

Demonstração - Aula 1

✓ Técnicas de pré-processamento de textos

✓ DOCUMENTO / CORPUS

```
import pandas as pd

# Documento e Corpus
df = pd.DataFrame({
    'text': [
        'Sobre MBA? Eu gostei muito do MBA da FIAP',
        'O MBA da FIAP pode melhorar, não gostei muito'
    ],
    'class': [
        'positivo',
        'negativo'
    ]
})

df.head()
```



	text	class
0	Sobre MBA? Eu gostei muito do MBA da FIAP	positivo
1	O MBA da FIAP pode melhorar, não gostei muito	negativo

✓ TOKENIZAÇÃO

```
# aplica tokenização em uma string comprando o módulo de tokenização do NLTK e o split do Python
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
```

```
nome = 'Vamos aprender o que é processamento de linguagem natural.'
```

```
word_tokenize(nome)
```

```
print(word_tokenize(nome))
print(nome.split())
```



```
['Vamos', 'aprender', 'o', 'que', 'é', 'processamento', 'de', 'linguagem', 'natural', '.']
['Vamos', 'aprender', 'o', 'que', 'é', 'processamento', 'de', 'linguagem', 'natural.']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Package punkt_tab is already up-to-date!
```

NLTK = Natural Language Tool Kit

```
# aplica tokenização em uma lista
texto = ['Vamos aprender o que é processamento de linguagem natural.', 'um dois, três']
type(texto)
```

```
# usando o split
print(texto[1].split())
[t.split() for t in texto]
```

```
⇒ ['um', 'dois,', 'três']
  [['Vamos',
    'aprender',
    'o',
    'que',
    'é',
    'processamento',
    'de',
    'linguagem',
    'natural.'],
   ['um', 'dois,', 'três']]
```

```
#from nltk.tokenize import word_tokenize
#import nltk
#nltk.download('punkt')
```

```
[word_tokenize(t) for t in texto]
```

```
⇒ [['Vamos',
    'aprender',
    'o',
    'que',
    'é',
    'processamento',
    'de',
    'linguagem',
    'natural',
    '.'],
   ['um', 'dois', ',', 'três']]
```

```
# aplica tokenização em um dataframe
#from nltk.tokenize import word_tokenize
#import nltk
#nltk.download('punkt')
```

```
print(df)
print(df.text.apply(word_tokenize))
```

```
df['tokens'] = df.text.apply(word_tokenize)
```

```
⇒
```

	text	class
0	Sobre MBA? Eu gostei muito do MBA da FIAP	positivo
1	O MBA da FIAP pode melhorar, não gostei muito	negativo
0	[Sobre, MBA, ?, Eu, gostei, muito, do, MBA, da...	
1	[O, MBA, da, FIAP, pode, melhorar, ,, não, gos...	

Name: text, dtype: object

```
# em uma sentença (representada pelo ponto final)
from nltk.tokenize import sent_tokenize, word_tokenize

s = 'Vamos aprender o que é processamento de linguagem natural.\nBora, vamos sim'

print(sent_tokenize(s))
print([word_tokenize(t) for t in sent_tokenize(s)])
```

```
➞ ['Vamos aprender o que é processamento de linguagem natural.', 'Bora, vamos sim']
[['Vamos', 'aprender', 'o', 'que', 'é', 'processamento', 'de', 'linguagem', 'natural', '.'], [
```

```
#from nltk.tokenize import wordpunct_tokenize (separa por qualquer pontuação, inclusive números R$)
```

✓ UNIGRAMA

Documento/Texto: "um dois três quatro"

- Unigrama ["um","dois","três","quatro"], temos 4 unigramas.
- Bigrama ["um dois","dois três","três quatro"], temos 3 bigramas.
- Trigrama ["um dois três","dois três quatro"], temos 2 trigramas.
- 4-grama ["um dois três quatro"], temos um 4-grama.

df.text



text

```
0    Sobre MBA? Eu gostei muito do MBA da FIAP
1    O MBA da FIAP pode melhorar, não gostei muito
```

dtype: object

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
vect = CountVectorizer(ngram_range=(1,1))
vect.fit(df.text)
text_vect = vect.transform(df.text)
```

```
#print(pd.DataFrame(text_vect.A, columns=vect.get_feature_names_out()).to_string())
print(pd.DataFrame(text_vect.toarray(), columns=vect.get_feature_names_out()).to_string())
```



```
da do eu fiap gostei mba melhorar muito não pode sobre
0  1  1  1  1      1  2      0  1  0  0  1
1  1  0  0  1      1  1      1  1  1  1  0
```

text_vect.shape[1]



11

```
len(vect.get_feature_names_out())
```

↗ 11

▼ BIGRAMA

```
df.text
```

↗

text

0 Sobre MBA? Eu gostei muito do MBA da FIAP

1 O MBA da FIAP pode melhorar, não gostei muito

dtype: object

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
vect = CountVectorizer(ngram_range=(2,2))
```

```
vect.fit(df.text)
```

```
text_vect = vect.transform(df.text)
```

```
#print(pd.DataFrame(text_vect.A, columns=vect.get_feature_names_out()).to_string())
```

```
print(pd.DataFrame(text_vect.toarray(), columns=vect.get_feature_names_out()).to_string())
```

↗

	da fiap	do mba	eu gostei	fiap pode	gostei muito	mba da	mba eu	melhorar	não	muito do
0	1	1	1	0	1	1	1	0	0	1
1	1	0	0	1	1	1	0	1	1	0

▼ TRIGRAMA

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
vect = CountVectorizer(ngram_range=(3,3))
```

```
vect.fit(df.text)
```

```
text_vect = vect.transform(df.text)
```

```
#print(pd.DataFrame(text_vect.A, columns=vect.get_feature_names_out()).T.to_string())
```

```
print(pd.DataFrame(text_vect.toarray(), columns=vect.get_feature_names_out()).T.to_string())
```

↗

	0	1
da fiap pode	0	1
do mba da	1	0
eu gostei muito	1	0
fiap pode melhorar	0	1
gostei muito do	1	0
mba da fiap	1	1
mba eu gostei	1	0
melhorar não gostei	0	1
muito do mba	1	0
não gostei muito	0	1

```
pode melhorar não 0 1
sobre mba eu 1 0
```

✓ REGEX

```
email = "professor@gmail.com"
```

```
# função split do Python
email.split("@")[1].split(".")[0]
```

```
#"dourado@gmail.com".split("@")[1].split('.')[0]
```

```
⇒ 'gmail'
```

```
"professor @ gmail . com".split("@")[1].split(".")[0]
```

```
⇒ ' gmail '
```

```
# importa pacote de regular expression
import re
```

```
regex = r"(?<=@)[^.]+(?:=\.)"
re.findall(regex, email)
```

```
⇒ ['gmail']
```

```
# mede o tempo de execução de um trecho de código
import timeit
```

```
timeit.Timer(
    're.findall(regex, "professor@gmail.com")',
    'import re; regex = r"(?<=@)[^.]+(?:=\.)"'
).repeat(2)
```

```
⇒ [1.1725736150000046, 1.0715074929999986]
```

```
import timeit
```

```
timeit.Timer(
    '"professor@gmail.com".split("@")[1].split(".")[0]'
).repeat(2)
```

```
⇒ [0.27291263900001184, 0.26688884099999655]
```

```
import re
print(email)

re.findall(r'.',email)
re.findall(r'[a-z]',email)
re.findall(r'[0-9]',email)
re.findall(r'.*',email)
re.findall(r'$',email)

re.findall(r'[a-z]+',email)

re.findall(r'^.',email)
re.findall(r'^d',email)

➡ professor@gmail.com
[]
```

```
import re
rex = re.compile('\w+') #qualquer caracter alfanumérico - compilado
bandas = 'Queen, Aerosmith & Beatles'
print (bandas, '->', rex.findall(bandas))
phone = "2004-959-559 # This is Phone Number"
num = re.sub('#.*$', "", phone) #elimina tudo após #
print ("Phone Num : ", num)
num = re.sub(r'\D', "", phone)# só deixa número
print ("Phone Num : ", num)
```

```
➡ Queen, Aerosmith & Beatles -> ['Queen', 'Aerosmith', 'Beatles']
Phone Num : 2004-959-559
Phone Num : 2004959559
```

Caracteres ou metacaracteres

```
meta - O que faz?
-----
. - Qualquer caractere
[] - Lista de caracteres
[^] - Lista negada
? - Anterior pode existir ou não
.* - Qualquer coisa
{x} - Anterior aparece x vezes
$ - Fim da linha
+ - Anterior ao menos uma vez
(xy) - Cria grupos
^ - Começo da linha
\ - escapa o meta (ignora)
| - ou
```

Teste seu código aqui:

<https://regex101.com/>

import this



The Zen of Python, by Tim Peters

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than *right* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!
```

Bonito é melhor que feio.
Explícito é melhor que implícito.
Simples é melhor que Complexo.
Complexo é melhor que complicado.
Achatado é melhor que aninhado.
Disperso é melhor que compacto.
Legibilidade conta.
Casos especiais não são especiais o suficiente para quebrar as regras.
Apesar de praticidade vencer a pureza.
Erros nunca devem passar despercebidos.
A menos que passem explicitamente "despercebidos".
Diante de ambiguidades, recuse a tentação de deduzir.
Deve haver uma --e preferencialmente só uma-- maneira fácil de fazer isto.
Apesar de que a maneira não pode ser óbvia de primeira, a não ser que você seja "asiático".
Agora é melhor do que nunca.
Porém, muitas vezes nunca é melhor do que *agora*.
Se a implementação é difícil de explicar, é uma péssima ideia.
Se a implementação é fácil de explicar, pode ser uma boa ideia.
Namespaces são uma grande ideia gritante -- vamos fazer mais dessas!

✓ STOPWORDS

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

```
# lista de stopwords do NLTK
stops = nltk.corpus.stopwords.words('english')
```

```
⇒ [nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

```
len(stops)
```

```
⇒ 198
```

```
# lista as 10 primeiras stopwords
stops[:10]
```

```
⇒ ['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an']
```

```
# remove uma stopword da lista
stops.pop(3)
```

```
⇒ 'after'
```

```
# lista as 10 primeiras stopwords
stops[:10]
```

```
⇒ ['a', 'about', 'above', 'again', 'against', 'ain', 'all', 'am', 'an', 'and']
```

```
# podemos criar nossa propria lista
stops = stops + ["mba", "fiap"]
```

```
# lista todas as stopwords
stops
```

```
⇒ ['a',
    'about',
    'above',
    'after',
    'again',
    'against',
    'ain',
    'all',
    'am',
    'an',
    'and',
    'any',
    'are',
    'aren',
    "aren't",
    'as',
    'at',
    'be',
    'because',
    'been',
    'before',
    'being',
    'below',
```



```
'between',  
'both',  
'but',  
'by',  
'can',  
'couldn',  
'couldn't',  
'd',  
'did',  
'didn',  
'didn't',  
'do',  
'does',  
'doesn',  
'doesn't',  
'doing',  
'don',  
'don't',  
'down',  
'during',  
'each',  
'few',  
'for',  
'from',  
'further',  
'had',  
'hadn',  
'hadn't',  
'has',  
'hasn',  
'hasn't',  
'have',  
'haven',  
'haven't',  
've',  
's'
```

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

```
➞ [nltk_data] Downloading package stopwords to /root/nltk_data...  
[nltk_data] Package stopwords is already up-to-date!  
True
```

```
print(nltk.corpus.stopwords.words('portuguese')[:10])
```

```
➞ ['a', 'à', 'ao', 'aos', 'aquela', 'aquelas', 'aquele', 'aqueles', 'aquilo', 'as']
```

```
len(nltk.corpus.stopwords.words('portuguese'))
```

```
➞ 207
```

```
# aplicando a utilização das stopwords
from sklearn.feature_extraction.text import CountVectorizer

stops = nltk.corpus.stopwords.words('portuguese') + ["fiap", "mba"]

vect = CountVectorizer(ngram_range=(1,1), stop_words=stops)
vect.fit(df.text)
text_vect = vect.transform(df.text)

#print(pd.DataFrame(text_vect.A, columns=vect.get_feature_names_out()).to_string())
print(pd.DataFrame(text_vect.toarray(), columns=vect.get_feature_names_out()).to_string())
```



```
gostei  melhorar  pode  sobre
0      1          0      0      1
1      1          1      1      0
```

✓ NORMALIZAÇÃO DE TEXTOS

```
print(df)

# aplicando a utilização das stopwords
from sklearn.feature_extraction.text import CountVectorizer

vect = CountVectorizer(ngram_range=(1,1))
vect.fit(df.text)
text_vect = vect.transform(df.text)

#print(pd.DataFrame(text_vect.A, columns=vect.get_feature_names_out()).to_string())
print(pd.DataFrame(text_vect.toarray(), columns=vect.get_feature_names_out()).to_string())
```



```
da  do  eu  fiap  gostei  mba  melhorar  muito  não  pode  sobre
0   1   1   1     1        1   2           0     1   0     0     1
1   1   0   0     1        1   1           1     1   1     1     0
```

Importante: note que intrinsicamente o CountVectorizer já realiza um processo de normalização, passando as palavras para minúscula por padrão. Acesse a documentação e veja [link](#)

```
vect.get_params()

{'analyzer': 'word',
 'binary': False,
 'decode_error': 'strict',
 'dtype': numpy.int64,
 'encoding': 'utf-8',
 'input': 'content',
 'lowercase': True,
 'max_df': 1.0,
 'max_features': None,
 'min_df': 1,
 'ngram_range': (1, 1),
 'preprocessor': None,
 'stop_words': None,
 'strip_accents': None,
 'token_pattern': '(?u)\\b\\w\\w+\\b',
```

```
'tokenizer': None,  
'vocabulary': None}
```

Vamos aplicar o processo de normalização mais completo por uma função mais para frente.

✓ Stemmer (Stemização)

```
from nltk.stem import PorterStemmer
```

```
ps = PorterStemmer()  
exemplos = ["connection", "connections", "connective", "connecting", "connected"]  
print(exemplos)
```

```
for word in exemplos:  
    print(ps.stem(word))
```

```
➡ ['connection', 'connections', 'connective', 'connecting', 'connected']  
connect  
connect  
connect  
connect  
connect  
connect
```

```
# Outro exemplo  
ps = PorterStemmer()  
exemplos = ["go", "going", "goes", "gone", "went"]  
print(exemplos)
```

```
for word in exemplos:  
    print(ps.stem(word))
```

```
➡ ['go', 'going', 'goes', 'gone', 'went']  
go  
go  
goe  
gone  
went
```

RSLP = Removedor de Suffixos da Língua Portuguesa

```
# Stemização
from nltk.stem import PorterStemmer
from nltk.stem.rslp import RSLPStemmer
import nltk
nltk.download('rslp')

doc = ["pedra", "pedreira", "pedreiro"]
print(doc)

ps = PorterStemmer()
rslp = RSLPStemmer()

for word in doc:
    print(ps.stem(word), ' - ', rslp.stem(word))
```

```
➡ ['pedra', 'pedreira', 'pedreiro']
pedra - pedr
pedreira - pedr
pedreiro - pedr
[nltk_data] Downloading package rslp to /root/nltk_data...
[nltk_data] Package rslp is already up-to-date!
```

▼ Aplicar Stemmer em uma frase

```
import pandas as pd

df = pd.DataFrame({
    'text': [
        'Sobre MBA? Eu gostei muito do MBA da FIAP',
        'O MBA da FIAP pode melhorar, não gostei muito'
    ],
    'class': [
        'positivo',
        'negativo'
    ]
})

df.head()
```

```
➡
```

	text	class
0	Sobre MBA? Eu gostei muito do MBA da FIAP	positivo
1	O MBA da FIAP pode melhorar, não gostei muito	negativo

```
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')

df['tokens'] = df.text.apply(word_tokenize)
df['tokens']
```

```
➡ [nltk_data] Downloading package punkt to /root/nltk_data...  
[nltk_data] Package punkt is already up-to-date!
```

tokens

```
0 [Sobre, MBA, ?, Eu, gostei, muito, do, MBA, da...  
1 [O, MBA, da, FIAP, pode, melhorar, ,, não, gos...
```

dtype: object

```
from nltk.stem.rslp import RSLPStemmer  
import nltk  
nltk.download('rslp')
```

```
tokens = df.tokens[0]  
tokens = tokens + df.tokens[1]
```

```
rslp = RSLPStemmer()
```

```
for tok in tokens:  
    #print('Original: %s \t\t RSLPStemmer: %s' % (tok, ps.stem(tok), rslp.stem(tok)))  
    print(f'Original: {tok:{11}} RSLPStemmer: {rslp.stem(tok):{10}}')
```

```
➡ Original: Sobre      RSLPStemmer: sobr  
Original: MBA         RSLPStemmer: mba  
Original: ?           RSLPStemmer: ?  
Original: Eu          RSLPStemmer: eu  
Original: gostei      RSLPStemmer: gost  
Original: muito       RSLPStemmer: muit  
Original: do          RSLPStemmer: do  
Original: MBA         RSLPStemmer: mba  
Original: da          RSLPStemmer: da  
Original: FIAP        RSLPStemmer: fiap  
Original: O           RSLPStemmer: o  
Original: MBA         RSLPStemmer: mba  
Original: da          RSLPStemmer: da  
Original: FIAP        RSLPStemmer: fiap  
Original: pode        RSLPStemmer: pod  
Original: melhorar    RSLPStemmer: melhor  
Original: ,           RSLPStemmer: ,  
Original: não         RSLPStemmer: não  
Original: gostei      RSLPStemmer: gost  
Original: muito       RSLPStemmer: muit  
[nltk_data] Downloading package rslp to /root/nltk_data...  
[nltk_data] Package rslp is already up-to-date!
```

O Porter foi criado para o inglês e o RSLP para o português

✓ Quantos unigramas existem após aplicar Stemmer?

```
' '.join(['Professor', 'MBA'])
```

```
➡ 'Professor MBA'
```

```
from nltk.stem.rslp import RSLPStemmer
from sklearn.feature_extraction.text import CountVectorizer
```

```
rslp = RSLPStemmer()

# aplica o stemmer no database
def stem_pandas(line):
    return ' '.join([rslp.stem(token) for token in line])

df['stemmer'] = df.tokens.apply(stem_pandas)

df.stemmer.head()
```



stemmer

```
0    sobr mba ? eu gost muit do mba da fiap
1    o mba da fiap pod melhor , não gost muit
```

dtype: object

```
import nltk

vect = CountVectorizer(ngram_range=(1,1))
vect.fit(df.stemmer)
text_vect = vect.transform(df.stemmer)

print('UNIGRAMAS', text_vect.shape[1])
#print(pd.DataFrame(text_vect.A, columns=vect.get_feature_names_out()).to_string())
print(pd.DataFrame(text_vect.toarray(), columns=vect.get_feature_names_out()).to_string())

#print(text_vect.shape[0])
#print(text_vect.shape[1])
```



```
UNIGRAMAS 11
      da  do  eu  fiap  gost  mba  melhor  muit  não  pod  sobr
0     1   1   1     1     1     2         0     1     0     0     1
1     1   0   0     1     1     1         1     1     1     1     0
```

Unigramas sem aplicar o stemmer

```
vect = CountVectorizer(ngram_range=(1,1))
vect.fit(df.text)

text_vect = vect.transform(df.text)

print('UNIGRAMAS com STOPWORDS', text_vect.shape[1])
```



```
UNIGRAMAS com STOPWORDS 11
```

Não diferença pois o texto não tem muitas variações de palavras que possam ser reduzidas

Unigramas sem aplicar o stemmer e removendo stopwords

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

```
stopwords = nltk.corpus.stopwords.words('portuguese')  
vect = CountVectorizer(ngram_range=(1,1), stop_words=stopwords)  
vect.fit(df.text)
```

```
text_vect = vect.transform(df.text)
```

```
print('UNIGRAMAS sem STOPWORDS', text_vect.shape[1])
```

```
⇒ UNIGRAMAS sem STOPWORDS 6  
[nltk_data] Downloading package stopwords to /root/nltk_data...  
[nltk_data] Package stopwords is already up-to-date!
```

Outra função de stematização do NLTK

```
from nltk.stem import SnowballStemmer
```

```
print(" ".join(SnowballStemmer.languages)) # See which languages are supported
```

```
⇒ arabic danish dutch english finnish french german hungarian italian norwegian porter portuguese
```



```
stemmer = SnowballStemmer("portuguese") # Escolha a linguagem
```

```
palavras = ['pedra', 'pedreira', 'criar']
```

```
for p in palavras:  
    print(stemmer.stem(p)) # Stem a palavra
```

```
⇒ pedr  
   pedreir  
   cri
```

Simplificando o processo de Stemização (RSLP)

```
import pandas as pd
```

```
df = pd.DataFrame({  
    'text': [  
        'Sobre MBA? Eu gostei muito do MBA da FIAP',  
        'O MBA da FIAP pode melhorar, não gostei muito'  
    ],  
    'class': [  
        'positivo',  
        'negativo'  
    ]  
})
```

```
df.head()
```



	text	class
--	------	-------

0	Sobre MBA? Eu gostei muito do MBA da FIAP	positivo
1	O MBA da FIAP pode melhorar, não gostei muito	negativo

```
from nltk.stem.rslp import RSLPStemmer
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('rslp')
```

```
rslp = RSLPStemmer()
```

```
# aplica o stemmer no database
```

```
def stem_pandas(text):
    line = word_tokenize(text)
    return ' '.join([rslp.stem(token) for token in line])
```

```
df['stemmer'] = df.text.apply(stem_pandas)
df
```



```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Package punkt_tab is already up-to-date!
[nltk_data] Downloading package rslp to /root/nltk_data...
[nltk_data] Package rslp is already up-to-date!
```

	text	class
--	------	-------

	text	class	stemmer
--	------	-------	---------

0	Sobre MBA? Eu gostei muito do MBA da FIAP	positivo	sobr mba ? eu gost muit do mba da fiap
1	O MBA da FIAP pode melhorar, não gostei muito	negativo	o mba da fiap pod melhor , não gost muit

Exercício 2

✓ Lemmatizer (Lematização)

com NLTK


```

import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer

examples = [
    "go","going",
    "goes","gone","went"
]

wnl = WordNetLemmatizer()

for word in examples:
    print(wnl.lemmatize(word, 'v'))

```

 [nltk_data] Downloading package wordnet to /root/nltk_data...
 [nltk_data] Downloading package omw-1.4 to /root/nltk_data...
 go
 go
 go
 go
 go


Importante: O NLTK só tem suporte para inglês na lematização!

com Spacy

```

#!pip install spacy
!python -m spacy download pt_core_news_sm --quiet

```

 13.0/13.0 MB 47.9 MB/s eta 0:00:00
 ✓ Download and installation successful
 You can now load the package via `spacy.load('pt_core_news_sm')`
 ⚠ Restart to reload dependencies
 If you are in a Jupyter or Colab notebook, you may need to restart Python in order to load all the package's dependencies. You can do this by selecting the 'Restart kernel' or 'Restart runtime' option.


```

import spacy
nlp = spacy.load('pt_core_news_sm')

```

```
doc = nlp('contrato contratou')
```

```
[token.lemma_ for token in doc]
```

 ['contrato', 'contratar']

```

def lema(texto):
    doc = nlp(texto)
    return " ".join([token.lemma_ for token in doc])

```

```
lema('olá contratou!')
```

 'olá contratar !'

```
doc[1].text
doc[1].lemma_
```

➡ 'contratar'

Comece a programar ou [gere código](#) com IA.

▼ Material complementar

▼ PART-OF-SPEECH TAGGER (POS-Tag)

▼ com o Spacy

```
#!pip install spacy
#!python -m spacy download pt_core_news_sm
```

```
import spacy
nlp = spacy.load('pt_core_news_sm')
```

```
doc = nlp(u'Ayrton Senna foi o melhor piloto de Fórmula 1 que já existiu')
```

```
print([token.orth_ for token in doc])
```

➡ ['Ayrton', 'Senna', 'foi', 'o', 'melhor', 'piloto', 'de', 'Fórmula', '1', 'que', 'já', 'existiu']

```
[(token.orth_, token.pos_) for token in doc]
```

➡

```
[('Ayrton', 'PROPN'),
 ('Senna', 'PROPN'),
 ('foi', 'AUX'),
 ('o', 'DET'),
 ('melhor', 'ADJ'),
 ('piloto', 'NOUN'),
 ('de', 'ADP'),
 ('Fórmula', 'PROPN'),
 ('1', 'PROPN'),
 ('que', 'PRON'),
 ('já', 'ADV'),
 ('existiu', 'VERB')]
```

De/Para do POS Tag com o tagset='universal':

- NOUN (nouns / substantivos)
- VERB (verbs / verbos)
- ADJ (adjectives / adjetivos)
- ADV (adverbs / advérbios)
- PRON (pronouns / pronomes)

- DET (determiners and articles / determinantes e artigos)
- ADP (adpositions - prepositions and postpositions / adições - preposições e postposições)
- NUM (numerals / numerais)
- CONJ (conjunctions / conjunções)
- PRT (particles / partículas)
- . (punctuation marks / sinais de pontuação)
- X (a catch-all for other categories such as abbreviations or foreign words / um exemplo geral para outras categorias, como abreviações ou palavras estrangeiras)

```
print('filtrando apenas verbos: ')
print([token.text for token in doc if token.pos_ == 'VERB'])
```

```
⇒ filtrando apenas verbos:
['existiu']
```

```
print('filtrando apenas verbos e lematizando: ')
print([token.lemma_ for token in doc if token.pos_ == 'VERB'])
```

```
⇒ filtrando apenas verbos e lematizando:
['existir']
```

Desafio:

Crie uma função que aplique a lematização em verbos. Essa função deve receber um texto e ter como output o texto com verbos lematizado. Use o Spacy!

resposta

Outras funcionalidades do Spacy

```
import spacy
nlp = spacy.load('pt_core_news_sm')
```

```
print('identificação de entidades: ')
```

```
doc1 = nlp(u'Machado de Assis um dos melhores escritores do Brasil, \
foi o primeiro presidente da Academia Brasileira de Letras')
```

```
print(doc1.ents)
```

```
⇒ identificação de entidades:
(Machado de Assis, Brasil, Academia Brasileira de Letras)
```

▼ com NLTK

```
# Tokenizxação
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('universal_tagset')
nltk.download('averaged_perceptron_tagger_eng')

doc = word_tokenize("John's big idea isn't all that bad.")
doc_tag = pos_tag(doc)
doc_tag
```



```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data] Unzipping taggers/universal_tagset.zip.
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger_eng.zip.
```

LookupError Traceback (most recent call last)

```
<ipython-input-12-8bc0b86b52d6> in <cell line: 0>()
      8 nltk.download('averaged_perceptron_tagger_eng')
      9
--> 10 doc = word_tokenize("John's big idea isn't all that bad.")
     11 doc_tag = pos_tag(doc)
     12 doc_tag
```

5 frames

```
/usr/local/lib/python3.11/dist-packages/nltk/data.py in find(resource_name, paths)
    577     sep = "*" * 70
    578     resource_not_found = f"\n{sep}\n{msg}\n{sep}\n"
--> 579     raise LookupError(resource_not_found)
    580
    581
```

LookupError:

Resource **punkt_tab** not found.
Please use the NLTK Downloader to obtain the resource:

```
>>> import nltk
>>> nltk.download('punkt_tab')
```

For more information see: <https://www.nltk.org/data.html>

Attempted to load **tokenizers/punkt_tab/english/**

Searched in:

- '/root/nltk_data'
- '/usr/nltk_data'
- '/usr/share/nltk_data'
- '/usr/lib/nltk_data'
- '/usr/share/nltk_data'
- '/usr/local/share/nltk_data'
- '/usr/lib/nltk_data'
- '/usr/local/lib/nltk_data'

```
text1 = nltk.word_tokenize("They refuse to permit us to obtain the refuse permit")
nltk.pos_tag(text1)
```

```
pos_tag(word_tokenize("John's big idea isn't all that bad."),tagset='universal')
```

De/Para do POS Tag com o tagset='universal':

- NOUN (nouns / substantivos)
- VERB (verbs / verbos)

- ADJ (adjectives / adjetivos)
- ADV (adverbs / advérbios)
- PRON (pronouns / pronomes)
- DET (determiners and articles / determinantes e artigos)
- ADP (adpositions - prepositions and postpositions / adições - preposições e postposições)
- NUM (numerals / numerais)
- CONJ (conjunctions / conjunções)
- PRT (particles / partículas)
- . (punctuation marks / sinais de pontuação)
- X (a catch-all for other categories such as abbreviations or foreign words / um exemplo geral para outras categorias, como abreviações ou palavras estrangeiras)

Comece a programar ou gere código com IA.

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('universal_tagset')

import pandas as pd

# Documento e Corpus
df = pd.DataFrame({
    'text': [
        'Sobre MBA? Eu gostei muito do MBA da FIAP',
        'O MBA da FIAP pode melhorar, não gostei muito'
    ],
    'class': [
        'positivo',
        'negativo'
    ]})

df.head()

# Tokenização
from nltk.tokenize import word_tokenize

df['tokens'] = df.text.apply(word_tokenize)
df[["tokens", "text"]]

# Rotular parte do discurso
from nltk.tag import pos_tag

df['tags'] = df.tokens.apply(pos_tag, tagset='universal')

df.tags
df.tags[1][0][1]
```

Importante: O NLTK só tem suporte para inglês com POS-Tagger nativo!

Como resolver isso usando o NLTK?

Podemos criar nosso próprio pos-tag usando o NLTK através de uma base de dados já "tageada" (ex.: Floresta) e treinar um modelo de classificação recursivo do NLTK.

com o NLTK - solução alternativa

▼ Sobre o corpus Floresta

Conhecido como "Floresta Sintática". Conjunto de frases já analisadas sintaticamente e tageadas.

<https://www.linguateca.pt/Floresta/>

<https://www.linguateca.pt/floresta/doc/VISLsymbolset-manual.html>

```
import nltk
nltk.download('floresta')
from nltk.corpus import floresta

floresta.tagged_words()

def simplify_tag(t):
    if "+" in t:
        return t.split("+")[1]
    return t

twords = nltk.corpus.floresta.tagged_words()
twords = [(w.lower(),simplify_tag(t)) for (w,t) in twords]
twords[:10]

print(nltk.corpus.floresta.readme())
```

▼ Default Tagger

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

tags = [tag for (word, tag) in twords]
nltk.FreqDist(tags).max()

raw = 'Esse é um exemplo utilizando o marcador padrão'
tokens = nltk.word_tokenize(raw)
default_tagger = nltk.DefaultTagger('n')
default_tagger.tag(tokens)
```

```

# Contagem das tags e mantendo a estruturas das sentenças do corpus floresta
from nltk.corpus import floresta
from collections import Counter

def simplifica_tag(t):
    if "+" in t:
        return t.split("+")[1]
    return t

counter = Counter()

tag_sents = floresta.tagged_sents()
tag_new_sents = []
for sent in tag_sents:
    new_sent = []
    for (w,t) in sent:
        tag = simplifica_tag(t)
        new_sent.append((w.lower(), tag))
        counter[tag] += 1
    tag_new_sents.append(new_sent)

#tag_sents[0]
#tag_new_sents[0]

#new_sent
tag_new_sents[:2]

counter.most_common(5)

# % dos substantivos
counter.get('n') / sum(counter.values())

```

Treinando o "motor" de classificação

```

tsents = floresta.tagged_sents()
tsents = [[(w.lower(),simplify_tag(t)) for (w,t) in sent] for sent in tsents if sent]
train = tsents[1000:]
test = tsents[:1000]
tsents[1:3]

tsents[1:3]

tagger0 = nltk.DefaultTagger('n')
print(tagger0.evaluate(test))

```

▼ Unigram Tagger

```

tagger1 = nltk.UnigramTagger(train)
print(tagger1.evaluate(test))

```



```
tagger2 = nltk.BigramTagger(train)
print(tagger2.evaluate(test))
```

✓ Combinação de Tagger

```
tagger1 = nltk.UnigramTagger(train, backoff=tagger0)
print('tagger1: ',tagger1.evaluate(test))
tagger2 = nltk.BigramTagger(train, backoff=tagger1)
print('tagger2: ',tagger2.evaluate(test))
```

Documentação dos métodos tag:

- <https://www.nltk.org/api/nltk.tag.html>

Nos baseamos no métodos da documentação abaixo:

- Capítulo 5 - N-Gram Tagging: <http://www.nltk.org/book/ch05.html>
- Exemplo em português: http://www.nltk.org/howto/portuguese_en.html
- Outros corpus tageados: <https://www.nltk.org/book/ch02.html>