



# PYTHON PARA PLN

**Introdução ao spaCy + Embeddings + BERT**

**Roney Lira de Sales Santos** [roneysantos@usp.br](mailto:roneysantos@usp.br)

Prof. Thiago A. S. Pardo

# SPACY

- Biblioteca Python para processamento de textos
  - Escala industrial
- Feito para uso em produção
  - Criação de aplicações que conseguem processar um grande volume de dados
- Versão ~~2.1~~ 3.0!
  - O parser sintático mais rápido do mundo (!!)
  - Acurácia de 92.6%
    - 1% a mais que o melhor parser disponível
- Suporte para mais de 61 linguagens

# SPACY - INSTALAÇÃO

- Guia de instalação completo [aqui](#)
- Compatível com versões ~~2.7/3.6+~~ do Python
  - ~~Uma das poucas bibliotecas que ainda possuem suporte para o Python 2.x~~
- Linux, MacOS e Windows 64-bit
  - Instalação por linha de comando

```
pip install -U spacy
```
- Necessário instalar dados adicionais
  - Parecido com o que fizemos no NLTK

# SPACY - INSTALAÇÃO

- Dados adicionais para lematização

```
pip install -U spacy-lookups-data
```

- Modelo de linguagem

- Para o spaCy conseguir realizar suas funções, é necessário que um modelo de linguagem esteja presente.
- Modelos pré-treinados
  - Entidades Nomeadas
  - Classes gramaticais
  - Dependências sintáticas
- Parecido com os corpúscos que utilizamos como treinamento no NLTK

# SPACY - INSTALAÇÃO

- Modelos de linguagem para o português

```
python -m spacy download pt_core_news_sm
python -m spacy download pt_core_news_md
python -m spacy download pt_core_news_lg
```
- Praticamente todas as atribuições que os modelos mais robustos possuem (exemplo: inglês)
  - Baseado no corpus WikiNER
    - Vetores dos tokens e classes gramaticais
    - Análise de dependência
    - Entidades nomeadas
- Mais detalhes sobre os modelos [aqui](#).

# SPACY - INSTALAÇÃO

- Além do modelo de linguagem padrão do spaCy, é possível criar o seu próprio modelo!
  - Ou usar um pronto, já treinado para algum fim
- Um ótimo guia pode ser encontrado [aqui!](#)
  - Spoiler: inserção de alguns exemplos para treinamento em um código próprio do spaCy

# SPACY - USO

- Para o uso das funções poderosas do spaCy, é preciso entender dois objetos importantes:
  - O objeto **Doc**
  - O objeto **Token**
- Um **Doc** é uma sequência de objetos **Token**
  - Ou seja, um documento com vários tokens manipuláveis
  - Métodos da classe **Doc** levam em consideração a manipulação desses tokens
    - Exemplo: quantidade de tokens no documento
- Um **Token** é o token que aprendemos na aula de NLTK: pode ser uma palavra, uma pontuação, numeral, espaços...

# SPACY - USO

- Assim, antes de qualquer utilização das funções do spaCy, deve-se criar a variável que vai guardar o modelo de linguagem

```
1 import spacy
2
3 nlp = spacy.load("pt_core_news_lg")
4 doc = nlp(palavras) #o texto, não os tokens!
```

- **IMPORTANTE:** no NLTK era utilizado sempre a lista de tokens, mas aqui no spaCy, o parâmetro é sempre a string do texto!
- Portanto, a partir de agora, todas as funções serão provenientes da variável **doc!**



# SPACY - USO

- Bom, aqui vamos começar a usar as funções mais interessantes do spaCy:
  - Tokenização
  - Stemming e Lematizador
  - Etiquetador
  - Entidades Nomeadas
- Utilizaremos o mesmo `corpus` das aulas anteriores
  - Tá [aqui](#), para quem ainda não tem.
- Claro, o spaCy contém várias outras funções!

# SPACY - TOKENIZAÇÃO

- Para recuperar os tokens, basta usar o conceito de *list comprehension*

```
12 tokens = [token for token in doc]
13 print(tokens)
14
```

---

```
[Giants, batem, os, Patriots, no, Super, Bowl, XLII,
, Azarões, acabam, com, a, invencibilidade, de, New, England, e,
ficam, com, o, título, da, temporada,
, 04/02/2008, -, 01h07, m, -, Atualizado, em, 04/02/2008, -,
09h49, m,
```

```
, Com, um, passe, de, Eli, Manning, para, Plaxico, Burress, a,
39, segundos, do, fim, ,, o, New, York, Giants, anotou, o,
touchdown, decisivo, e, derrubou, o, favorito, New, England,
Patriots, por, 17, a, 14, n, este, domingo, ,, em, Glendale, ,,
```

# SPACY - TOKENIZAÇÃO

- Para recuperar os tokens, basta usar o conceito de *list comprehension*

```
12 tokens = [token for token in doc]
13 print(tokens)
14
```

```
[Giants, batem, os, Patriots, no, Super, Bowl, XLII,
, Azarões, acabam, com, a, invencibilidade, de, New, England, e,
ficam, com, o, título, da, temporada,
, 04/02/2008, -, 01h07, m, -, Atualizado, em, 04/02/2008, -,
09h49, m,
```

```
, Com, um, passe, de, Eli, Manning, para, Plaxico, Burrell, a,
39, segundos, do, fim, ,, o, New, York, Giants, anotou, o,
touchdown, decisivo, e, derrubou, o, favorito, New, England,
Patriots, por, 17, a, 14, n, este, domingo, ,, em, Glendale, ,,
```

- Dá pra perceber algumas coisas aqui, concordam?
  - Uma delas: não parece ser uma lista de strings...

# SPACY - TOKENIZAÇÃO

- Para recuperar os tokens, basta usar o conceito de *list comprehension*

```
12 tokens = [token.orth_ for token in doc]
13 print(tokens)
14
```

```
['Giants', 'batem', 'os', 'Patriots', 'no', 'Super', 'Bowl',
'XLII', '\n', 'Azarões', 'acabam', 'com', 'a', 'invencibilidade',
'de', 'New', 'England', 'e', 'ficam', 'com', 'o', 'título', 'da',
'temporada', '\n', '04/02/2008', '-', '01h07', 'm', '-',
'Atualizado', 'em', '04/02/2008', '-', '09h49', 'm', '\n\n',
'Com', 'um', 'passe', 'de', 'Eli', 'Manning', 'para', 'Plaxico',
'Burress', 'a', '39', 'segundos', 'do', 'fim', ',', 'o', 'New',
'York', 'Giants', 'anotou', 'o', 'touchdown', 'decisivo', 'e',
'derrubou', 'o', 'favorito', 'New', 'England', 'Patriots', 'por',
'17', 'a', '14', 'n', 'este', 'domingo', ',', 'em', 'Glendale',
',', 'no', 'Super', 'Bowl', 'XLII', '.', 'O', 'resultado', ',',
'uma', 'das', 'maiores', 'zebras', 'da', 'história', 'do',
'Super', 'Bowl', ',', 'acabou', 'com', 'a', 'temporada',
'perfeita', 'de', 'Tom', 'Brady', 'e', 'companhia', ',', 'que',
```

- Agora sim! Só usar o atributo **orth\_**

# SPACY - TOKENIZAÇÃO

- Retorno com tipos de tokens diferentes:
  - Somente as palavras: **is\_alpha**

```
>>> texto = "Com um passe de Eli Manning para Plaxico Burress a 39 segundos do fim, o New York Giants anotou o touchdown decisivo e derrubou o favorito New England Patriots por 17 a 14 neste domingo."
>>> doc = nlp(texto)
>>> tokens_palavras = [token .orth_ for token in doc if token.is_alpha]
>>> tokens_palavras
['Com', 'um', 'passe', 'de', 'Eli', 'Manning', 'para', 'Plaxico', 'Burress', 'a', 'segundos', 'do', 'fim', 'o', 'New', 'York', 'Giants', 'anotou', 'o', 'touchdown', 'decisivo', 'e', 'derrubou', 'o', 'favorito', 'New', 'England', 'Patriots', 'por', 'a', 'n', 'este', 'domingo']
```

- Somente os números: **is\_digit**
- Somente pontuações: **is\_punct**

```
>>> tokens_numeros = [token .orth_ for token in doc if token.is_digit]
>>> tokens_numeros
['39', '17', '14']
>>> tokens_pontuacoes = [token .orth_ for token in doc if token.is_punct]
>>> tokens_pontuacoes
['.', ',', '.']
```

# SPACY - TOKENIZAÇÃO

- Retorno com tipos de tokens diferentes:
  - Pontuação esquerda ou direita
    - Parênteses e colchetes
  - Espaços
  - Símbolos financeiros
  - Números (10.9, 10, “dez”)
  - E-mail
  - Stopwords...
- Lista completa com os atributos [aqui](#).



# SPACY – STEMMING E LEMATIZAÇÃO

- Olha que interessante (e surpreendente): o spaCy não tem um stemmer padrão...
- Porém, o spaCy tem um lematizador!
  - O inverso do NLTK, pelo menos para o Português!
- Lematizar é simples: só utilizar o atributo **lemma\_**:

```
>>> import spacy
>>> nlp = spacy.load("pt_core_news_lg")
>>> texto = "Os Giants começaram com a posse de bola, e mostraram logo que iriam
alongar ao máximo suas posses de bola."
>>> doc = nlp(texto)
>>> lemmas = [token.lemma_ for token in doc if token.pos_ == 'VERB']
>>> lemmas
['começar', 'mostrar', 'alongar']
```

## SPACY – LEMATIZAÇÃO

- É importante observar que foi utilizado um outro atributo que ainda não falamos: o **pos\_**
- Esse atributo é referente ao *Part-Of-Speech*, ou simplesmente, a classe gramatical do token.
- Vale ressaltar que a lematização geralmente remete à forma canônica da palavra para os verbos, então é necessária essa condição.
- Ok, mas como obtida essa classe gramatical? É simples assim, só com um atributo?



# SPACY – ETIQUETADOR

- Sim. Basta chamar o atributo `pos_` no token e assim é retornada a classe gramatical referente!

```
>>> import spacy
>>> nlp = spacy.load("pt_core_news_lg")
>>> texto = "Os Giants começaram com a posse de bola, e mostraram logo que iriam
alongar ao máximo suas posses de bola."
>>> doc = nlp(texto)
>>> etiquetas = [(token.orth_, token.pos_) for token in doc]
>>> etiquetas
[('Os', 'DET'), ('Giants', 'PROPN'), ('começaram', 'VERB'), ('com', 'ADP'), ('a',
'DET'), ('posse', 'NOUN'), ('de', 'ADP'), ('bola', 'NOUN'), (',', 'PUNCT'), ('
e', 'CCONJ'), ('mostraram', 'VERB'), ('logo', 'ADV'), ('que', 'SCONJ'), ('iriam',
'AUX'), ('alongar', 'VERB'), ('ao', 'DET'), ('máximo', 'NOUN'), ('suas', 'DET'),
('posses', 'NOUN'), ('de', 'ADP'), ('bola', 'NOUN'), ('.', 'PUNCT')]
```

- O conjunto de etiquetas para o português está [aqui!](#)

# SPACY – ETIQUETADOR

- O spaCy tem um atributo que contém informações mais detalhadas: o **morph**

```
>>> import spacy
>>> nlp = spacy.load("pt_core_news_lg")
>>> import spacy
>>> nlp = spacy.load("pt_core_news_lg")
>>> texto = "Os Giants começaram com a posse de bola, e mostraram logo que iriam
alongar ao máximo suas posses de bola."
>>> doc = nlp(texto)
>>> etiquetas = [(token.orth_, token.morph) for token in doc]
>>> etiquetas
[('Os', Definite=Def|Gender=Masc|Number=Plur|PronType=Art), ('Giants', Gender=Masc|Number=Plur), ('começaram', Mood=Ind|Number=Plur|Person=3|Tense=Past|VerbForm=Fin), ('com', ), ('a', Definite=Def|Gender=Fem|Number=Sing|PronType=Art), ('posse', Gender=Fem|Number=Sing), ('de', ), ('bola', Gender=Fem|Number=Sing), (',', ), ('e', ), ('mostraram', Mood=Ind|Number=Plur|Person=3|Tense=Past|VerbForm=Fin), ('logo', ), ('que', ), ('iriam', Mood=Cnd|Number=Plur|Person=3|VerbForm=Fin), ('alongar', VerbForm=Inf), ('ao', Definite=Def|Gender=Masc|Number=Sing|PronType=Art), ('máximo', Gender=Masc|Number=Sing), ('suas', Gender=Fem|Number=Plur|PronType=Prs), ('posses', Gender=Fem|Number=Plur), ('de', ), ('bola', Gender=Fem|Number=Sing), ('.', )]
```

- Características mais morfológicas dos tokens!

# SPACY – ETIQUETADOR

- O modelo de linguagem para o Português usado no Spacy tem como fonte o Bosque
  - Acurácia de **97%** na etiquetagem quando utilizado o modelo de linguagem **large (lg)**
- Existem outros etiquetadores para o Português que alcançam uma acurácia maior
  - NLPNet – 97,33% (free)
  - PALAVRAS – 98% (pago)
    - É importante frisar que estes etiquetadores tem um foco total no Português.

# SPACY – ENTIDADES NOMEADAS

- Vamos colocar a mão na massa agora em coisas mais robustas que o spaCy nos proporciona
- Vimos no NLTK uma dificuldade inicial de identificar as entidades nomeadas de uma sentença
- Será que o spaCy facilita esse trabalho?

# SPACY – ENTIDADES NOMEADAS

- SIM!! Basta usar a propriedade **ents** na variável **doc**!

```
>>> import spacy
>>> nlp = spacy.load("pt_core_news_lg")
>>> texto = "Com um passê de Eli Manning para Plaxico Burress a 39 segundos do fim, o New York Giants anotou o touchdown decisivo e derrubou o favorito New England Patriots por 17 a 14 neste domingo, em Glendale, no Super Bowl XLII. O resultado, uma das maiores zebras da história do Super Bowl, acabou com a temporada perfeita de Tom Brady e companhia, que esperavam fazer história ao levantar o troféu da NFL sem sofrer uma derrota no ano."
>>> doc = nlp(texto)
>>> entidades = list(doc.ents)
>>> entidades
[Eli Manning, Plaxico, Burress, New York Giants, New England Patriots, Glendale, Super Bowl XLII, Super Bowl, Tom Brady, NFL]
```

- Olha só! Nossa lista contém praticamente todas as entidades nomeadas da sentença!



# SPACY – ENTIDADES NOMEADAS

- Detalhadamente, veja como o spaCy classifica cada entidade:

```
>>> entidades_detalhes = [(entidade, entidade.label_) for entidade in doc.ents]
>>> entidades_detalhes
[(Eli Manning, 'PER'), (Plaxico, 'PER'), (Burress, 'PER'), (New York Giants, 'ORG'), (New England Patriots, 'ORG'), (Glendale, 'LOC'), (Super Bowl XLII, 'ORG'), (Super Bowl, 'ORG'), (Tom Brady, 'PER'), (NFL, 'ORG')]
```

- A acurácia alta permite que as entidades sejam classificadas corretamente.
  - Por mais que tenhamos no nosso resultado uma entidade “separada”
    - A medida de acerto do modelo de linguagem treinado é de **91.24%** (F-Score)

# SPACY – ENTIDADES NOMEADAS

- Um exemplo com menos entidades em inglês:

```
>>> texto = "'A gente sabe que, quando uma pessoa está mentindo, inconscientem  
ente, isso afeta a produção do texto. Mudam as palavras que ela usa e as estrut  
uras do texto. Além disso, a pessoa costuma ser mais assertiva e emotiva. Então  
, uma das formas de detectar textos enganosos é medir essas características',  
explica o professor Thiago Pardo, do Instituto de Ciências Matemáticas e de Com  
putação (ICMC) da USP, em São Carlos. Pesquisador do Núcleo Interinstitucional  
de Linguística Computacional (NILC), Thiago é o coordenador do projeto que resu  
ltou na criação da plataforma e na publicação do artigo Contributions to the St  
udy of Fake News in Portuguese: New Corpus and Automatic Detection Results, apr  
esentado no final de setembro na 13ª Conferência Internacional de Processamento  
Computacional do Português."
```

```
>>> doc = nlp(texto)
```

```
>>> entidades = list(doc.ents)
```

```
>>> entidades
```

```
[Thiago Pardo, Instituto de Ciências Matemáticas e de Computação, ICMC, USP, Sã  
o Carlos, Núcleo Interinstitucional de Linguística Computacional, NILC, Thiago,  
Contributions to the Study of Fake News in Portuguese: New Corpus and Automatic  
Detection Results, Conferência Internacional de Processamento Computacional do  
Português]
```

```
>>> entidades_detalhes = [(entidade, entidade.label_) for entidade in doc.ents]
```

```
>>> entidades_detalhes
```

```
[(Thiago Pardo, 'PER'), (Instituto de Ciências Matemáticas e de Computação, 'OR  
G'), (ICMC, 'ORG'), (USP, 'LOC'), (São Carlos, 'LOC'), (Núcleo Interinstitucion  
al de Linguística Computacional, 'ORG'), (NILC, 'MISC'), (Thiago, 'ORG'), (Cont  
ributions to the Study of Fake News in Portuguese: New Corpus and Automatic Det  
ection Results, 'MISC'), (Conferência Internacional de Processamento Computacio  
nal do Português, 'ORG')]
```

# SPACY – ENTIDADES NOMEADAS

- É possível visualizar essas entidades nomeadas de forma gráfica, por meio do **displaCy**.
  - Usando o nosso primeiro exemplo, com um trecho do **cópus**, é possível destacar todas as entidades nomeadas:

```
15 import spacy
16 from pathlib import Path
17
18 nlp = spacy.load("pt_core_news_lg")
19 texto = "Com um passe de Eli Manning para Plaxico Burress a 39
20 doc = nlp(texto)
21
22 html = spacy.displacy.render(doc, style="ent")
23 output_path = Path("entidades_nomeadas.html")
24 output_path.open("w", encoding="utf-8").write(html)
```



# SPACY – ENTIDADES NOMEADAS

## ○ Resultado:

Com um passe de **Eli Manning PER** para **Plaxico PER** **Burress PER** a 39 segundos do fim, o **New York Giants ORG** anotou o touchdown decisivo e derrubou o favorito **New England Patriots ORG** por 17 a 14 neste domingo, em **Glendale LOC**, no **Super Bowl XLII ORG**. O resultado, uma das maiores zebras da história do **Super Bowl ORG**, acabou com a temporada perfeita de **Tom Brady PER** e companhia, que esperavam fazer história ao levantar o troféu da **NFL ORG** sem sofrer uma derrota no ano.

- Obs: o layout pode ser modificado da forma que você prefira. Veja aqui [nesse link](#) como fazer!

# SPACY – ANÁLISE SINTÁTICA

- Uma outra função importante do spaCy é a representação sintática do texto
  - Quais as relações entre os tokens
- O atributo **dep\_** retorna a dependência sintática do token em questão

```
>>> import spacy
>>> nlp = spacy.load("pt_core_news_lg")
>>> texto = "Os Giants começaram com a posse de bola, e mostraram logo que iriam alongar ao máximo suas posses de bola."
>>> doc = nlp(texto)
>>> sintatica = [(token.orth_, token.dep_) for token in doc]
>>> sintatica
[('Os', 'det'), ('Giants', 'nsubj'), ('começaram', 'ROOT'), ('com', 'case'), ('a', 'det'), ('posse', 'obl'), ('de', 'case'), ('bola', 'nmod'), (',', 'punct'), ('e', 'cc'), ('mostraram', 'conj'), ('logo', 'advmod'), ('que', 'mark'), ('iriam', 'aux'), ('alongar', 'ccomp'), ('ao', 'case'), ('máximo', 'obl'), ('suas', 'det'), ('posses', 'obj'), ('de', 'case'), ('bola', 'nmod'), ('.', 'punct')]
```

- O conjunto de etiquetas tá [nesse link](#).

# SPACY – ANÁLISE SINTÁTICA

- O spaCy também permite a visualização das dependências de forma gráfica pelo **displaCy**:

```
15 import spacy
16 from pathlib import Path
17
18 nlp = spacy.load("pt_core_news_lg")
19 texto = "Os Giants começaram com a posse de bola, e m
20 doc = nlp(texto)
21
22 svg = spacy.displacy.render(doc, style="dep")
23 output_path = Path("analise_dependencia.svg")
24 output_path.open("w", encoding="utf-8").write(svg)
```

- [Aqui](#) o resultado!

# SPACY – DISPLACY

- O spaCy contém dois sites onde podem ser feitas as análises de entidades nomeadas e de dependências de forma bem simples:
- Visualizador de Entidades Nomeadas
  - <https://explosion.ai/demos/displacy-ent>
- Visualizador de Dependências
  - <https://explosion.ai/demos/displacy>
- Basta selecionar o modelo para português (ou qualquer outra linguagem) e brincar!

# SPACY – SIMILARIDADE ENTRE PALAVRAS

- Por ter um bom e grande modelo de linguagem para o Português, o spaCy permite avaliar similaridade entre palavras!
- E continua sendo simples: só usar o método **similarity()**!

```
>>> import spacy
>>> nlp = spacy.load("pt_core_news_lg")
>>> palavras = "conversar falar correr"
>>> doc = nlp(palavras)
>>> tokens = [token for token in doc]
>>> tokens[0].similarity(tokens[1])
0.73501545
>>> tokens[0].similarity(tokens[2])
0.44497716
>>> tokens[1].similarity(tokens[2])
0.4326754
```

# SPACY – SIMILARIDADE ENTRE PALAVRAS

- Então, podemos fazer várias análises de similaridade entre palavras no texto!
- Exemplo 1: homem e mulher

```
>>> tokens[0].similarity(tokens[1])  
0.6595782
```

- Exemplo 2: Roma e Itália

```
>>> tokens[0].similarity(tokens[1])  
0.6953801
```

- Exemplo 3: eu e livro

```
>>> tokens[0].similarity(tokens[1])  
0.19232121
```

# SPACY – SIMILARIDADE ENTRE PALAVRAS

- O cálculo da similaridade é feito por meio da medida do cosseno

$$scos(\vec{f}, \vec{v}) = \frac{\vec{f} \cdot \vec{v}}{|\vec{f}| |\vec{v}|} = \frac{\sum_{i=1}^n f_i v_i}{\sqrt{\sum_{i=1}^n f_i^2} \sqrt{\sum_{i=1}^n v_i^2}}$$

- Intervalo [0-1], onde 0 representa vetores completamente diferentes e 1 representa vetores completamente similares.



# SPACY – SIMILARIDADE: WORD2VEC

- O Word2Vec é uma técnica cuja a ideia é transformar cada token do texto em um vetor numérico para representação semântica.
- É uma das técnicas mais utilizadas no pré-processamento de textos e aprendizado de **word embeddings**.
- É possível a utilização dessa técnica dentro do spaCy
  - É parecido com o atributo **similarity()**, porém, como geralmente usam-se modelos maiores e treinados com mais dados, **pode ser** mais eficiente o uso do word2vec.



# SPACY – SIMILARIDADE: WORD2VEC

- Precisamos seguir 3 passos para usar os princípios do word2vec no spaCy:
  - 1. Encontrar modelos de embeddings treinados
  - 2. Converter o modelo para o spaCy
  - 3. Adequar o código da aplicação no spaCy para utilização do word2vec

# SPACY – SIMILARIDADE: WORD2VEC

- Precisamos seguir 3 passos para usar os princípios do word2vec no spaCy:
- 1. Encontrar modelos de embeddings treinados
  - Existem vários modelos de word embeddings treinados, um para cada fim. Utilizaremos as word embeddings do NILC, que estão [aqui](#).
  - Dois modelos são disponibilizados: CBOW e SKIP-GRAM
    - CBOW: modelo utilizado para **descobrir a palavra central** de uma sentença, baseado nas palavras que o cercam.
    - SKIP-GRAM: modelo utilizado para **descobrir as palavras de contexto** a partir de uma palavra central.

# SPaCY – SIMILARIDADE: WORD2VEC

- Precisamos seguir 3 passos para usar os princípios do word2vec no spaCy:
- 2. Converter o modelo para o spaCy

```
python -m spacy init vectors pt <local_emb> <nome_pasta>
```

- <nome\_da\_pasta> é a identificação de onde será armazenado o modelo convertido
- <local\_emb> é o caminho que se encontra o modelo baixado anteriormente

# SPACY – SIMILARIDADE: WORD2VEC

- Precisamos seguir 3 passos para usar os princípios do word2vec no spaCy:
- 2. Converter o modelo para o spaCy

```
python -m spacy init vectors pt <local_emb> <nome_pasta>
```

```
C:\Users\roney\Desktop>python -m spacy init vectors pt cbow_s50.txt vectors_spacy
[2021-03-23 15:25:10,224] [INFO] Creating blank nlp object for language 'pt'
[2021-03-23 15:25:10,224] [INFO] Reading vectors from cbow_s50.txt
929606it [00:24, 38397.92it/s]
[2021-03-23 15:25:34,474] [INFO] Loaded vectors from cbow_s50.txt
[2021-03-23 15:25:34,474] [INFO] Successfully converted 929606 vectors
[2021-03-23 15:25:34,474] [INFO] Saved nlp object with vectors to output directory. You can now use the path to
it in your config as the 'vectors' setting in [initialize].
C:\Users\roney\Desktop\vectors_spacy
```

- Ao final, é criada uma pasta com vários itens que são usados pelo spaCy na manipulação dos vetores.

# SPACY – SIMILARIDADE: WORD2VEC

- Precisamos seguir 3 passos para usar os princípios do word2vec no spaCy:
- 3. Adequar o código no spaCy para utilização do word2vec

```
35 import spacy
36 from spacy import util as spc_util
37
38 palavras = "conversar falar"
39 nlp = spacy.load("pt_core_news_lg")
40 doc = nlp(palavras)
41 tokens = [token for token in doc]
42
43 print("Similaridade - spaCy:", tokens[0].similarity(tokens[1]))
44
45 pathw2v = 'vectors_spacy'
46 spc_util.load_model(pathw2v, vocab=nlp.vocab)
47
48 print("Similaridade - word2vec:", tokens[0].similarity(tokens[1]))
```

---

```
Similaridade - spaCy: 0.73501545
Similaridade - word2vec: 0.7504553
```

# SPaCY – SIMILARIDADE: WORD2VEC

- Precisamos seguir 3 passos para usar os princípios do word2vec no spaCy:
- 3. Adequar o código no spaCy para utilização do word2vec
  - Perceba que a similaridade aumentou com o modelo word2vec treinado em comparação com o modelo de linguagem do spaCy
    - E se testarmos a similaridade entre justiça e trabalho?

# SPaCY – SIMILARIDADE: WORD2VEC

- Precisamos seguir 3 passos para usar os princípios do word2vec no spaCy:
- 3. Adequar o código no spaCy para utilização do word2vec
  - Perceba que a similaridade aumentou com o modelo word2vec treinado em comparação com o modelo de linguagem do spaCy
    - E se testarmos a similaridade entre justiça e trabalho?

```
Similaridade - spaCy: 0.31346163  
Similaridade - word2vec: 0.2301711
```

- O modelo do spaCy foi melhor. Veja que vai depender muito do modelo word2vec treinado e de quantas dimensões os vetores estão dispostos.
  - Como resolve? Testes, testes e mais testes...!

# SPACY – SIMILARIDADE: WORD2VEC

- Vamos fazer aquele teste clássico:

**MADRI – ESPANHA + FRANÇA ≈ PARIS**

- Precisamos fazer **operações entre vetores**.
- O spaCy tem um atributo que retorna o vetor do token em questão: **vector**
  - Para as operações com vetores utilizaremos o módulo **Numpy**
    - Instalação: `pip install numpy`
  - Para o cálculo da similaridade, utilizaremos o método pronto proveniente do módulo **Scikit-learn**
    - Instalação: `pip install -U scikit-learn`



# SPACY – SIMILARIDADE: WORD2VEC

- Vamos fazer aquele teste clássico:

**MADRI – ESPANHA + FRANÇA ≈ PARIS**

```
32 import spacy
33 from spacy import util as spc_util
34 import numpy as np
35 from sklearn.metrics.pairwise import cosine_similarity
36
37 palavras = "madri espanha França paris"
38 nlp = spacy.load("pt_core_news_lg")
39 doc = nlp(palavras)
40 tokens = [token for token in doc]
41
42 pathw2v = 'vectors_spacy'
43 spc_util.load_model(pathw2v, vocab=nlp.vocab)
44
45 # Madri - Espanha + França
46 vetor_res = np.array(tokens[0].vector) - np.array(tokens[1].vector) + np.array(tokens[2].vector)
47
48 # É necessário remodelar o vetor retornado pelo spaCy,
49 # pois ele está em 1 dimensão e para o uso do cosseno, é necessário um vetor de 2 dimensões
50 vetor_res = vetor_res.reshape(1,-1)
51 vetor_paris = tokens[3].vector.reshape(1,-1)
52
53 similaridade = cosine_similarity(vetor_res,vetor_paris)
54 print(similaridade)
```

```
[[0.8048478]]
```

# SPACY – EXERCÍCIOS DE FIXAÇÃO

- 1. Dada uma palavra, encontrar no cópús as 3 outras palavras que mais são próximas semanticamente e as 3 palavras que são mais distantes.
  - Faça o teste com o modelo do próprio spaCy e algum modelo word2vec do NILC Embeddings

# SPACY – EXERCÍCIOS DE FIXAÇÃO

- 2. Encontrar os vetores de todas as palavras do `córpus` e descobrir quais são as palavras relacionadas.
  - Dica: testar a similaridade entre todas as palavras do `córpus` e ordenar o par (`palavra1`, `palavra2`) pelo seu valor de similaridade
    - Utilização de estrutura de repetição (**for** ou **while**)
    - Utilização de um algoritmo de ordenação ou estrutura de controle (**if**)
  - Analisar quais valores foram retornados e confirmar a relação entre as palavras.

# BERT – BUSCANDO O CONTEXTO

- BERT - *Bidirectional Encoder Representations from Transformers*
  - Nível linguístico mais **semântico**
- Basicamente, o BERT analisa o **contexto à esquerda e à direita** do token
  - Compreensão muito mais profunda sobre as relações entre palavras e entre frases.
- O BERT constrói um **modelo de linguagem** e após o treinamento do modelo, passa pelo “**ajuste fino**”
  - O que o submete a **tarefas específicas** com entradas e saídas conforme preferido.

# BERT – BUSCANDO O CONTEXTO

- O que vamos precisar para fazer o BERT rodar:
  - Os transformadores
  - Os modelos BERT treinados
  - Um módulo que permita execução dos modelos
  - O ambiente de execução do processo

# BERT – BUSCANDO O CONTEXTO

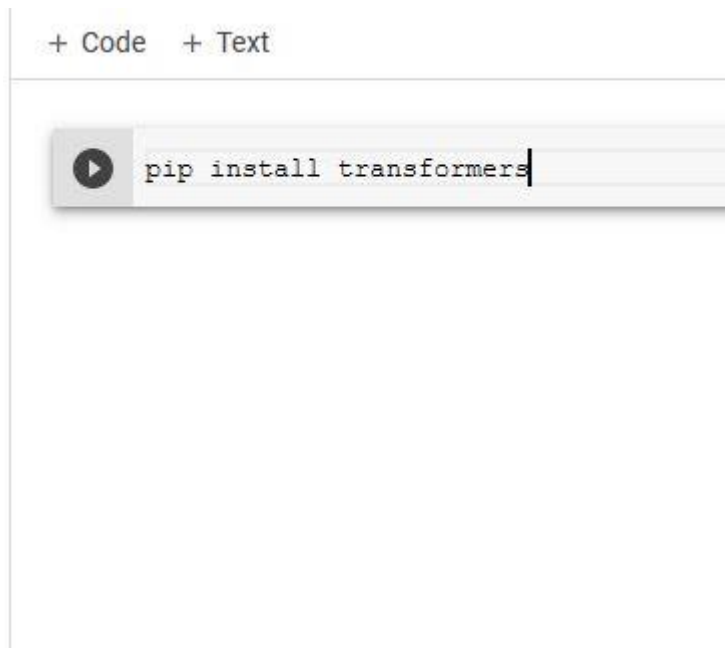
- O que vamos precisar para fazer o BERT rodar:
  - Os transformadores
    - `pip install transformers`
  - Os modelos BERT treinados
    - [BERTimbau – Portuguese BERT](#)
  - Um módulo que permita execução dos modelos
    - PyTorch
    - TensorFlow
  - O ambiente de execução do processo
    - Google Colab
    - GPU

# BERT – BUSCANDO O CONTEXTO

- Para nossas práticas, utilizaremos o Google Colab
  - <https://colab.research.google.com/notebooks/intro.ipynb>
- Uma vez que o BERT é do Google, você pode utilizar parte dos servidores poderosos deles para fazer alguns testes!
  - Até porque a execução do BERT é super pesada, nossos computadores podem não serem suficientes...
- Duas tarefas práticas:
  - 1. Predizer **qual palavra completa** uma dada parte de uma sentença.
  - 2. Verificar a **similaridade entre duas palavras** dentro do **contexto** da sentença.

# BERT – BUSCANDO O CONTEXTO

- Fazer o download/instalação dos transformadores



The image shows a screenshot of a Google Colab interface. At the top, there are two tabs: '+ Code' and '+ Text'. Below the tabs is a code cell with a play button icon on the left and the text 'pip install transformers' followed by a cursor. The code cell is highlighted with a light gray background.

- O uso do Google Colab é idêntico ao prompt de comando ou terminal que utilizamos no Windows/Linux/MacOS!





# BERT – BUSCANDO O CONTEXTO: TAREFA 1

- Predizer a palavra que completa a sentença por meio do BERT
  - O código pode ser executado [aqui](#):
- Método `calc_score_tarefa1()`
  - Recebe como parâmetros:
    - `text` – a sentença completa
    - `target_word` – o token esperado que completa corretamente a sentença
    - `tokenizer` – modelo treinado com o vocabulário da língua
    - `model` – modelo pré-treinado (BERTimbau)
    - `debug=True` – particularidade do código para mostrar as informações mais detalhadas

# BERT – BUSCANDO O CONTEXTO: TAREFA 1

- Predizer a palavra que completa a sentença por meio do BERT
  - O código pode ser executado [aqui](#):
- A execução:
  - Primeiro fazemos a importação do módulo para a execução do modelo: **PyTorch**
  - Logo após importamos os módulos que irão manipular os modelos pré-treinados do **BERTimbau**
  - Na variável **text**, três detalhes:
    - **[CLS]** indica o início da sentença
    - **[SEP]** indica o final da sentença
    - **[MASK]** indica a parte da sentença que queremos predizer

# BERT – BUSCANDO O CONTEXTO: TAREFA 1

- Predizer a palavra que completa a sentença por meio do BERT
  - O código pode ser executado [aqui](#):
- A execução:
  - Por fim, retorna-se o token predito (*predicted token*), o esperado (*expected token*) e uma medida de confiança do token predito com o esperado no contexto.
    - No nosso exemplo, esperávamos ‘pressões’, mas o BERT retornou ‘cargas’

```
predicted token ---> cargas 0.9999845
expected token ---> pressões 0.999969
Score: 0.9999845027923584
```

- Quanto mais próximo de 1, mais confiável é o token predito no contexto.

# BERT – BUSCANDO O CONTEXTO: TAREFA 2

- Verificar a similaridade de contexto entre duas palavras dentro de uma sentença
  - O código pode ser executado [aqui](#):
- Método `calc_score_tarefa2()`
  - Recebe como parâmetros:
    - `text` – a sentença completa
    - `target_word` e `predicted_word` – tokens para análise de similaridade de contexto na sentença
    - `tokenizer` – modelo treinado com o vocabulário da língua
    - `model` – modelo pré-treinado (BERTimbau)
    - `debug=True` – particularidade do código para mostrar as informações mais detalhadas

# BERT – BUSCANDO O CONTEXTO: TAREFA 2

- Verificar a similaridade de contexto entre duas palavras dentro de uma sentença
  - O código pode ser executado [aqui](#):

- A execução:

- Mesmos princípios da Tarefa 1, porém, aqui, existe um parâmetro adicional que é o **token predito**
- A inclusão do token predito serve para o retorno do valor da similaridade no contexto
  - Chegamos na metade da temporada! Você disputou [MASK] contra todas as equipes do grupo uma vez e a agora inicia-se o retorno

```
predicted_word = 'jogos'  
target_word = 'casas'
```

```
predicted token ---> jogos 0.9997973  
expected token ---> casas 0.49738413  
Score: 0.49758684635162354
```

- A predição de 'jogos' é melhor que a predição esperada de 'casas' no contexto da frase. O valor retornado é bem distante de 1.

# BERT – EXERCÍCIOS DE FIXAÇÃO

- **3.** Usando os modelos pré-treinados do BERT para português e os métodos apresentados em aula, retorne a predição de mais de uma palavra dentro da mesma sentença
  - Deve-se pensar em tratar a variável **target\_word** como uma lista e utilizar estruturas de repetição quando for tratar cada um dos tokens.
    - Dicas: observar como se recuperam os tokens (**get tokens**) e como retorna-se o token predito (variável **predicted\_token**)

**DESAFIADOR!!!**

# BERT – EXERCÍCIOS DE FIXAÇÃO

- 4. Dada uma sentença, selecionar um token a ser predito e comparar com outros 10 tokens, ordenando-os por seus valores de similaridade no contexto da sentença.

**DESAFIADOR!!!**