x — length of vector y

Cosine Similarity

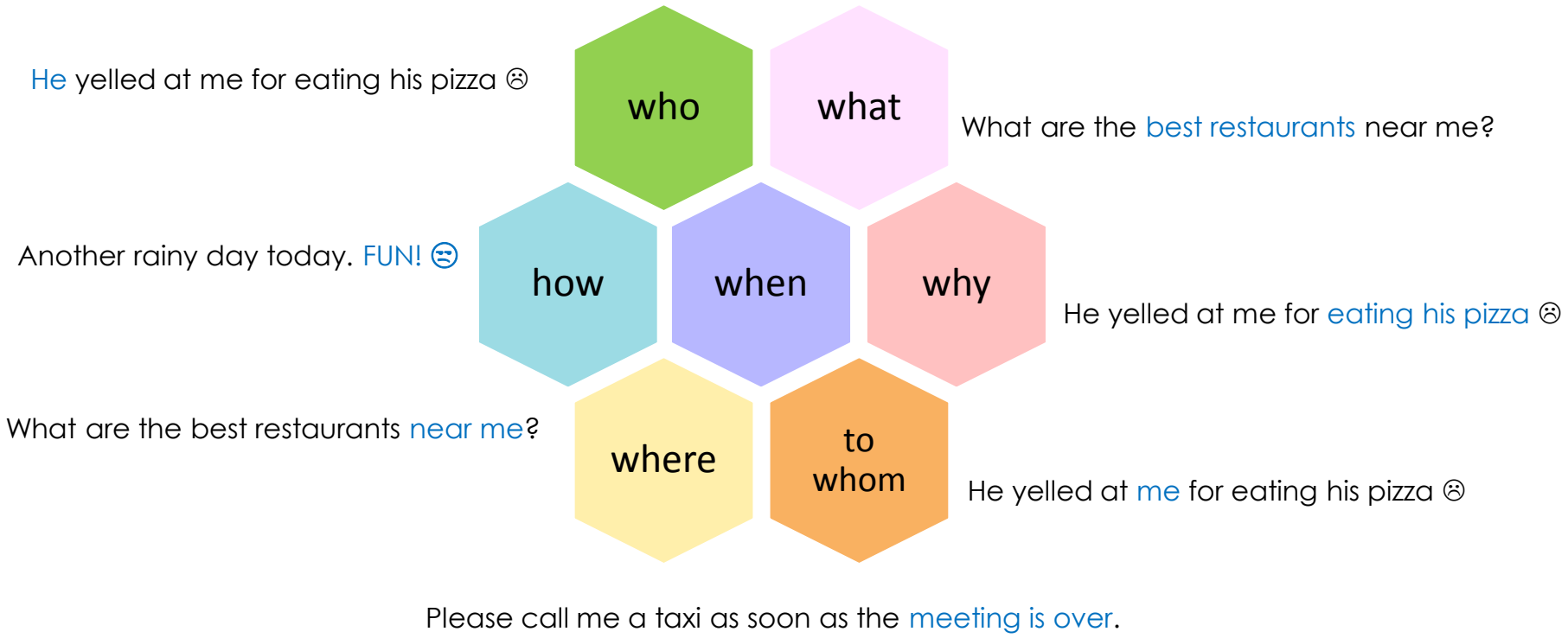
Calculate the cosine similarity between the strings:
S1 = '*AI is our friend and it has been friendly.*'
S2= '*AI and humans have always been friendly.*'

First obtain the TFIDF vectors and then dot product to obtain the cosine scores.

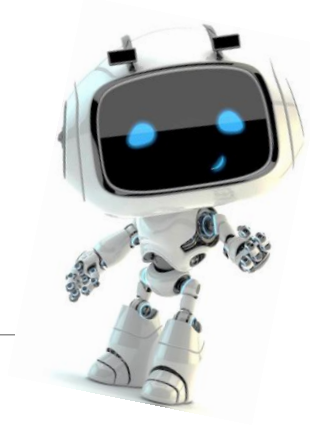
Why is NLP difficult?

I didn't really enjoy Sherlock Holmes. Won't be watching the next one.
Novel? Movie? Character? TV Series?

Understanding the meaning of what is said/written depends on not only the meaning itself, but also on:

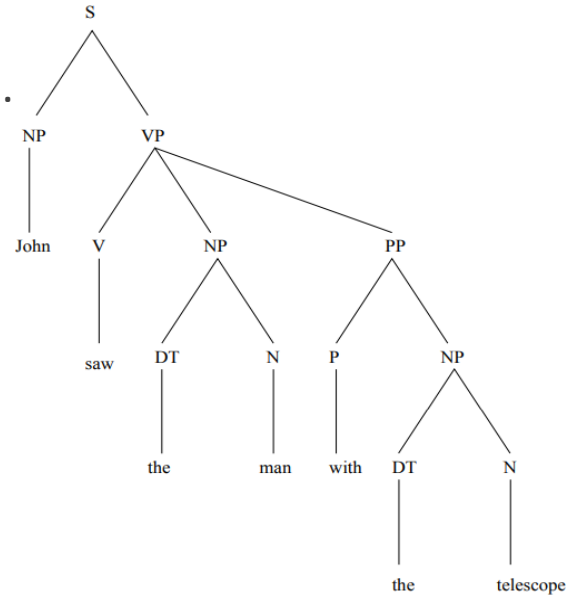


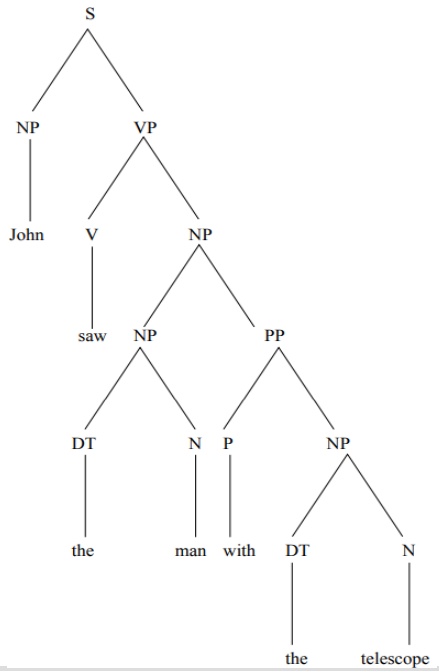
Why is NLP difficult?



Natural languages are ambiguous. This makes EVERY step in NLP hard!

- Morphological/lexical ambiguity: structure of words --- '*design*' as an adjective, noun and a verb (ambiguous part-of-speech)
  - That is a *well-designed* document.
  - I like that *design*.
  - She has a job in *designing* websites.
- Syntactical ambiguity: structure/grammar of phrases and sentences.
  - Call me a cab!
  - John saw the man with the telescope.
- Semantic ambiguity: Meaning of phrases and sentences.
  - I'm gonna take the bar. (The exam? Place for alcohol?)
  - John and Rachel are married. (To each other? Separately?)





- Pragmatic ambiguity: sense of goals, wishes, motivations (i.e., knowledge external to the document and/or query) – looking beyond the literal meaning of an utterance – THE CONTEXT.
  - Do you know what time it is? (You are late!)
  - Can you pass the salt? (Will you pass the salt?)
- Ellipsis: Omission of words that are needed for grammatical completion and are ‘understood’.
  - John has five dollars and Jane nine.
  - Mike: Who just walked by? Oliver: A tall blonde man.
- Anaphora: A phrase or word refers to something previously mentioned, but there is more than one possibility.
  - I went to the hospital, and they told me to go home and rest. (They = hospital staff)
  - Margaret invited Susan for a visit, and she gave her a good lunch. (she = Margaret; her = Susan)

Why is NLP difficult?

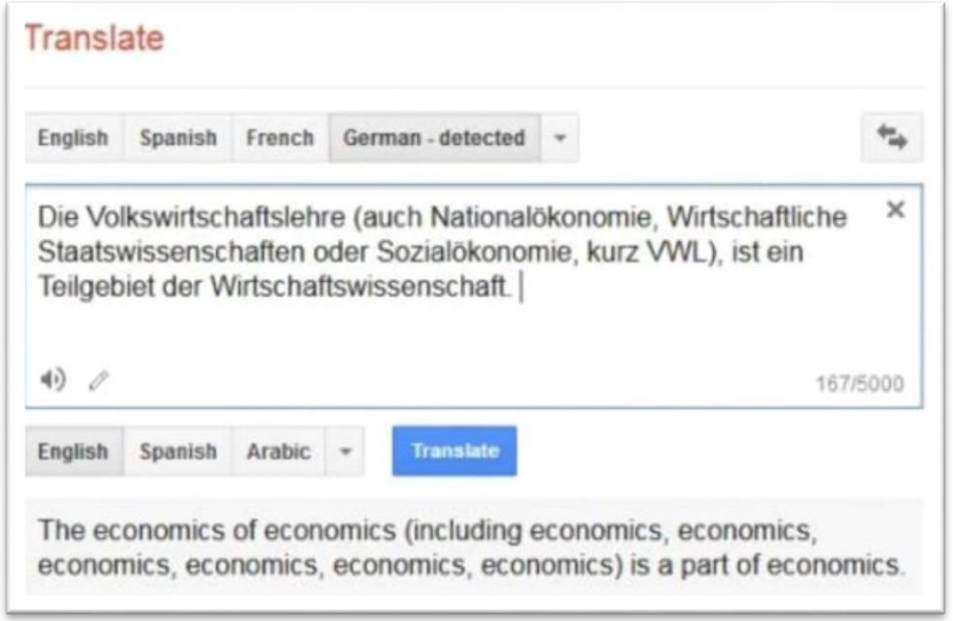
- Slang, abbreviations, typoes and grammatical errors
  - Can't wait to get home and drunk. TGIF I say!!
  - OMG. Ths standup is hilarious. I'm dying.
  - My dinner was better then yours.
- Idioms and metaphors
  - You light up my life.
  - Time is a thief.
- Body Language (eye roll, side eye, word stress,..)
- Context (personal/situational) matters!
  - It was a wet, muddy Sunday. The car parks were almost too full, rain was beating down.
  - Took my grandma out for a trip to Killerton. She has walking difficulties, but I shouldn't have worried as the staff were quite helpful and considerate.

Deeper NLP: Requires more human effort; less accurate!

Significance of NLP

-- Modern Applications

- Machine Translation (Breaking language barriers: Google Translate, Duolingo, Watson..)



Significance of NLP

-- Modern Applications

- Machine Translation (Breaking language barriers: Google Translate, Duolingo, Watson..)
- Sentiment Analysis (Predator/troll detection, Customer Exp. Management, ..)

Significance of NLP

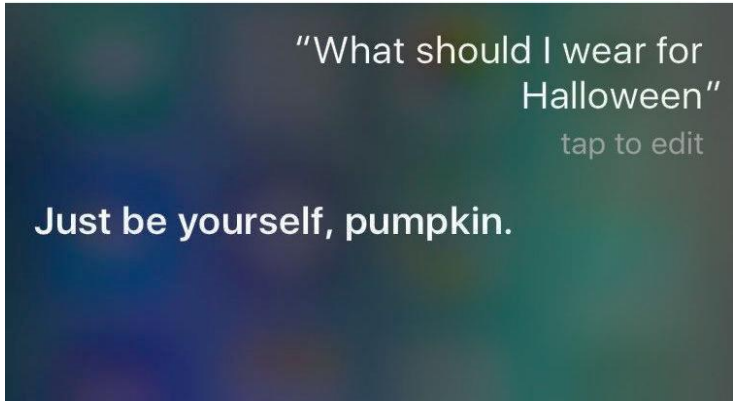
-- Modern Applications

- Machine Translation (Breaking language barriers: Google Translate, Duolingo, Watson..)
- Sentiment Analysis (Predator/troll detection, Customer Exp. Management, ..)
- Predictive text (keyboards,..)

Significance of NLP

-- Modern Applications

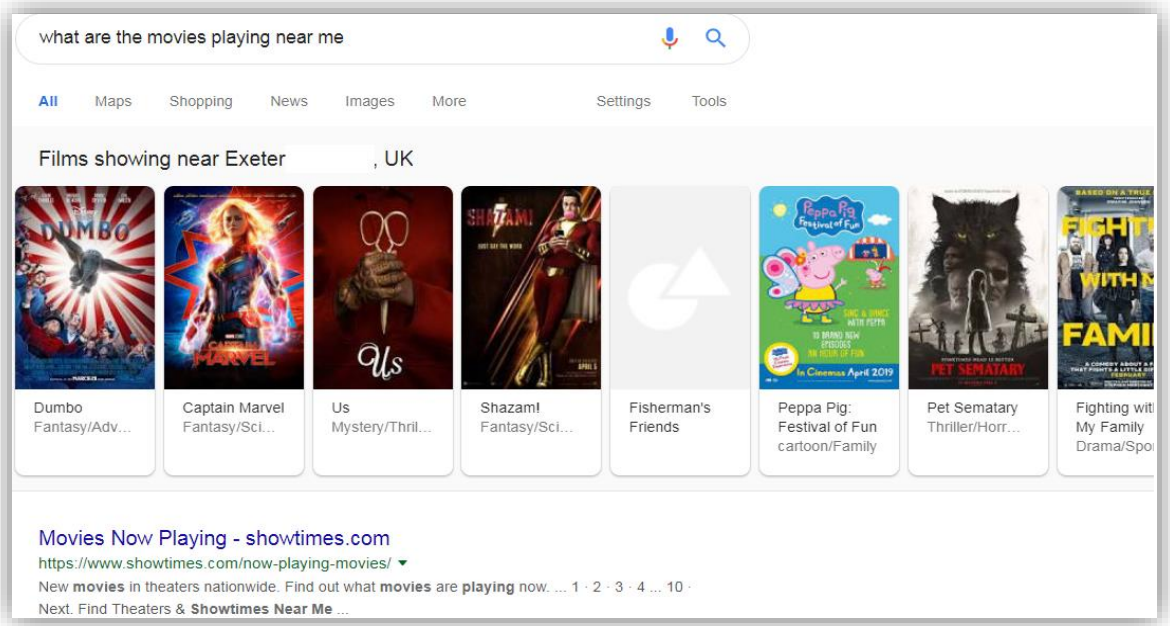
- Machine Translation (Breaking language barriers: Google Translate, Duolingo, Watson..)
- Sentiment Analysis (Predator/troll detection, Customer Exp. Management, ..)
- Predictive text (keyboards,..)
- Speech to text translation (Google Assistant, Alexa, Siri,..)



Significance of NLP

-- Modern Applications

- Machine Translation (Breaking language barriers: Google Translate, Duolingo, Watson..)
- Sentiment Analysis (Predator/troll detection, Customer Exp. Management, ..)
- Predictive text (keyboards,..)
- Speech to text translation (Google Assistant, Alexa, Siri,..)
- Question Answering (Google Q/A,..)



Significance of NLP

-- Modern Applications

- Machine Translation (Breaking language barriers: Google Translate, Duolingo, Watson..)
- Sentiment Analysis (Predator/troll detection, Customer Exp. Management, ..)
- Predictive text (keyboards,..)
- Speech to text translation (Google Assistant, Alexa, Siri,..)
- Question Answering (Google Q/A,..)
- Spam detection,..

Reference Material

1. Natural Language Processing with Python by Steven Bird, Ewan Klein and Edward Loper.
2. Foundations of Statistical Natural Language Processing by Christopher Manning and Hinrich Schütze.
3. Text Mining with R by Julia Silge and David Robinson.
4. Introduction to Information Retrieval by Christopher Manning, Prabhakar Raghavan and Hinrich Schütze. *(Information on Vector Space Models, TFIDF, Document Similarity Methods,..)*
5. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
6. A Beginner's Guide to Word2Vec and Neural Word Embeddings.


THANK YOU