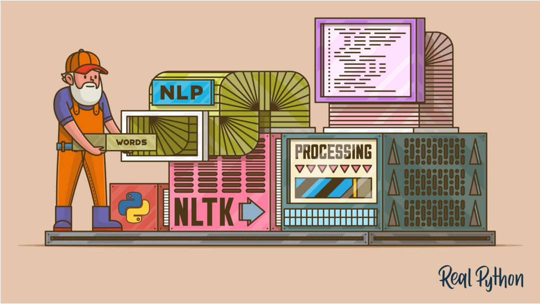
NLP 2

Inteligencia artificial avanzada para la ciencia de datos II Modulo 5 NLP 2

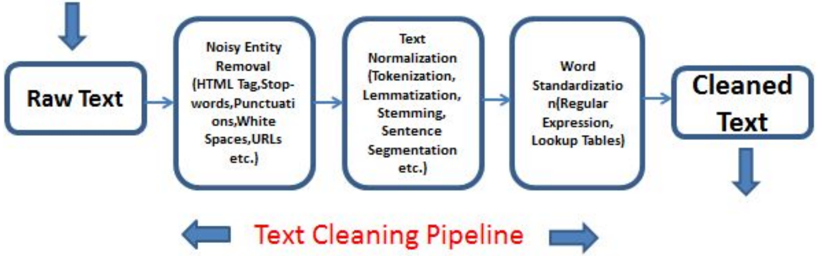
Text processing

Text Processing

Text preprocessing is a crucial step in NLP. Cleaning our text data in order to convert it into a presentable form that is **analyzable and predictable for our task** is known as text preprocessing.



Text normalization is the process by which we prepared our input into a **standard and less noisy language representation**.



Contractions

In some languages and communication tasks it is desired to avoid contractions.

Contractions are combinations or mutations of words in slang, for example:

I'm means **I am**
U means **you**

```
#Expanding Word Contractions

import contractions

s = "I'll, he'll, I'm, can't, won't, aren't, doesn't, haven't"
[contractions.fix(w) for w in s.split()]
```

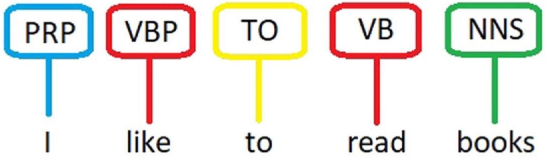
```
['I will,',
 'he will,',
 'I am,',
 'can not,',
 'will not,',
 'are not,',
 'does not,',
 'have not']
```

POS tagging

For some tasks it is desirable to identify and classify our tokens, the process to do so is called "**Part of Speech tagging".**

There are several methods to do it, the simplest one is do in it like in elementary school by **"grammar rules patterns".**

Example:
Article + Noun + verb



NLTK POS

For some tasks it is desirable to identify and classify our tokens, the process to do so is called **“Part of Speech tagging”.**

There are several methods to do it, the simplest one is do in it like in elementary school by **“grammar rules patterns”.**

Example:
Article + Noun + verb

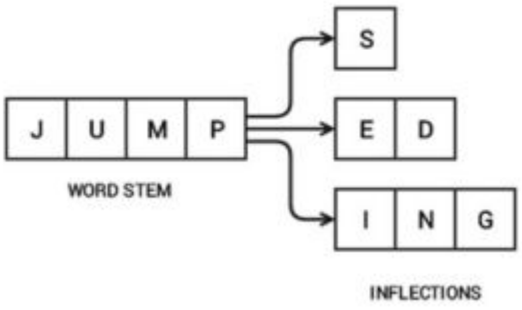
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Terminal File Edit View Search Terminal Help

```
$ python speechTagging.py
[('Whether', 'IN'), ('you', 'PRP'), ("'re", 'VBP'), ('new', 'JJ'), ('to', 'TO'), ('programming'
, 'VBG'), ('or', 'CC'), ('an', 'DT'), ('experienced', 'JJ'), ('developer', 'NN'), (',', ','), (
'it', 'PRP'), ("'s", 'VBZ'), ('easy', 'JJ'), ('to', 'TO'), ('learn', 'VB'), ('and', 'CC'), ('us
e', 'VB'), ('Python', 'NNP'), ('.', '.')]
$
```

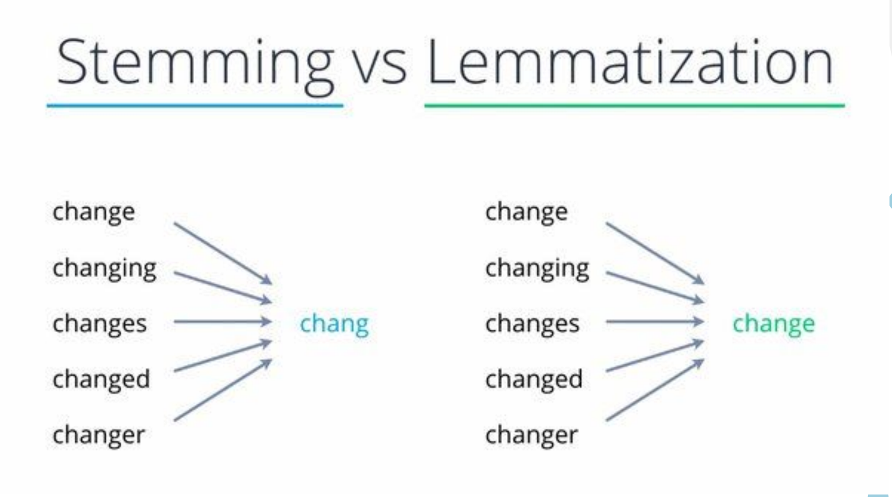
Stemming

Word stems are also known as the base form of a word, and we can create new words by attaching affixes to them in a process known as inflection. Consider the word **JUMP**. You can add affixes to it and form new words like **JUMPS**, **JUMPED**, and **JUMPING**. In this case, the base word **JUMP** is the word stem.



Lemmatization

Lemmatization is very similar to stemming, where we remove word affixes to get to the base form of a word. However, the base form in this case is known as the root word, but not the root stem. The difference being that the root word is **always a lexicographically correct word (present in the dictionary), but the root stem may not be so**. Thus, root word, also known as the lemma, will always be present in the dictionary.



Comparative

Stemming	Lemmatization
Stemming is a process that stems or removes last few characters from a word, often leading to incorrect meanings and spelling.	Lemmatization considers the context and converts the word to its meaningful base form, which is called Lemma.
For instance, stemming the word ‘Caring‘ would return ‘Car‘.	For instance, lemmatizing the word ‘Caring‘ would return ‘Care‘.
Stemming is used in case of large dataset where performance is an issue.	Lemmatization is computationally expensive since it involves look-up tables and what not.

Strategy

One common strategy to find a balance root word process is to define a **pareto** principle analysis.

Most of the used words in real life represents over the 20% or less of total language vocabulary.

If we use a **lemmatizer** process of the most used words and the rest a **stemmer**, we might guarantee a optimal model between meaning and cost.

Lemmatizer

The most common language tokens (around 20%)

+

Stemmer

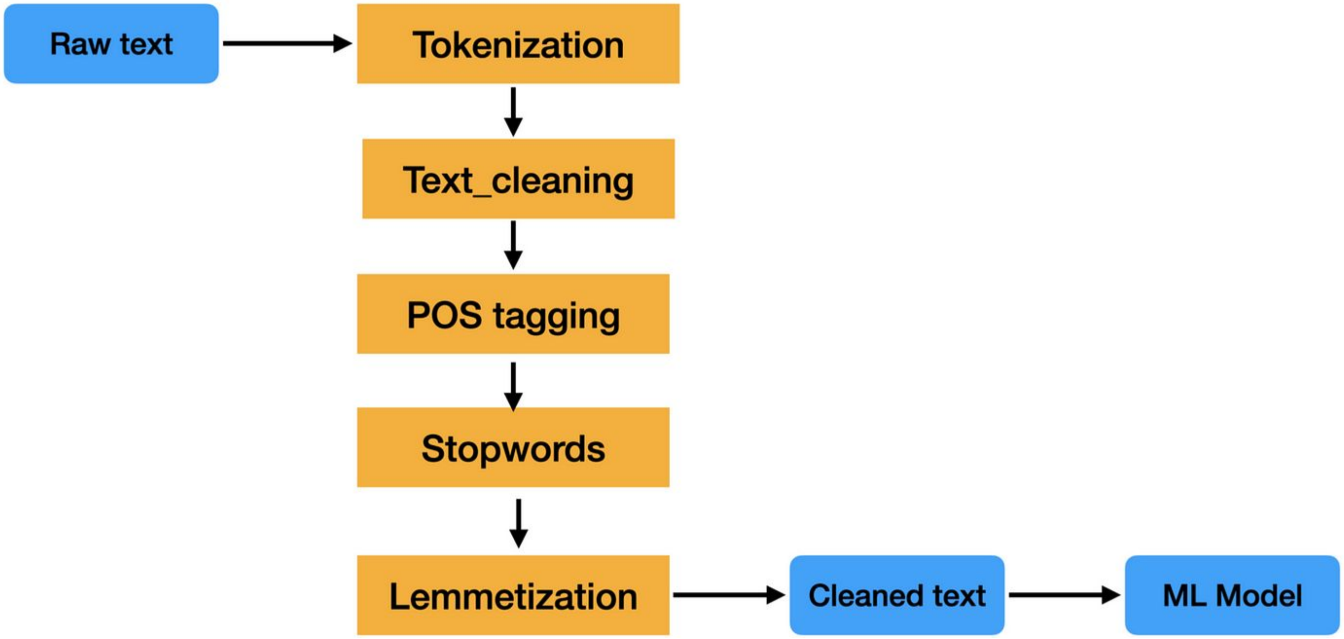
Not common words, not general usage or no so ambiguous.

→

Balance understanding

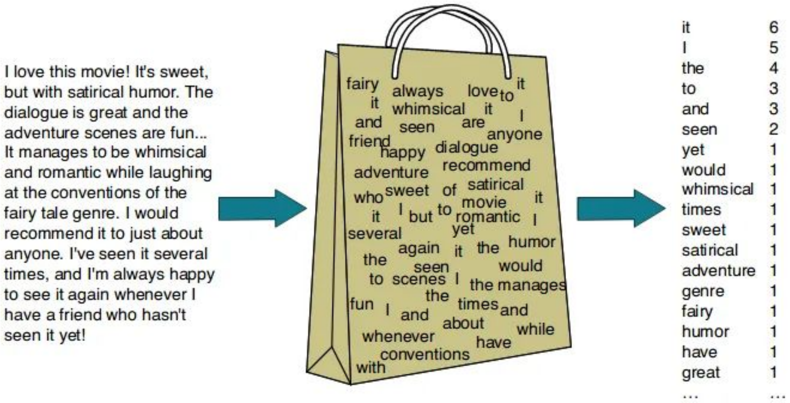
The goal is to reach most of the root words in a cheap computational cost.

Full pipeline



Bag of words model (BoW)

Bag-of-words(BoW) is a statistical language model used to analyze text and documents based on word count. The model does **not account for word order within a document**.



Vectorization

Feature extraction (or vectorization) in NLP is the process of **turning text into a BoW vector**, in which features are unique words and feature values are word counts.

```
# given the following features dictionary mapping:
{'are':0,
 'many':1,
 'success':2,
 'there':3,
 'to':4,
 'ways':5}

# "many success ways" could be represented
# as the following BoW vector:
[0, 1, 1, 0, 0, 1]
```

Ml models

The vector might be useful as features dimensions in order to train machine learning algorithms.

However this simple model persists the probability problem that assumes words (features) **doesn't have an order or a dependant probability.**

"Cats are horrible animals, I hate them"

Hate	like	unlike	horrible	love
1	0	0	1	0

"Cats !! I think persons who hate this animals are horrible"

Hate	like	unlike	horrible	love
1	0	0	1	0

Feature vocabulary

Hate	like	unlike	horrible	love

N-grams

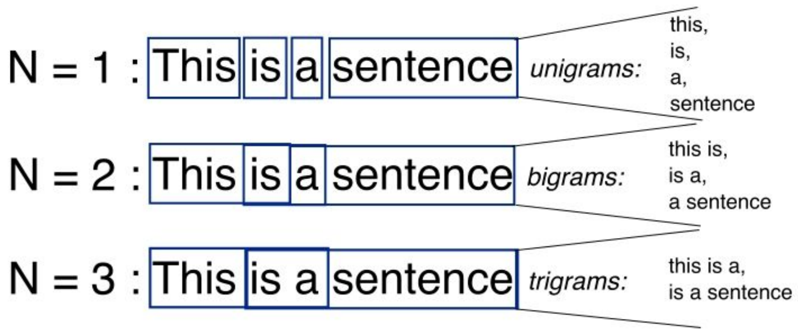
N-grams provides a less independent probability based not only in words.

A gram is a sequence of words where N is the number of words registered in the event.

Example: x = **“I am your father”**

In bi-gram tokenizer (two words):**[I am, am your, your father”]**

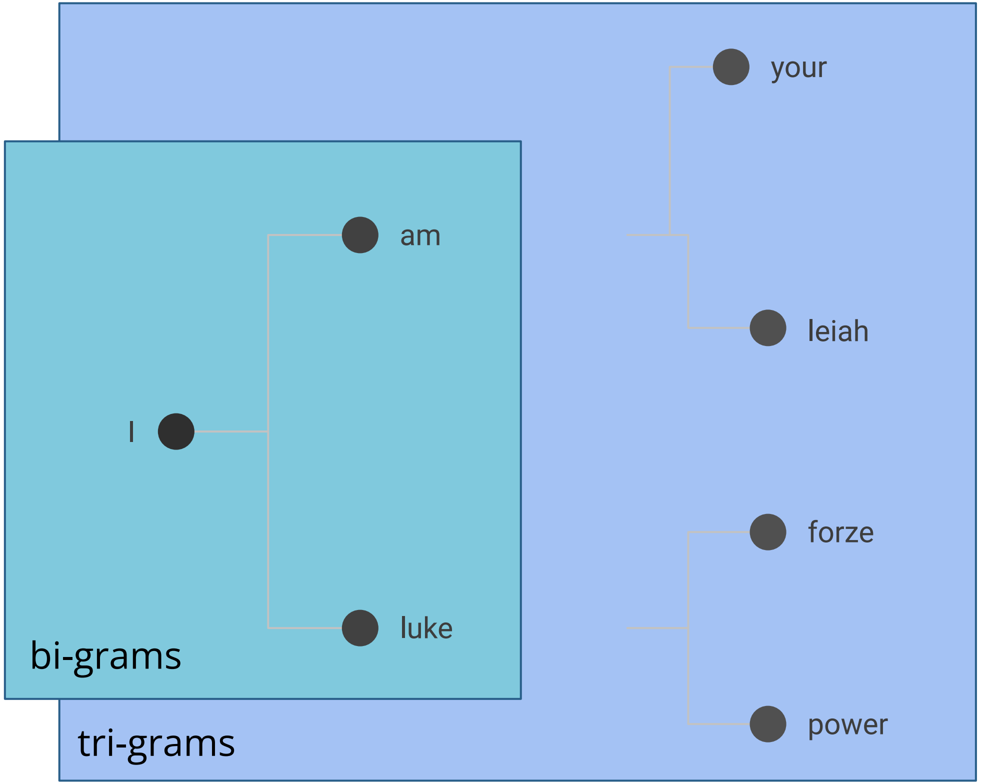
In tri-gram tokenizer (three words):**[I am your,am your father”]**



p(x=”i am”) = p(I am)/p(bi-gram corpus)

p(x=”i am your”) = p(I am yout)/p(three-gram corpus)

N-grams



N-grams probability/smoothing

The purpose of smoothing is to prevent a language model from assigning zero probability to unseen events.

That is needed because in some cases, words can appear in the same context, but they didn't in your train set. Smoothing is a quite rough trick to make your model more generalizable and realistic by setting a default probability "not zero"

By the Chain Rule we can decompose a joint probability, e.g. $P(w_1,w_2,w_3)$ as follows

$$P(w_1,w_2, \ldots,w_n) = P(w_n|w_{n-1},w_{n-2},\ldots,w_1)\ \underbrace{P(w_{n-1}|w_{n-2}, \ldots,w_1) \ldots P(w_2|w_1)\ P(w_1)}_{P(w_{n-1}, \ldots,w_1)}$$

$$P(w_1^n)=\prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

Compute the product of component conditional probabilities?

– P(the mythical unicorn) = P(the) * P(mythical|the) * P(unicorn|the mythical)

For bigrams then, the probability of a sequence is just the product of the conditional probabilities of its bigrams, e.g.

P(the,mythical,unicorn) = P(unicorn|mythical) P(mythical|the) P(the|<start>)

Thanks

Do you have any questions?

emmanuel.paez@tec.mx
Slack #module-5-nlp-1