



PYTHON PARA PLN

spaCy + Embeddings

Rogério Figueredo de Sousa rogerfig@usp.br

Roney Lira de Sales Santos roneysantos@usp.br

Prof. Thiago A. S. Pardo

SPACY – RELEMBRANDO...

- Biblioteca Python para **uso em produção**
- **Modelos de linguagem** robustos para o português
- A maioria do processamento gira em torno dos objetos **Doc** e **Token**
- Tarefas de PLN facilmente realizadas por meio de **atributos**
 - `lemma_`, `pos_`, `morph`, `ents`, `label_`, `dep_`, ...
- **Visualização gráfica** de algumas tarefas de PLN pelo **displaCy**

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
```

INTRODUÇÃO À *WORD EMBEDDINGS*

○ Hipótese distribucional

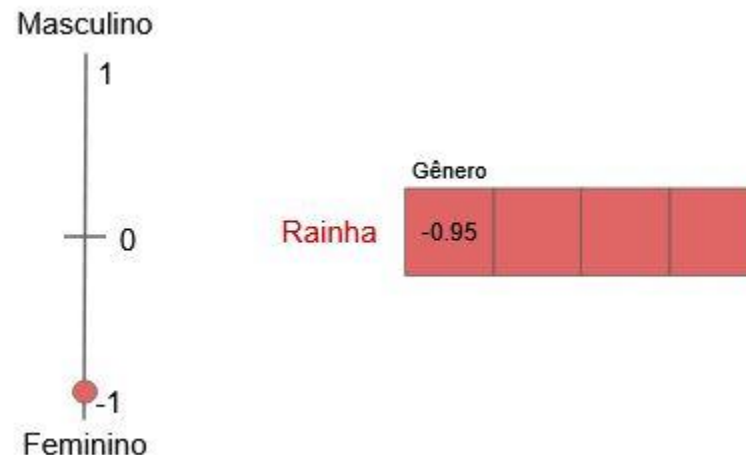
- Palavras tem **significados parecidos** quando são usadas em **contextos parecidos**.

○ Modelos de linguagem

- Predizem a próxima palavra, dado um conjunto de palavras
- Exemplo: “O gato corre atrás do _____”
 - Qual a próxima palavra? “rato”? “cachorro”? “carro”?
- Os modelos de linguagem são usados para tarefas como processamento de voz, autocorreção de ortografia, etc.

INTRODUÇÃO À *WORD EMBEDDINGS*

- **WORD EMBEDDING:** representação vetorial de uma palavra.
 - texto -> números
- Exemplo: Definição da palavra “rainha” com uma escala “Gênero”, que vai de -1 a 1: quanto mais perto de -1, mais feminina:



INTRODUÇÃO À *WORD EMBEDDINGS*

- Porém, só com a informação sobre gênero não é possível representar bem a palavra...
- Podem ser adicionadas **várias outras dimensões**, ou quadradinhos, com a **escala que a palavra tem mais a ver**.

- No exemplo anterior, imagine “rainha” e “rei” em escalas de “**Realeza**”, “**Fruta**” e “**Violência**”, por exemplo:

	Rainha	Rei
Gênero	-0.95	0.789
Realeza	0.89	0.96
...
Fruta	0.015	-0.05
Violência	0.56	0.8

INTRODUÇÃO À *WORD EMBEDDINGS*

- E como esses valores são atribuídos?
- A partir de **aprendizado de máquina!**
 - Usa-se algum algoritmo para gerar, a partir do seu contexto.
- E o **tamanho ideal** do vetor, ou seja, a quantidade de dimensões?
 - **Depende do seu corpus/dataset de treinamento:** quanto menor, menos dimensões
 - Geralmente é um valor entre **100 e 1000**.
- Um algoritmo muito utilizado para obter as *word embeddings* é chamado **Word2Vec**.

SIMILARIDADE DO COSSENO

- 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  $\approx$  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]]
```

# 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 BERT gera o modelo de linguagem:
  - Masked Language Model (MLM)
  - Next Sentence Prediction (NSP)

“BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks...”

- Essas duas tarefas são capazes de dar ao BERT um excelente entendimento da língua.

# BERT – BUSCANDO O CONTEXTO

- Termos frequentes
  - Pre-Training
  - Transfer Learning
  - Fine-Tuning
- Onde aplicar o BERT
  - Classificação de Tokens
    - Reconhecimento de Entidades Nomeadas
    - Respostas de Questões
  - Classificação de Textos
    - Análise de Sentimentos
  - Classificação de Pares de Textos
    - Similaridade Semântica e Inferência textual
    - Comparação de Perguntas (Duplicação)

# 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

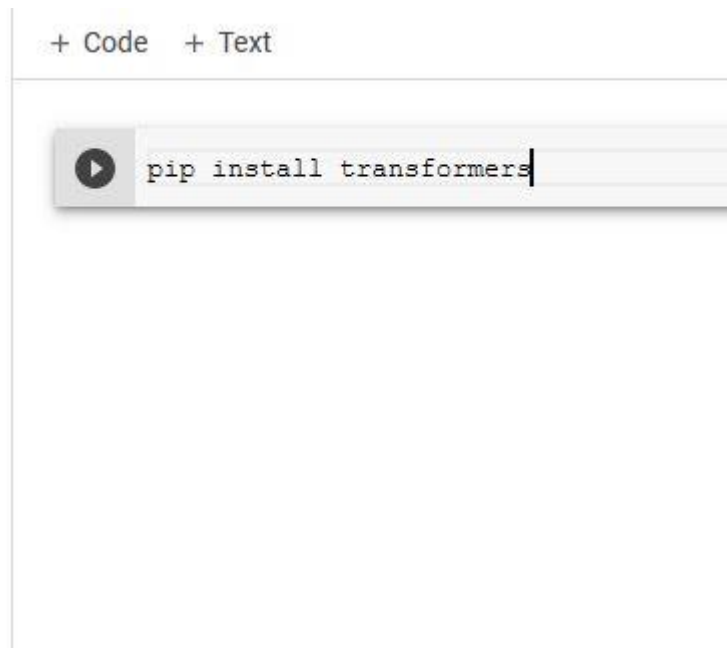


# BERT – BUSCANDO O CONTEXTO

- 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 snippet 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

- Fazer o download/instalação dos transformadores
  - O progresso da instalação será mostrado na tela

```
[1] pip install transformers
```

```
Collecting transformers
 Downloading https://files.pythonhosted.org/packages/2c/4e/4f1ede0fd7a36278844a277f8d53c21f88f37f/
 [REDACTED] | 1.3MB 2.8MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.6/dist-packages (from transformers)
Collecting sacremoses
 Downloading https://files.pythonhosted.org/packages/7d/34/09d19aff26edcc8eb2a01bed8e98f13a15370c/
 [REDACTED] | 890kB 16.6MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from transformers)
Requirement already satisfied: dataclasses; python_version < "3.7" in /usr/local/lib/python3.6/dist-packages (from transformers)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.6/dist-packages (from transformers)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from transformers)
Requirement already satisfied: protobuf in /usr/local/lib/python3.6/dist-packages (from transformers)
Collecting tokenizers==0.9.2
 Downloading https://files.pythonhosted.org/packages/7c/a5/78be1a55b2ac8d6a956f0a211d372726e2b1dc/
 [REDACTED] | 2.9MB 16.5MB/s
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.6/dist-packages (from transformers)
Collecting sentencepiece!=0.1.92
 Downloading https://files.pythonhosted.org/packages/e5/2d/6d4ca4bef9a67070fa1cac508606328329152k/
 [REDACTED] | 1.1MB 40.1MB/s
Requirement already satisfied: packaging in /usr/local/lib/python3.6/dist-packages (from transformers)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from sacremoses->transformers)
Requirement already satisfied: click in /usr/local/lib/python3.6/dist-packages (from sacremoses->transformers)
Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-packages (from sacremoses->transformers)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from protobuf)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.6/dist-packages (from packaging)
Building wheels for collected packages: sacremoses
 Building wheel for sacremoses (setup.py) ... done
 Created wheel for sacremoses: filename=sacremoses-0.0.43-cp36-none-any.whl size=893257 sha256=21
 Stored in directory: /root/.cache/pip/wheels/29/3c/fd/7ce5c3f0666dab31a50123635e6fb5e19ceb42ce3e
Successfully built sacremoses
Installing collected packages: sacremoses, tokenizers, sentencepiece, transformers
Successfully installed sacremoses-0.0.43 sentencepiece-0.1.94 tokenizers-0.9.2 transformers-3.4.0
```

# 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 prever

# 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.



# SPACY – EXERCÍCIOS DE FIXAÇÃO

- 1. Dada uma palavra, encontrar no *córpus* as 3 outras palavras que mais são próximas semanticamente e as 3 palavras que são mais distantes.
  - Dessa vez faça o teste com algum modelo word2vec do NILC Embeddings

# BERT – EXERCÍCIOS DE FIXAÇÃO

- **2.** 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

- **3.** 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!!!**