Technical Session II

Topic Modelling- LDA

Dr. Ambika P
ambika2202@gmail.com
www.linkedin.com/in/drambika
+91 9590399036

Agenda

- Topic Modeling
- Methods
- LDA
- Word to Vec models
- Named Entity Recognition

Topic Modeling and why TM?

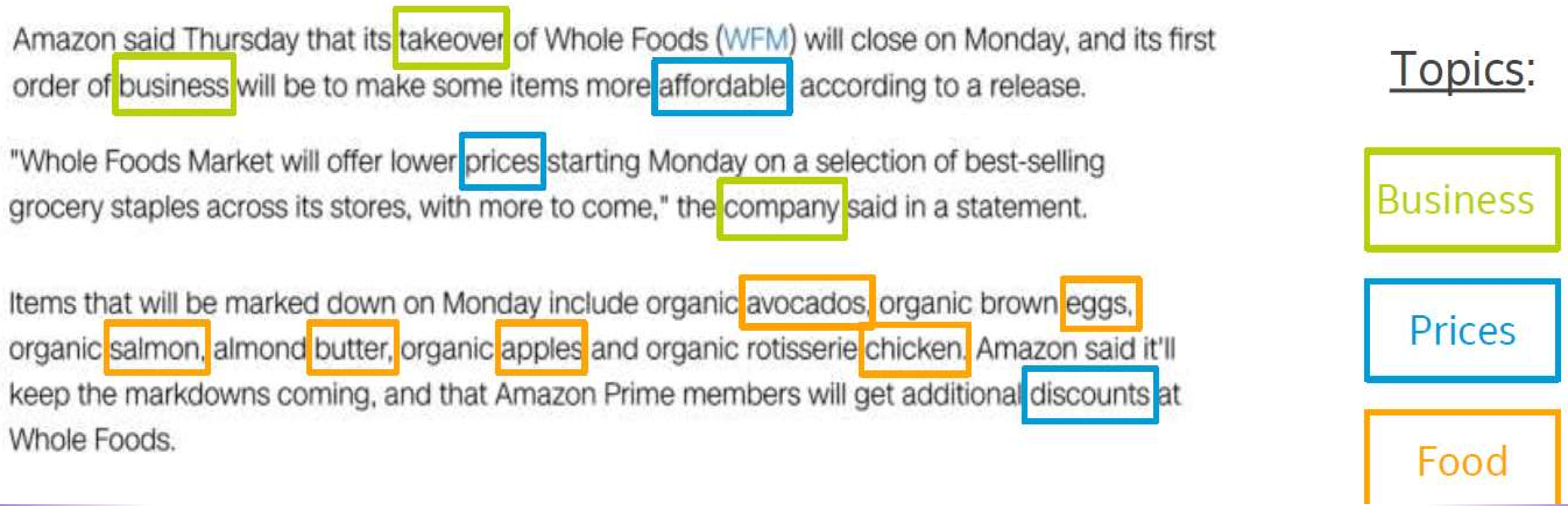
- Every document we read can be thought of as consisting of many topics all stacked upon one another.
- Large amounts of data are collected everyday. As more information becomes available, it becomes difficult to access what we are looking for. So, we need tools and techniques to organize, search and understand vast quantities of information.
- **Topic modelling provides us with methods to organize, understand and summarize large collections of textual information.**
- It helps in:
- Discovering hidden topical patterns that are present across the collection
- Annotating documents according to these topics
- Using these annotations to organize, search and summarize texts

Topic Modeling

- Topic modelling - described as a method for finding a group of words (i.e topic) from a collection of documents that best represents the information in the collection.
- It can also be thought of as a form of text mining – a way to obtain recurring patterns of words in textual material.
- There are many techniques that are used to obtain topic models.
- **Most popular techniques today: LSA, pLSA, LDA, lda2vec.(Deep learning model)**

LDA

- Goal: break text documents down into "topics" by word:



Amazon said Thursday that its takeover of Whole Foods (WFM) will close on Monday, and its first order of business will be to make some items more affordable according to a release.

"Whole Foods Market will offer lower prices starting Monday on a selection of best-selling grocery staples across its stores, with more to come," the company said in a statement.

Items that will be marked down on Monday include organic avocados, organic brown eggs, organic salmon, almond butter, organic apples and organic rotisserie chicken. Amazon said it'll keep the markdowns coming, and that Amazon Prime members will get additional discounts at Whole Foods.

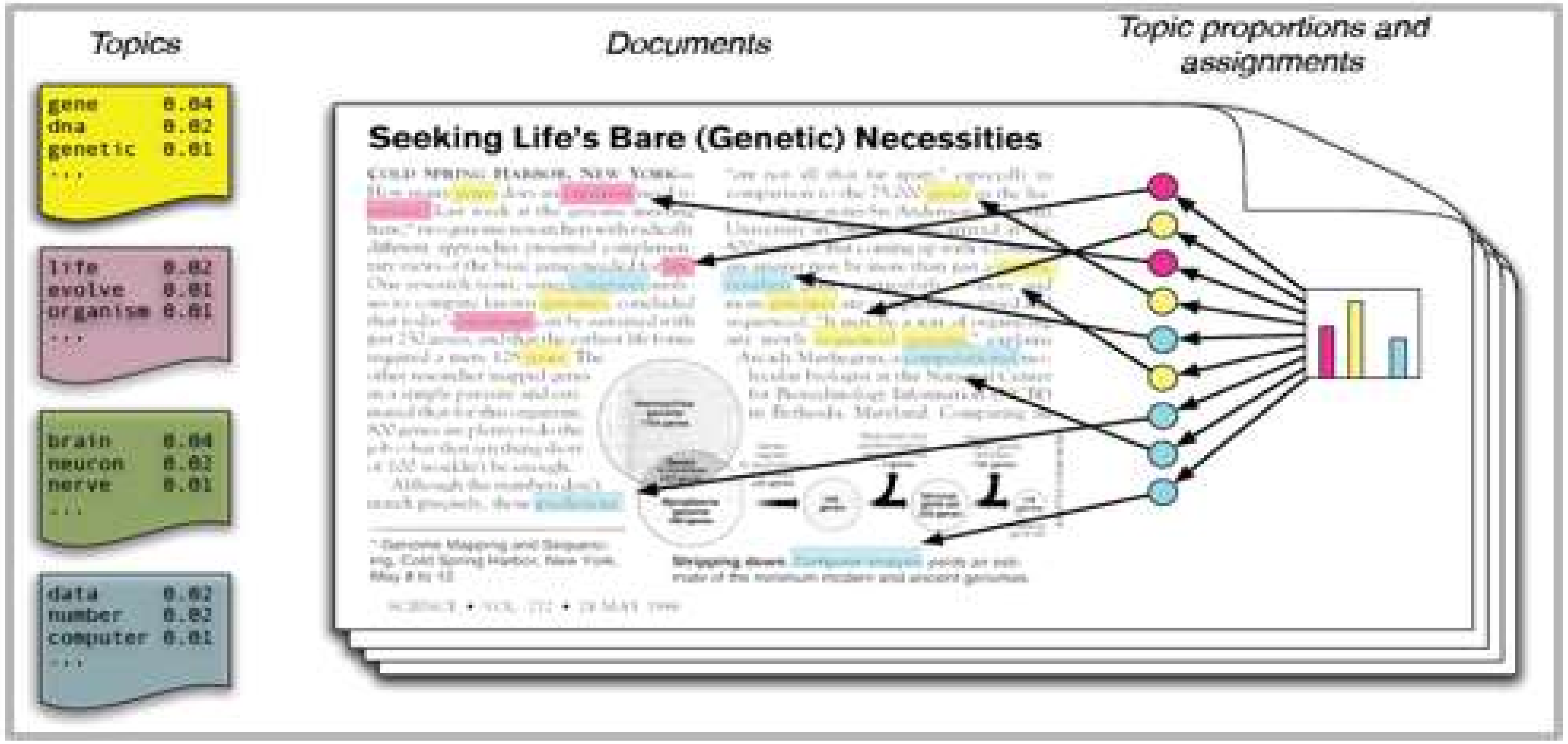
Topics:

Business

Prices

Food

Topic modeling



Example sentences

- For example, consider the following set of documents as the corpus:

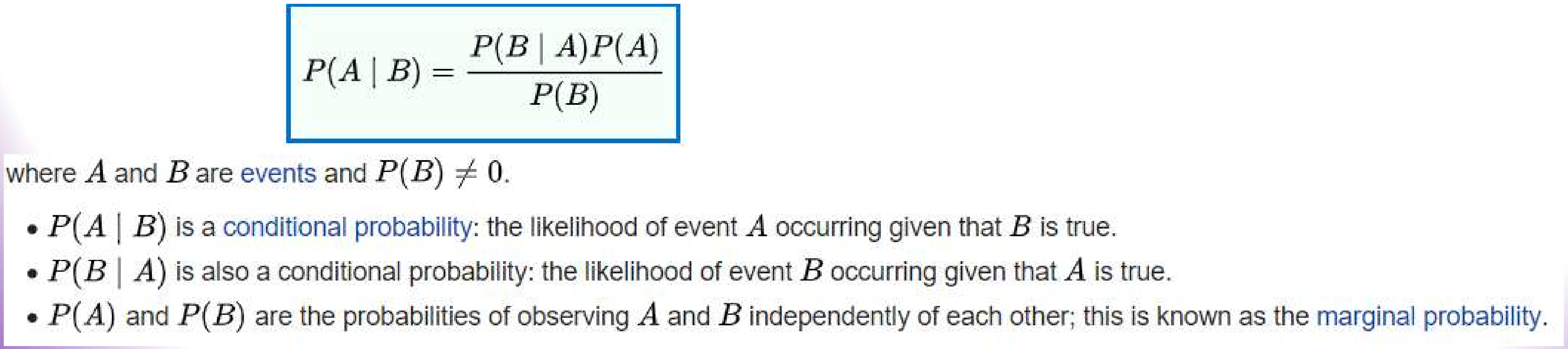
Document 1: I had a peanut butter sandwich for breakfast.
Document 2: I like to eat almonds, peanuts and walnuts.
Document 3: My neighbor got a little dog yesterday.
Document 4: Cats and dogs are mortal enemies.
Document 5: You mustn't feed peanuts to your dog.

LDA model discovers the different topics that the documents represent and how much of each topic is present in a document.

- **Topic 1**: 30% peanuts, 15% almonds, 10% breakfast… (you can interpret that this topic deals with food)
  Topic 2: 20% dogs, 10% cats, 5% peanuts… (you can interpret that this topic deals with pets or animals)
- **Documents 1 and 2**: 100% Topic 1
  Documents 3 and 4: 100% Topic 2
  Document 5: 70% Topic 1, 30% Topic 2

LDA Procedure

- Collapsed Gibbs sampling is one way the LDA learns the topics and the topic representations of each document.
- Gibbs sampling is commonly used as a means of statistical inference, especially Bayesian inference.



$$\boxed{P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}}$$

where A and B are events and $P(B) \neq 0$.

- $P(A \mid B)$ is a conditional probability: the likelihood of event A occurring given that B is true.
- $P(B \mid A)$ is also a conditional probability: the likelihood of event B occurring given that A is true.
- $P(A)$ and $P(B)$ are the probabilities of observing A and B independently of each other; this is known as the marginal probability.

LDA procedure

- *Go through each document and randomly assign each word in the document to one of K topics (K is chosen beforehand)*
- *This random assignment gives topic representations of all documents and word distributions of all the topics, albeit not very good ones*
- *So, to improve upon them:*
  - *For each document d, go through each word w and compute:*
    - *p(topic t | document d): proportion of words in document d that are assigned to topic t*
    - *p(word w| topic t): proportion of assignments to topic t, over all documents d, that come from word w*
- *Reassign word w a new topic t', where we choose topic t' with probability p(topic t' | document d) * p(word w | topic t')*
  This generative model predicts the probability that topic t' generated word w
- repeating the last step a large number of times, we reach a steady state where topic assignments are pretty good. These assignments are then used to determine the topic mixtures of each document
- After a number of iterations, a steady state is achieved where the document topic and topic term distributions are fairly good. This is the convergence point of LDA

Applying the LDA model

corpus is a document-term matrix and now we're ready to generate an LDA model:
ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=3, id2word = dictionary, passes=20)
The LdaModel Parameters:
•num_topics: *required.* An LDA model requires the user to determine how many topics should be generated. Our document set is small, so we're only asking for three topics.
•id2word: *required.* The LdaModel class requires our previous dictionary to map ids to strings.
•passes: *optional.* The number of laps the model will take through corpus. The greater the number of passes, the more accurate the model will be. A lot of passes can be slow on a very large corpus.

what does LDA *actually* do?

- LDA assumes documents are produced from a mixture of topics. Those topics then generate words based on their probability distribution, like the ones in our walkthrough model. In other words, LDA assumes a document is made from the following steps:
- Determine the number of words in a document. Let's say our document has 6 words.
- Determine the mixture of topics in that document. For example, the document might contain 1/2 the topic "health" and 1/2 the topic "vegetables."
- Using each topic's multinomial distribution, output words to fill the document's word slots. In our example, the "health" topic is 1/2 our document, or 3 words. The "health" topic might have the word "diet" at 20% probability or "exercise" at 15%, so it will fill the document word slots based on those probabilities.
- Given this assumption of how documents are created, LDA backtracks and tries to figure out what topics would create those documents in the first place.

How to improve

- **Frequency Filter –** Arrange every term according to its frequency. Terms with higher frequencies are more likely to appear in the results as compared ones with low frequency. The low frequency terms are essentially weak features of the corpus, hence it is a good practice to get rid of all those weak features. An exploratory analysis of terms and their frequency can help to decide what frequency value should be considered as the threshold.
- **Part of Speech Tag Filter –** POS tag filter is more about the context of the features than frequencies of features. Topic Modelling tries to map out the recurring patterns of terms into topics. However, every term might not be equally important contextually. For example, POS tag IN contain terms such as – "within", "upon", "except". "CD" contains – "one","two", "hundred" etc. "MD" contains "may", "must" etc. These terms are the supporting words of a language and can be removed by studying their post tags.
- **Batch Wise LDA –**In order to retrieve most important topic terms, a corpus can be divided into batches of fixed sizes. Running LDA multiple times on these batches will provide different results, however, the best topic terms will be the intersection of all batches.

Package required and installation

- Gensim
- -conda install -c conda-forge genism
- conda install -c conda-forge spacy
- python -m spacy download en (default English model)

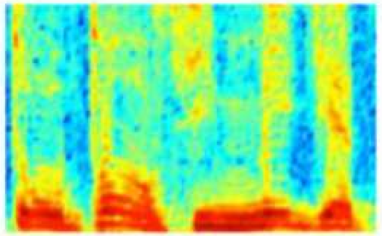
Result

```
[(0, '0.045*"never" + 0.045*"perform" + 0.045*"seems"'), (1,
'0.085*"sugar" + 0.084*"father" + 0.084*"sister"'), (2,
'0.065*"driving" + 0.065*"pressure" + 0.064*"stress"')]
```

Word to Vec

Image and audio processing systems work with rich, high-dimensional datasets encoded as vectors.

AUDIO



Audio Spectrogram

DENSE

IMAGES

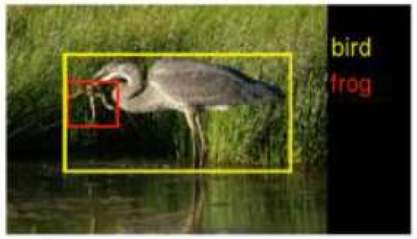


Image pixels

DENSE

TEXT

0	0	0	0.2	0	0.7	0	0	0	...	...

Word, context, or document vectors

SPARSE

- Model perplexity and topic coherence provide a convenient measure to judge how good a given topic model is. topic coherence score, in particular, has been more helpful.

```
# Compute Perplexity print('\nPerplexity: ', lda_model.log_perplexity(corpus))
 # a measure of  how good the model is. lower the better.
# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized,
 dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

Word Embedding Approaches

- Represent words in a numeric format that is understandable by the computers.
- Word embedding refers to the numeric representations of words.
- Several word embedding approaches currently exist and all of them have their pros and cons. three of them here:
- Bag of Words
- TF-IDF Scheme
- Word2Vec

Bag of Words

- The bag of words approach is one of the simplest word embedding approaches. The following are steps to generate word embeddings using the bag of words approach.
- We will see the word embeddings generated by the bag of words approach with the help of an example. Suppose you have a corpus with three sentences.
- S1 = I love rain
- S2 = rain rain go away
- S3 = I am away
- To convert above sentences into their corresponding word embedding representations using the bag of words approach, we need to perform the following steps:
- Create a dictionary of unique words from the corpus. In the above corpus, we have following unique words: [I, love, rain, go, away, am]
- Parse the sentence. For each word in the sentence, add 1 in place of the word in the dictionary and add zero for all the other words that don't exist in the dictionary. For instance, the bag of words representation for sentence S1 (I love rain), looks like this: [1, 1, 1, 0, 0, 0]. Similarly for S2 and S3, bag of word representations are [0, 0, 2, 1, 1, 0] and [1, 0, 0, 0, 1, 1], respectively.
- Notice that for S2 we added 2 in place of "rain" in the dictionary; this is because S2 contains "rain" twice.

TF-IDF

- The scheme is a type of bag words approach where instead of adding zeros and ones in the embedding vector, you add floating numbers that contain more useful information compared to zeros and ones. The idea behind TF-IDF scheme is the fact that words having a high frequency of occurrence in one document, and less frequency of occurrence in all the other documents, are more crucial for classification.
- TF-IDF is a product of two values: Term Frequency (TF) and Inverse Document Frequency (IDF).
- Term frequency refers to the number of times a word appears in the document and can be calculated as:
- Term frequency = (Number of Occurrences of a word)/(Total words in the document)

- For instance, if we look at sentence S1 from the previous section i.e. "I love rain", every word in the sentence occurs once and therefore has a frequency of 1. On the contrary, for S2 i.e. "rain rain go away", the frequency of "rain" is two while for the rest of the words, it is 1.
- IDF refers to the log of the total number of documents divided by the number of documents in which the word exists, and can be calculated as:
- IDF(word) = Log((Total number of documents)/(Number of documents containing the word))

For instance, the IDF value for the word "rain" is 0.1760, since the total number of documents is 3 and rain appears in 2 of them, therefore log(3/2) is 0.1760.
On the other hand, if you look at the word "love" in the first sentence
it appears in one of the three documents and therefore its IDF value is log(3), which is 0.4771.

Motivation

However, natural language processing systems traditionally treat words as discrete atomic symbols.

- Encodings are arbitrary
- Provide no useful information to the system
- Leads to data sparsity (So, we need more data)

Word as Atomic Units

cat		[0,...,0,1,0,...,0]
dog		[0,...,1,0,0,...,0]

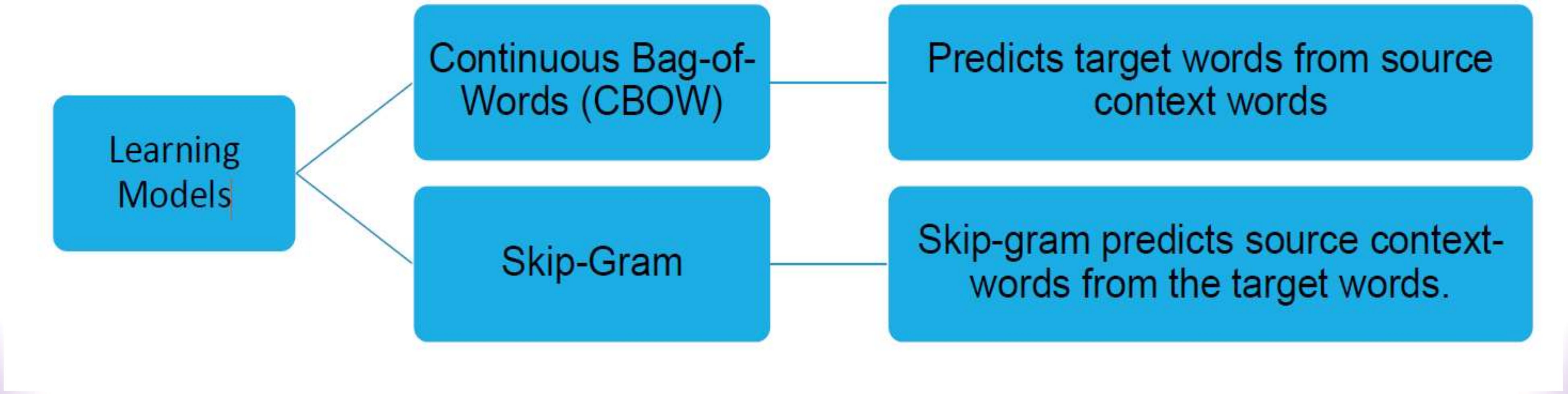
What is word-vec

Using vector representations can overcome some of these obstacles!

$$cat \longrightarrow [0.122, 0.81, 0.405, \ldots, 0.77]$$

Word2Vec is a model for learning vector representations of words, called "word embeddings".

Learning Models



Learning Models
Continuous Bag-of-Words (CBOW)
Predicts target words from source context words
Skip-Gram
Skip-gram predicts source context-words from the target words.

Gensim

The basic idea of word embedding is words that occur in similar context tend to be closer to each other in vector space. For generating word vectors in Python nltk and genism is required

Gensim is a FREE Python library.

It served to generate a short list of the most similar articles to a given article (gensim = “generate similar”).

The Word2Vec training algorithms were originally ported from the C package and extended with additional functionality.

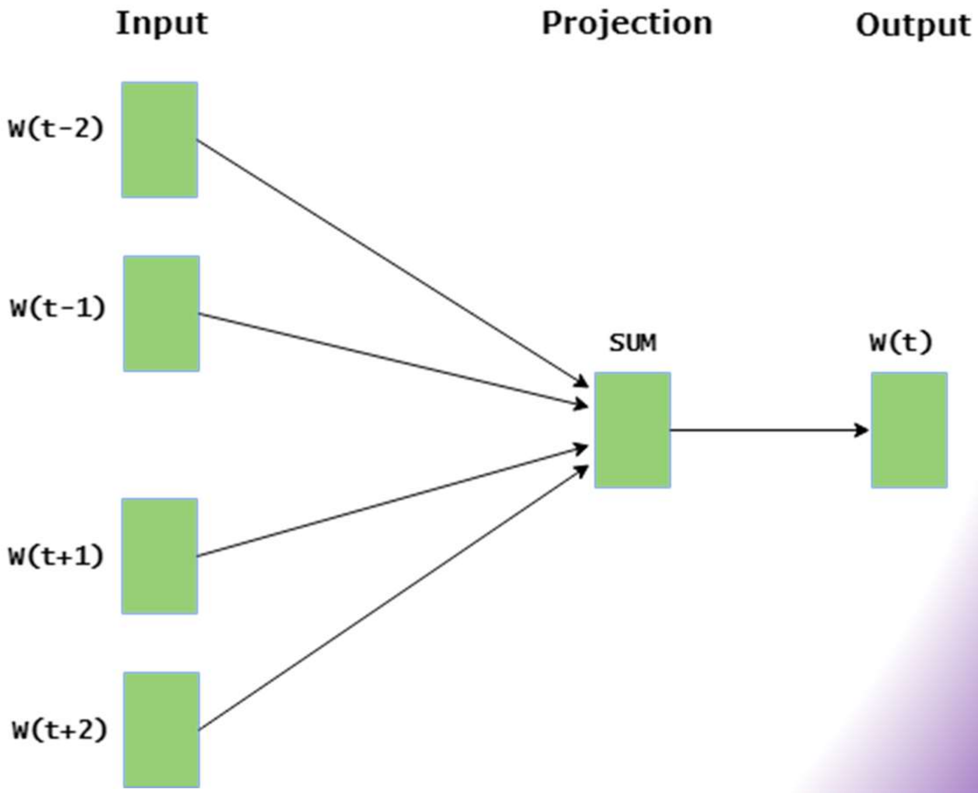
Two architectures

- **Word2Vec** consists of models for generating word embedding. These models are shallow two layer neural networks .
- It has one input layer, one hidden layer and one output layer. Word2Vec utilizes two architectures :

- **CBOW (Continuous Bag of Words)**
- **Skip Gram**

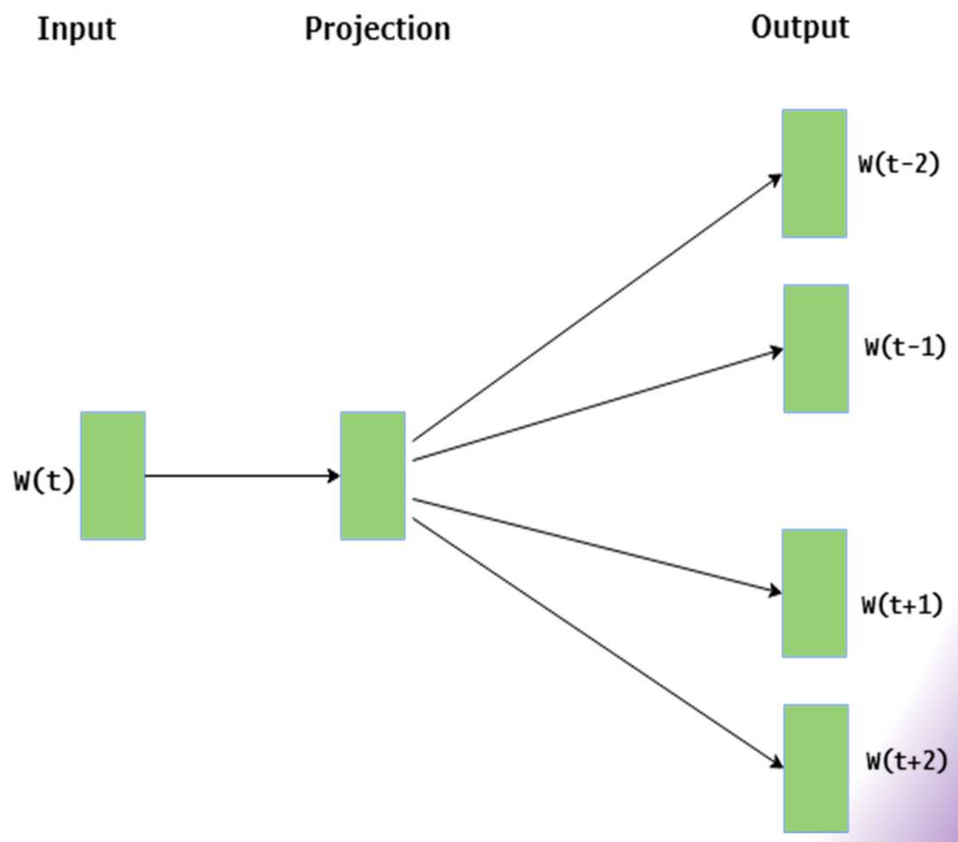
CBOW (Continuous Bag of Words)

- CBOW model predicts the current word given context words within specific window.
- The input layer contains the context words
- the output layer contains the current word.
- The hidden layer contains the number of dimensions in which we want to represent current word present at the output layer.



Skip Gram

- Skip gram predicts the surrounding context words within specific window given current word.
- The input layer contains the current word
- output layer contains the context words.
- The hidden layer contains the number of dimensions in which we want to represent current word present at the input layer.

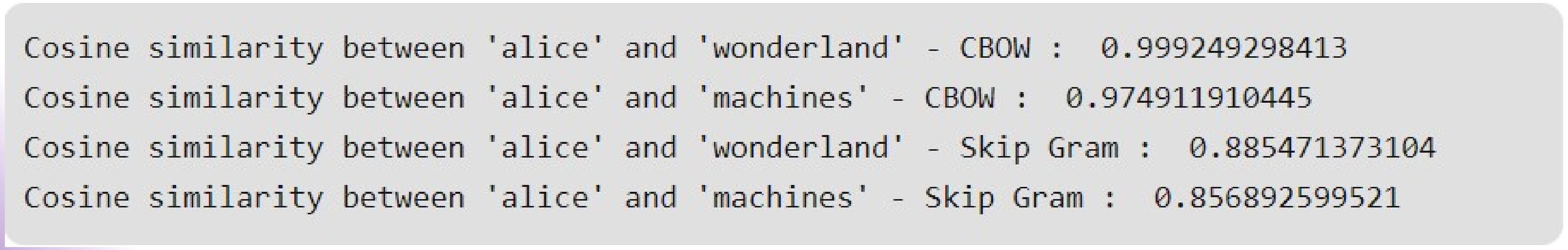


Example and difference

- CBOW: The cat ate ______. Fill in the blank, in this case, it's "food".
- Skip-gram: ___ ___ ___ food. Complete the word's context. In this case, it's "The cat ate"

- Output indicates the cosine similarities between word vectors 'alice', 'wonderland' and 'machines' for different models.
- One interesting task might be to change the parameter values of 'size' and 'window' to observe the variations in the cosine similarities.

Output



```
Cosine similarity between 'alice' and 'wonderland' - CBOW :  0.999249298413
Cosine similarity between 'alice' and 'machines' - CBOW :  0.974911910445
Cosine similarity between 'alice' and 'wonderland' - Skip Gram :  0.885471373104
Cosine similarity between 'alice' and 'machines' - Skip Gram :  0.856892599521
```

Applications – Word embedding

```
Applications of Word Embedding
 >> Sentiment Analysis
>> Speech Recognition
>> Information Retrieval
>> Question Answering
```

Word to Vec

- Bag of Words
- TF-IDF Scheme
- Word2Vec

NER Named Entity Recognition

- Named entity recognition (NER) , also known as entity chunking/extraction
- popular technique used in information extraction to identify and segment the named entities and classify or categorize them under various predefined classes.
- In any text document, there are particular terms that represent specific entities that are more informative and have a unique context.
- These entities are known as named entities , which more specifically refer to terms that represent real-world objects like people, places, organizations, and so on, which are often denoted by proper names.
- A naive approach could be to find these by looking at the noun phrases in text documents.
- Named entity recognition (NER) , also known as entity chunking/extraction , is a popular technique used in information extraction to identify and segment the named entities and classify or categorize them under various predefined classes.

Named Entity Recognition

- NER - First step towards information extraction
- It locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.
- NER is used in many fields in Natural Language Processing (NLP), and it can help answering many real-world questions, such as:
- **Which companies were mentioned in the news article?**
- **Were specified products mentioned in complaints or reviews?**
- **Does the tweet contain the name of a person? Does the tweet contain this person's location?**

Steps

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
```

Information Extraction
sentence from The New York Times, “European authorities fined Google a record $5.1 billion on Wednesday for abusing its power in the mobile phone market and ordered the company to alter its practices.”

2. apply word tokenization and part-of-speech tagging to the sentence. get a list of tuples containing the individual words in the sentence and their associated part-of-speech.

```
def preprocess(sent):
    sent = nltk.word_tokenize(sent)
    sent = nltk.pos_tag(sent)
    return sent
```

Noun phrase chunking

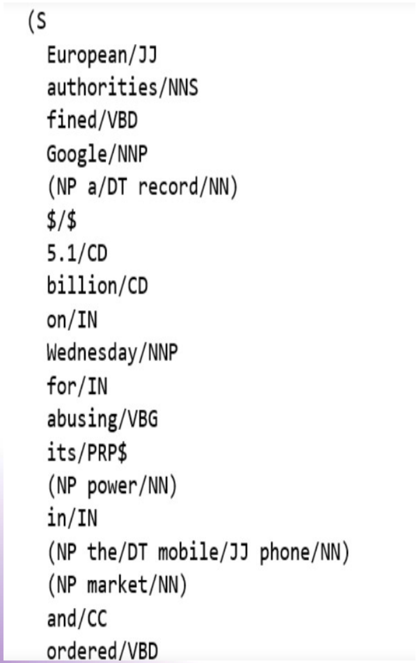
- Noun phrase chunking - identify named entities using a regular expression consisting of rules that indicate how sentences should be chunked.
- Example
- Our chunk pattern consists of one rule, that a noun phrase, NP, should be formed whenever the chunker finds an optional determiner, DT, followed by any number of adjectives, JJ, and then a noun, NN.

```
pattern = 'NP: {<DT>?<JJ>*<NN>}'
```

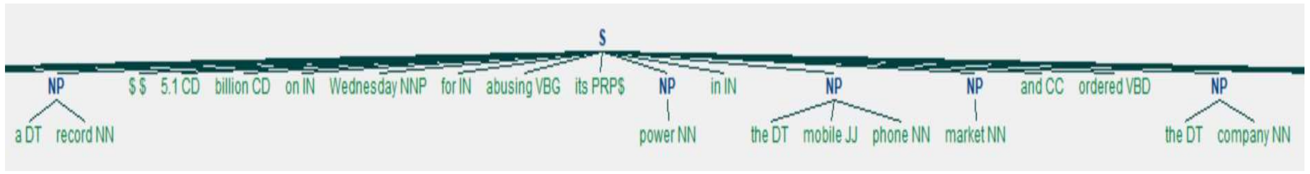
- Using this pattern, we create a chunk parser and test it on our sentence.

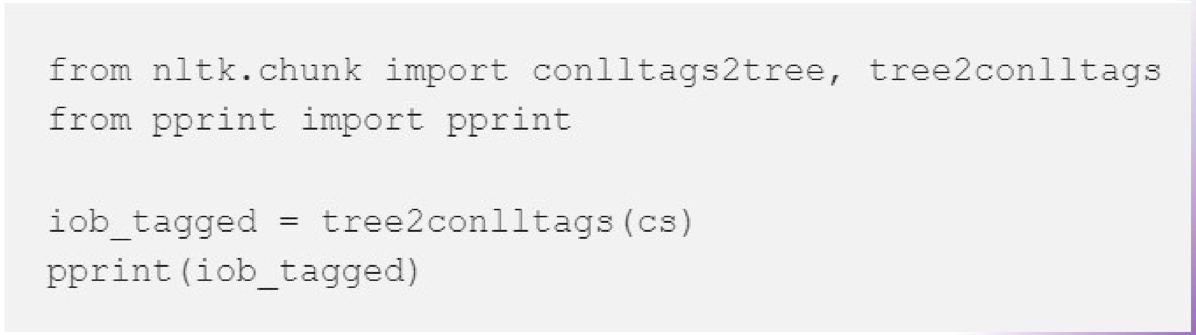
```
cp = nltk.RegexpParser(pattern)
cs = cp.parse(sent)
print(cs)
```

- Output can be read as a tree or a hierarchy with S as the first level, denoting sentence
- IOB tags have become the standard way to represent chunk structures in files



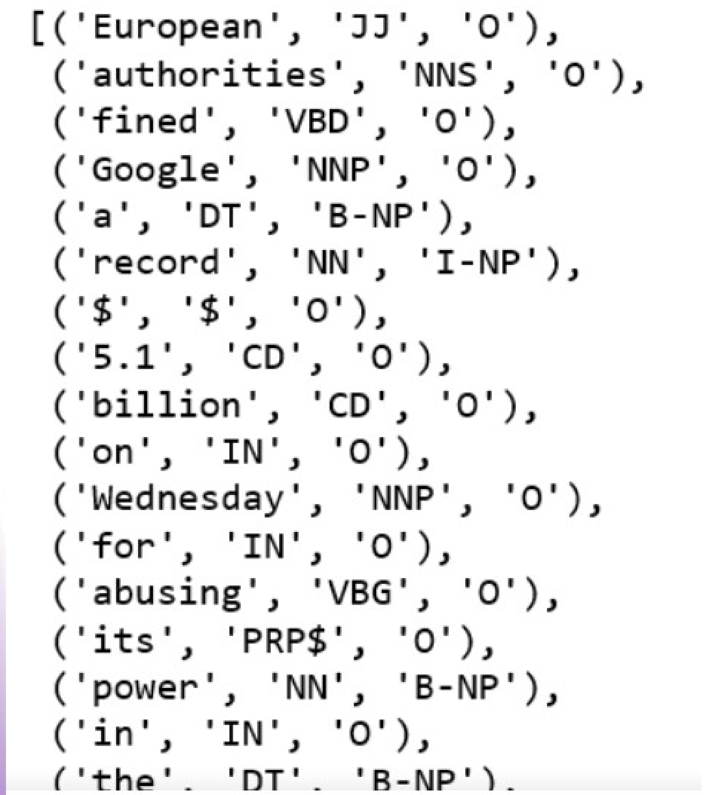
```
(S
  European/JJ
  authorities/NNS
  fined/VBD
  Google/NNP
  (NP a/DT record/NN)
  $/$
  5.1/CD
  billion/CD
  on/IN
  Wednesday/NNP
  for/IN
  abusing/VBG
  its/PRP$
  (NP power/NN)
  in/IN
  (NP the/DT mobile/JJ phone/NN)
  (NP market/NN)
  and/CC
  ordered/VBD
```





```
from nltk.chunk import conlltags2tree, tree2conlltags
from pprint import pprint

iob_tagged = tree2conlltags(cs)
pprint(iob_tagged)
```

Output – IOB Tagged

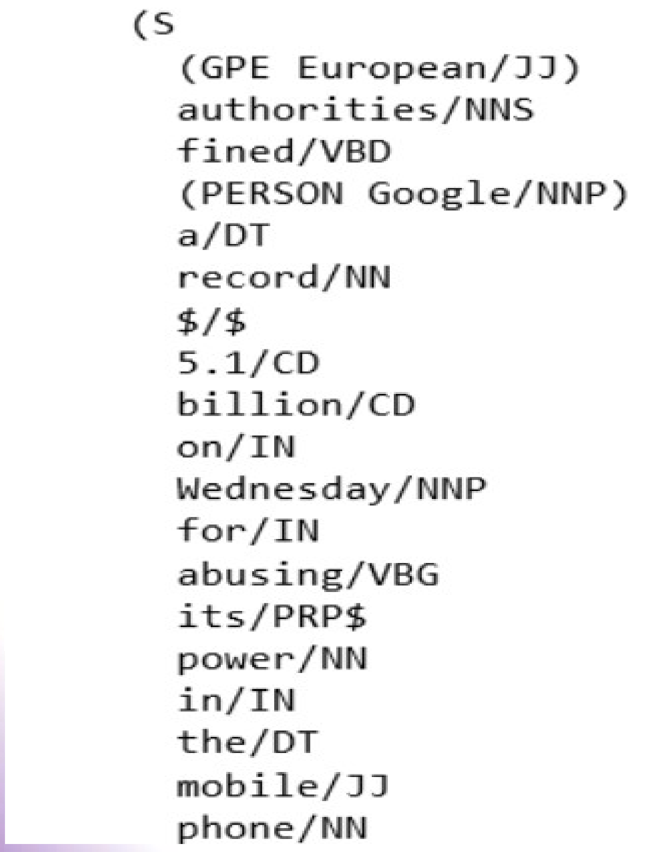


```
[('European', 'JJ', 'O'),
 ('authorities', 'NNS', 'O'),
 ('fined', 'VBD', 'O'),
 ('Google', 'NNP', 'O'),
 ('a', 'DT', 'B-NP'),
 ('record', 'NN', 'I-NP'),
 ('$', '$', 'O'),
 ('5.1', 'CD', 'O'),
 ('billion', 'CD', 'O'),
 ('on', 'IN', 'O'),
 ('Wednesday', 'NNP', 'O'),
 ('for', 'IN', 'O'),
 ('abusing', 'VBG', 'O'),
 ('its', 'PRP$', 'O'),
 ('power', 'NN', 'B-NP'),
 ('in', 'IN', 'O'),
 ('the', 'DT', 'B-NP'),
```

- one token per line, each with its part-of-speech tag and its named entity tag.
- Based on this training corpus, we can construct a tagger that can be used to label new sentences
- use nltk.chunk.conlltags2tree() function to convert the tag sequences into a chunk tree.
- function nltk.ne_chunk() - Can recognize named entities using a classifier, the classifier adds category labels such as PERSON, ORGANIZATION, and GPE.

```
ne_tree = ne_chunk(pos_tag(word_tokenize(ex)))
print(ne_tree)
```

Output

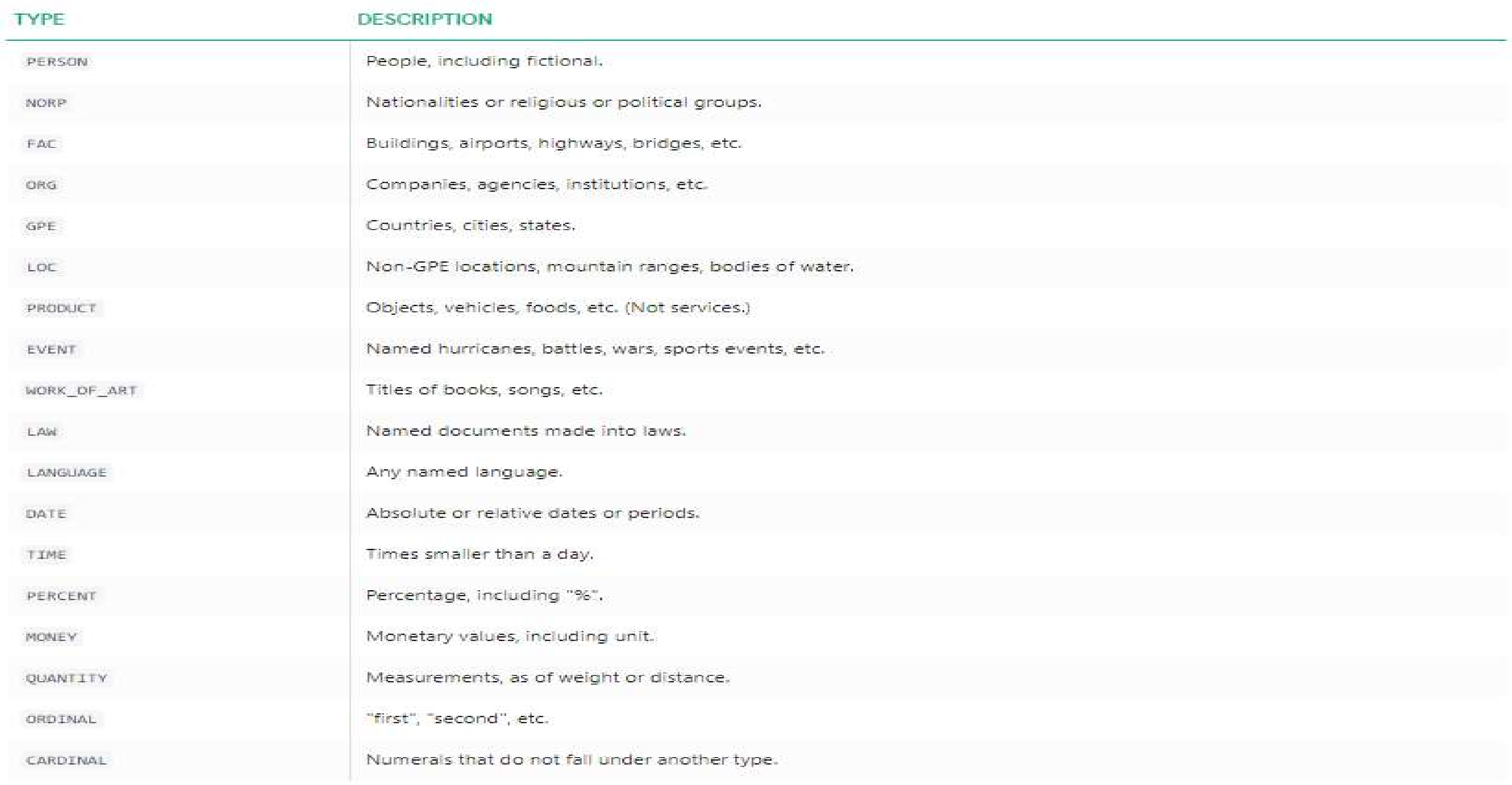


```
(S
  (GPE European/JJ)
  authorities/NNS
  fined/VBD
  (PERSON Google/NNP)
  a/DT
  record/NN
  $/$
  5.1/CD
  billion/CD
  on/IN
  Wednesday/NNP
  for/IN
  abusing/VBG
  its/PRP$
  power/NN
  in/IN
  the/DT
  mobile/JJ
  phone/NN
```

- Google is recognized as a person. It’s quite disappointing,

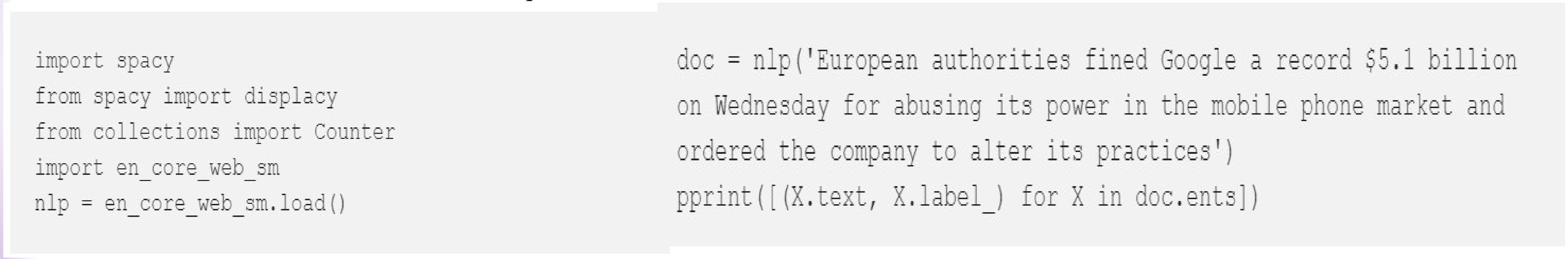
Spacy

- SpaCy's named entity recognition has been trained on the OntoNotes 5 corpus and it supports the following entity types:



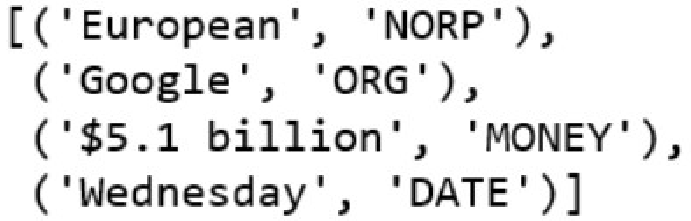
TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.
CARDINAL	Numerals that do not fall under another type.

One of the nice things about Spacy is that we only need to apply nlp once, the entire background pipeline will return the objects.
We use same text test it and the output is >> conda install - conda



```
import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm
nlp = en_core_web_sm.load()
```

```
doc = nlp('European authorities fined Google a record $5.1 billion
on Wednesday for abusing its power in the mobile phone market and
ordered the company to alter its practices')
pprint([(X.text, X.label_) for X in doc.ents])
```

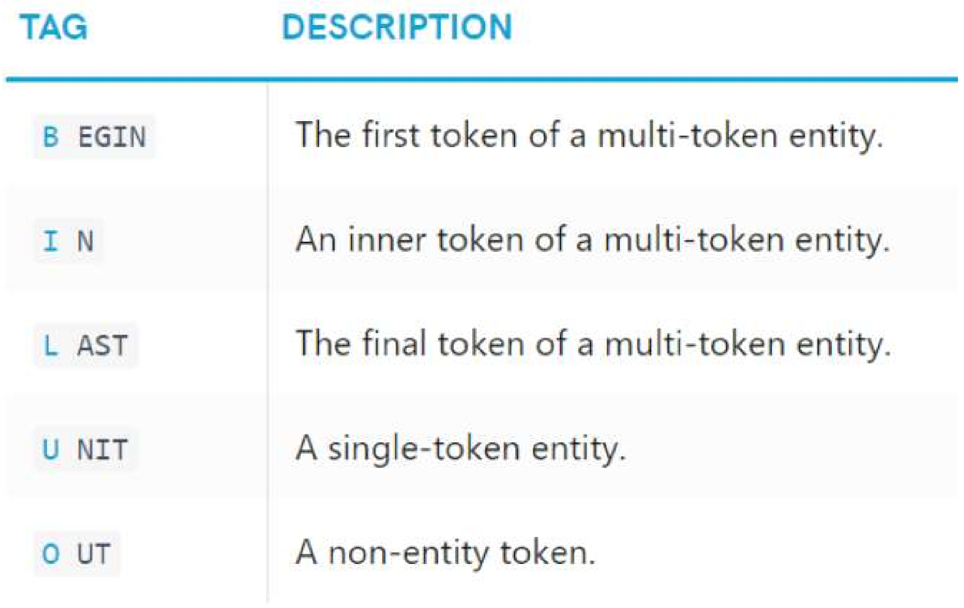


```
[('European', 'NORP'),
 ('Google', 'ORG'),
 ('$5.1 billion', 'MONEY'),
 ('Wednesday', 'DATE')]
```

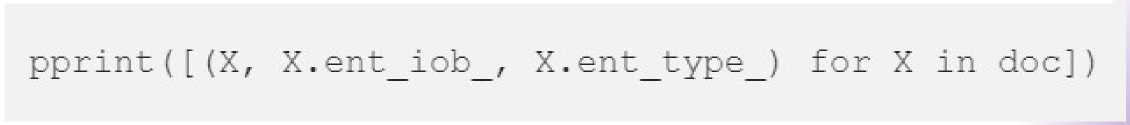
Output : European is NORD (nationalities or religious or political groups), Google is an organization, $5.1 billion is monetary value and Wednesday is a date object. They are all correct.

BILOU tagging :token-level entity annotation

- In the previous example
- During the above example, we were working on entity level, in the following example, we are demonstrating token-level entity annotation using the BILUO tagging scheme to describe the entity boundaries.

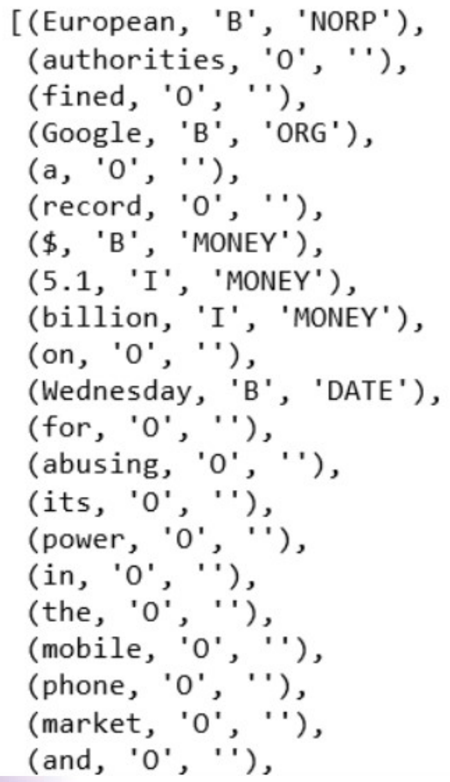


TAG	DESCRIPTION
B EGIN	The first token of a multi-token entity.
I N	An inner token of a multi-token entity.
L AST	The final token of a multi-token entity.
U NIT	A single-token entity.
O UT	A non-entity token.



```
pprint([(X, X.ent_iob_, X.ent_type_) for X in doc])
```

Output



```
[(European, 'B', 'NORP'),
 (authorities, 'O', ''),
 (fined, 'O', ''),
 (Google, 'B', 'ORG'),
 (a, 'O', ''),
 (record, 'O', ''),
 ($, 'B', 'MONEY'),
 (5.1, 'I', 'MONEY'),
 (billion, 'I', 'MONEY'),
 (on, 'O', ''),
 (Wednesday, 'B', 'DATE'),
 (for, 'O', ''),
 (abusing, 'O', ''),
 (its, 'O', ''),
 (power, 'O', ''),
 (in, 'O', ''),
 (the, 'O', ''),
 (mobile, 'O', ''),
 (phone, 'O', ''),
 (market, 'O', ''),
 (and, 'O', ''),
```

"B" means the token begins an entity
"I" means it is inside an entity
"O"means it is outside an entity
and "" means no entity tag is set.

Extracting named entity from an article

- Extracting named entities from a New York Times article,— "F.B.I. Agent Peter Strzok, Who Criticized Trump in Texts, Is Fired."

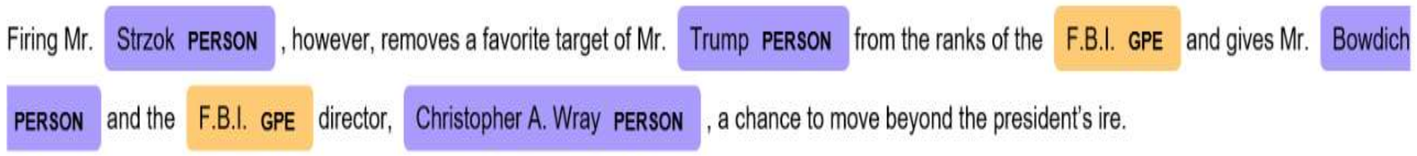
- There are 188 entities in the article and they are represented as 10 unique labels:

```
from bs4 import BeautifulSoup
import requests
import re

def url_to_string(url):
    res = requests.get(url)
    html = res.text
    soup = BeautifulSoup(html, 'html5lib')
    for script in soup(["script", "style", 'aside']):
        script.extract()
    return " ".join(re.split(r'[\n\t]+', soup.get_text()))

ny_bb =
url_to_string('https://www.nytimes.com/2018/08/13/us/politics/peter-
strzok-fired-fbi.html?
hp&action=click&pgtype=Homepage&clickSource=story-
heading&module=first-column-region&region=top-news&WT.nav=top-news')
article = nlp(ny_bb)
len(article.ents)

labels = [x.label_ for x in article.ents]
Counter(labels)
```

Output

```
displacy.render(nlp(str(sentences[20])), jupyter=True, style='ent')
```

```
Counter({'CARDINAL': 5,
         'DATE': 29,
         'EVENT': 1,
         'GPE': 35,
         'LOC': 1,
         'NORP': 5,
         'ORDINAL': 1,
         'ORG': 26,
         'PERSON': 84,
         'WORK_OF_ART': 1})
```



Firing Mr. Strzok PERSON , however, removes a favorite target of Mr. Trump PERSON from the ranks of the F.B.I. GPE and gives Mr. Bowdich PERSON and the F.B.I. GPE director, Christopher A. Wray PERSON , a chance to move beyond the president's ire.

The above are three most frequent tokens.

```
items = [x.text for x in article.ents]
Counter(items).most_common(3)
```

```
[('Strzok', 32), ('F.B.I.', 17), ('Trump', 10)]
```

- spaCy’s built-in [displaCy visualizer](), here’s what the above sentence and its dependencies look like:

```
displacy.render(nlp(str(sentences[20])), style='dep', jupyter =
True, options = {'distance': 120})
```

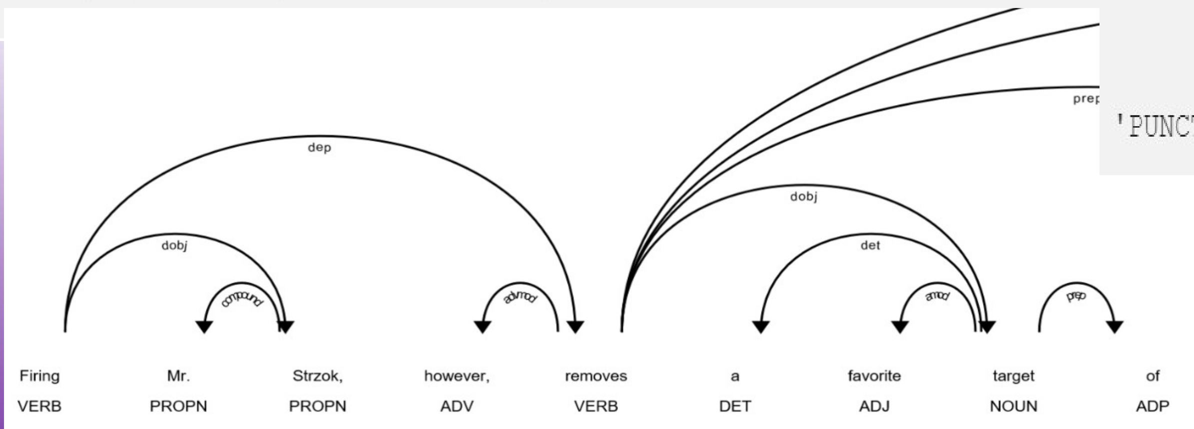


Figure 14

- Next, we verbatim, extract part-of-speech and lemmatize this sentence.

```
[(x.orth_,x.pos_, x.lemma_) for x in [y
                                       for y
                                       in nlp(str(sentences[20]))
                                       if not y.is_stop and y.pos_ !=
'PUNCT']]
```

Result of NER

```
dict([(str(x), x.label_) for x in nlp(str(sentences[20])).ents])
```

```
{'Bowdich': 'PERSON',
 'Christopher A. Wray': 'PERSON',
 'F.B.I.': 'GPE',
 'Strzok': 'PERSON',
 'Trump': 'PERSON'}
```

```
print([(x, x.ent_iob_, x.ent_type_) for x in sentences[20]])
```

```
[(Firing, 'O', ''), (Mr., 'O', ''), (Strzok, 'B', 'PERSON'), (,, 'O', ''), (however, 'O', ''), (,, 'O', ''), (removes, 'O',
''), (a, 'O', ''), (favorite, 'O', ''), (target, 'O', ''), (of, 'O', ''), (Mr., 'O', ''), (Trump, 'B', 'PERSON'), (from, 'O',
''), (the, 'O', ''), (ranks, 'O', ''), (of, 'O', ''), (the, 'O', ''), (F.B.I., 'B', 'GPE'), (and, 'O', ''), (gives, 'O', ''),
(Mr., 'O', ''), (Bowdich, 'B', 'PERSON'), (and, 'O', ''), (the, 'O', ''), (F.B.I., 'B', 'GPE'), (director, 'O', ''), (,, 'O',
''), (Christopher, 'B', 'PERSON'), (A., 'I', 'PERSON'), (Wray, 'I', 'PERSON'), (,, 'O', ''), (a, 'O', ''), (chance, 'O', ''),
(to, 'O', ''), (move, 'O', ''), (beyond, 'O', ''), (the, 'O', ''), (president, 'O', ''), (’s, 'O', ''), (ire, 'O', ''), (.,
'O', '')]
```

Thank You