Semântica Distribucional

IX International Seminar on Statistics with R

Eduardo Corrêa Gonçalves (ENCE-IBGE)

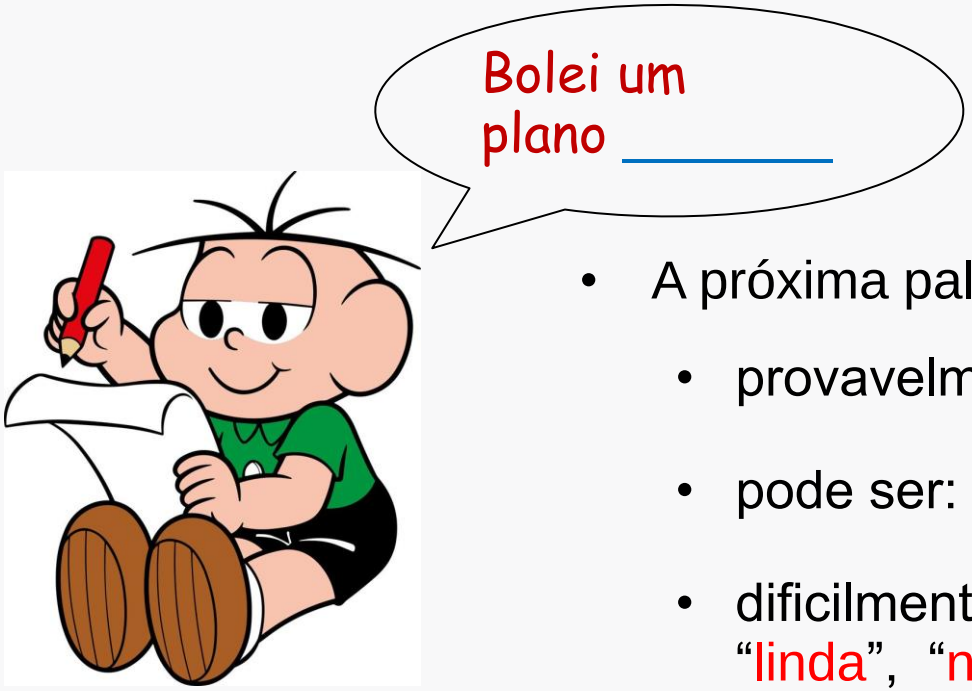
28/05/2025

Sumário

Introdução (1/6)

- **Prevendo o futuro**
  - Prever o futuro é muito difícil...

- Porém, prever as próximas palavras que alguém vai falar é um pouco menos difícil.



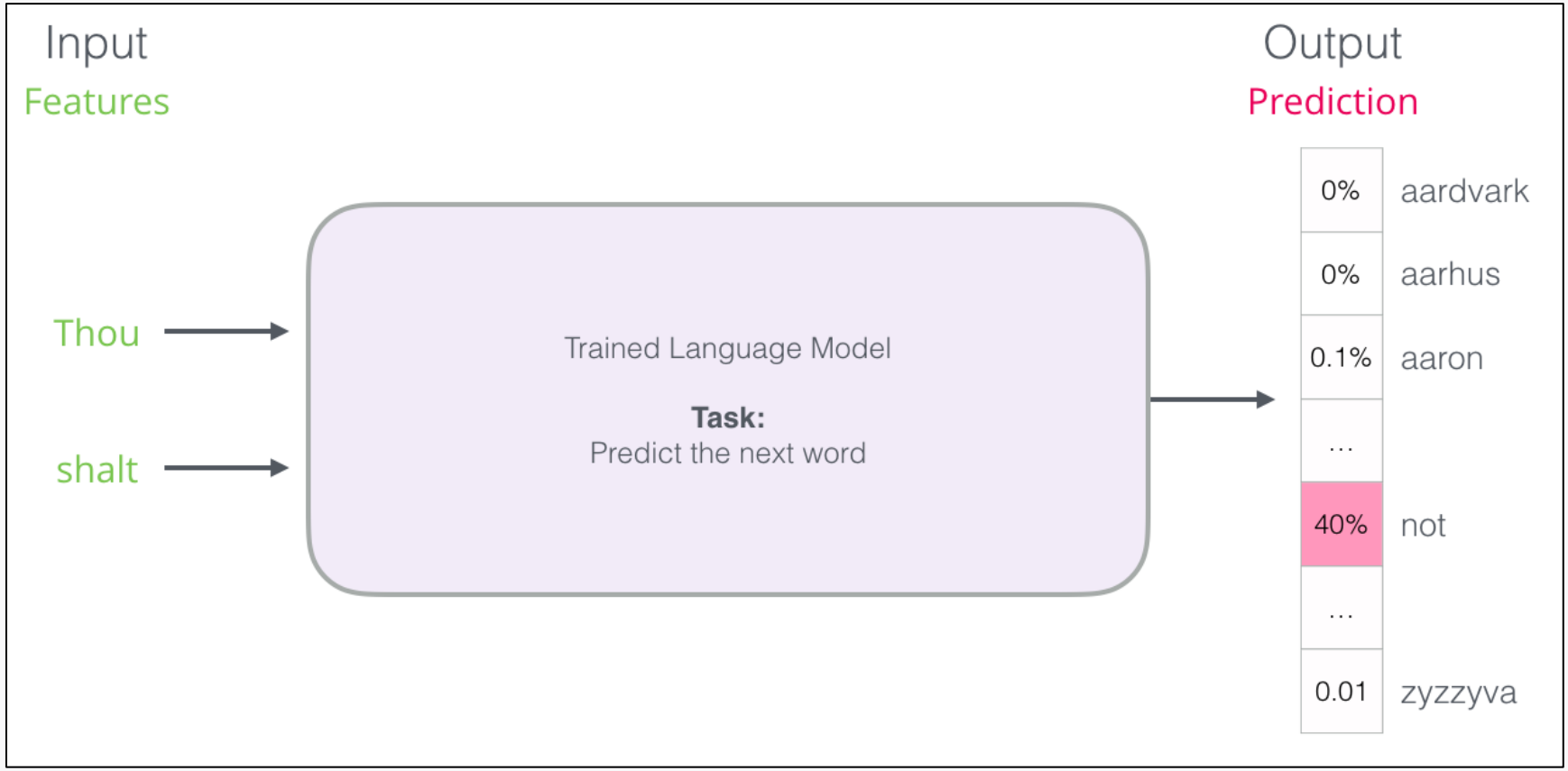
- A próxima palavra:
  - provavelmente é "infalível"
  - pode ser: "ótimo", "novo", ...
  - dificilmente "Icaraí", "subia", "soja", "CO_2", "linda", "nadar", "estocástico", "mamãe", "marmoraria", "Paysandu", "atum", ...

Introdução (2/6)

- **Prevendo a próxima palavra**
  - No Processamento de Linguagem Natural (PLN), a tarefa de prever a próxima palavra (***next-word prediction***) é realizada por um **modelo de linguagem**.
    - Modelos de linguagem trabalham associando uma probabilidade para cada próxima palavra possível.
    - Servem também para associar uma probabilidade para uma sentença inteira.
      - Esses modelos "sabem" que: "Bolei um plano infalível"
      - É muito mais provável que:
        - "Bolei um plano atum"
        - "Bolei um plano Icaraí"
        - "Infalível plano um bolei"
        - ...

Introdução (3/6)

- **Modelo de Linguagem** – é um modelo treinado sobre uma grande quantidade de textos.
  - Ele recebe como entrada uma lista de palavras (ex.: 2 palavras) e associa uma probabilidade a cada próxima palavra possível



Fonte: Alammar (2019)

Introdução (4/6)

- **Mas os modelos de linguagem sabem ler ?**
- *humanos assistindo essa apresentação, por favor leiam a frase abaixo:*

> *"Ser feliz sem motivo é a forma mais autêntica de felicidade."*

- É **relativamente fácil** para **nós**, **seres humanos**:
  - Extrair informações da frase a partir de uma simples leitura.
  - Reconhecer o sentido das palavras e o significado do texto formado pela junção de todas as palavras.

* Exemplo retirado de Seno et al. (2024).

Introdução (5/6)

- **Mas os modelos de linguagem sabem ler ?**

> *"Ser feliz sem motivo é a forma mais autêntica de felicidade."*

- Entretanto para **eles**, os **modelos de linguagem**, compostos por **algoritmos** e **métodos estatísticos**:
  - Não há como entender palavras.
  - Ao invés, eles precisam de uma representação numérica do texto a ser processado.
    - Apenas assim, conseguirão realizar suas operações.

Introdução (6/6)

- **É aí que entra a Semântica Distribucional (SD)**
  - Principal abordagem para a representação do **significado** lexical na PLN.

- **Princípio básico – "palavras como vetores":**
  - Palavras são representadas por meio de **vetores** de **valores reais**.
    - São denominados **vetores semânticos**.
    - Eles **codificam o significado** das palavras a partir de sua distribuição em textos.

Semântica Distribucional (1/7)

- **A SD é ancorada na Hipótese Distribucional ,**
  - "palavras que têm um **contexto linguístico semelhante**, tendem a ter **significado similar** ou aproximado"
  - Teoria formulada na década de 1950 por linguistas como J. R. Firth e Zellig S. Harris.
  - Eles perceberam que:
    - Palavras com **significado relacionado** (ex.: "quente" e "abafado") ...
    - ... tendem a ocorrer em um **mesmo contexto** (ex.: perto de certas palavras como "clima", "dia", "local")

Semântica Distribucional (2/7)

- **Exemplo** (Parrish, 2018):
  - “Estava muito frio ontem.”
  - “Ficará muito quente amanhã”
  - “Vai fazer muito calor na terça?”
  - De acordo com a Hipótese Distribucional:
    - As palavras “frio”, “quente” e “calor” devem possuir algum tipo de relação.
    - Uma vez que ocorrem em um **contexto similar**.
      - Nesse caso, entre a palavra “muito” e uma palavra que indica um dia em particular.

Semântica Distribucional (3/7)

- **Exemplo**:
  - “Estava muito frio ontem.”
  - “Ficará muito quente amanhã”
  - “Vai fazer muito calor na terça?”
  - De maneira análoga:
    - “ontem”, “amanhã” e “terça” devem estar relacionadas.
    - Já que ocorrem em um contexto de uma palavra que indica temperatura !

Semântica Distribucional (4/7)

> *"You shall know a word by the company it keeps"*
>
> J. R. Firth, 1957

- Para a Hipótese Distribucional, o significado de uma palavra é baseado em uma grande lista de todos os contextos em que ela ocorre.

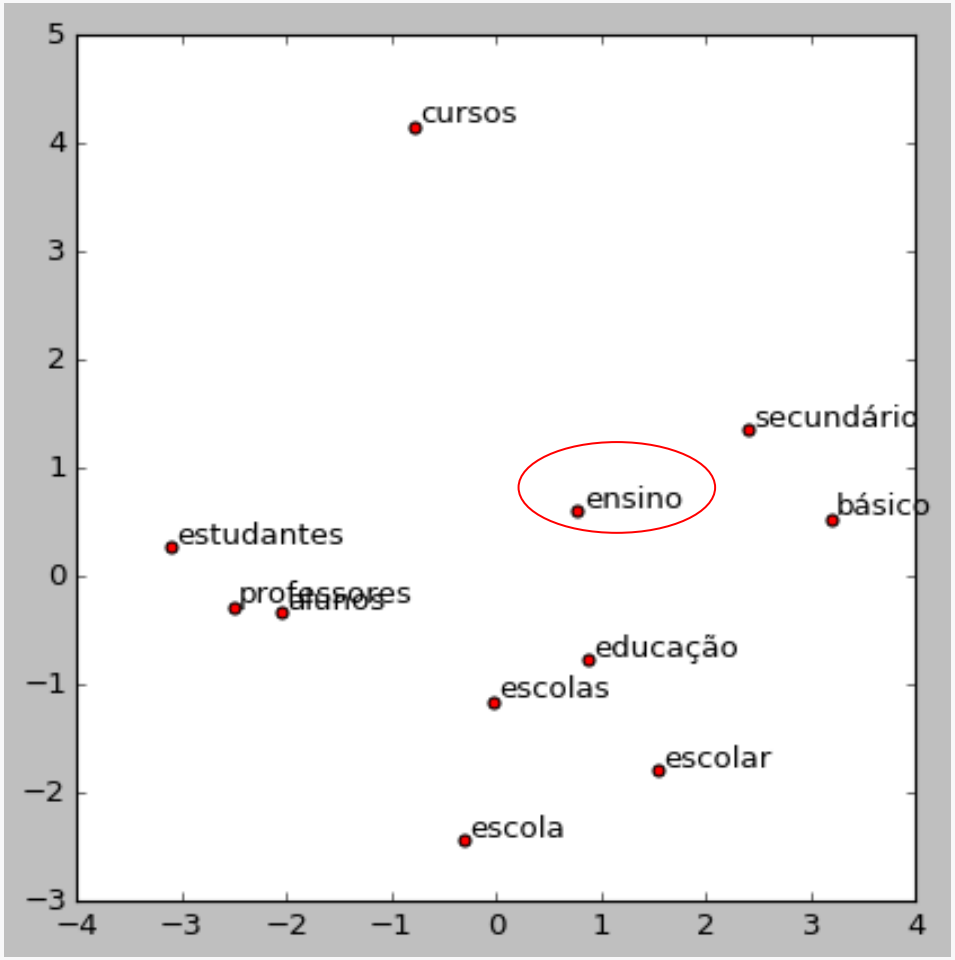
- Palavras que compartilham contextos possuem alguma relação semântica.

Semântica Distribucional (5/7)

- **Modelos Semânticos Distribucionais (MSD)**
  - São modelos que empregam a Hipótese Distribucional para fazer o computador “entender” a semântica das palavras.
  - Têm por objetivo produzir vetores de propósito geral que capturam o significado das palavras.
    - São denominados **vetores semânticos**
  - Os MSD produzem uma representação vetorial que retrata o significado de uma palavra a partir da distribuição das palavras que formam o seu contexto.

Semântica Distribucional (6/7)

- **Modelos Semânticos Distribucionais (MSD)**
  - **Exemplo** (Seno et al. 2024): espaço vetorial semântico de “ensino”.
  - Note que palavras como “educação” e “escolar” compartilham esse mesmo espaço semântico.
  - Cada ponto do espaço multidimensional representa a conotação de uma palavra.



Semântica Distribucional (7/7)

- **Vetores Semânticos**
  - A representação vetorial semântica é o padrão de representação mais comum em PLN.
  - Ela pode retratar **vários aspectos** do **significado** das palavras.
  - **Exemplo**:

    - similaridade → “comércio” × “negócios”
    - sentimento → “fenomenal” × “estúpido”
    - associação → “futebol” × “bola”

Fonte: Seno et al. (2024)

Matriz Termo-Contexto (1/5)

- **Vetores semânticos podem ser criados com a matriz termo-contexto**
  - Nesta matriz, o espaço vetorial é formado por uma coleção de **palavras**, que **representam os vetores** nesse espaço.
    - Cada **célula** representa o número de vezes que a **palavra da linha** (alvo) e a **palavra da coluna** (contexto) **coocorrem em algum contexto** no corpus.
    - O objetivo é que os vetores representem o significado das palavras.
    - Também é chamada de matriz palavra-palavra ou termo-termo

	aadvark	aargh	...	computer	data	pie
cherry	0	0		2	8	442
strawberry	0	0		0	0	60
digital	0	0		1670	1683	5
information	0	0		3325	3982	5

Matriz Termo-Contexto (2/5)

- **Mas o que é contexto ?**
  - Normalmente, um **número de palavras à esquerda e à direita** da **palavra-alvo** (linha).
    - Por exemplo, 3 palavras à esquerda e 3 palavras à direita.
  - Dessa forma, cada célula representa o número de ocorrência da palavra da coluna (contexto) em uma janela de três palavras em torno da palavra-alvo.
  - **Ex.**: matriz termo-contexto do corpus da Wikipedia*:

	aadvark	**aargh**	**...**	**computer**	**data**	**pie**
cherry	0	0		2	8	442
strawberry	0	0		0	0	60
digital	0	0		1670	1683	5
information	0	0		3325	3982	5

* Exemplo retirado de Jurafsky e Martin (2025)

Matriz Termo-Contexto (3/5)

- **Exemplo de Matriz Termo-Contexto – Corpus da Wikipedia**
  - Linha = representa cada palavra-alvo
  - Coluna = representa o contexto
  - Célula = indica o número de vezes que a palavra-alvo (linha) coocorreu em um dado contexto (coluna)
  - Por exemplo, um **contexto de tamanho 5** → significa 5 palavras à esquerda e 5 à direita da palavra-alvo.

	aadvark	**aargh**	**...**	**computer**	**data**	**pie**
cherry	0	0		2	8	442
strawberry	0	0		0	0	60
digital	0	0		1670	1683	5
information	0	0		3325	3982	5

Matriz Termo-Contexto (4/5)

- **Exemplo de Matriz Termo-Contexto – Corpus da Wikipedia**
  - Cada **palavra-alvo** (linha da matriz) representa um **vetor**.
    - Assim temos os vetores das palavras “cherry”, “strawberry”, “digital” e “information”. **Veja que**:
      - Os vetores de “digital” e “information” são razoavelmente similares e bem diferentes dos vetores de “cherry” e “strawberry”
      - Da mesma forma, os vetores “cherry” e “strawberry” são mais parecidos entre si do que com os outros 2 vetores;

	aadvark	aargh	...	computer	data	pie
cherry	0	0		2	8	442
strawberry	0	0		0	0	60
digital	0	0		1670	1683	5
information	0	0		3325	3982	5

Matriz Termo-Contexto (5/5)

- **Qual é a grande DESVANTAGEM da Matriz Termo-Contexto ?**
  - A matriz tem dimensão |V| x |V|, onde |V| é o tamanho do vocabulário.
  - Em aplicações reais, os vocabulários podem ter dezenas ou centenas de milhares de palavras.
    - E a maioria das palavras não ocorrerá no contexto de uma palavra-alvo.
      - Neste exemplo “aadvark” e “aargh” não coocorrem com “cherry”, “strawberry”, “digital” e “information”.

	aadvark	aargh	...	computer	data	pie
cherry	0	0		2	8	442
strawberry	0	0		0	0	60
digital	0	0		1670	1683	5
information	0	0		3325	3982	5

Embeddings (1/4)

- **O que são embeddings?**
  - Vetores das palavras como os gerados por Matrizes Termo-Contexto são chamados de **embeddings**.

cherry	0	0	...	2	1	442
strawberry	0	0	...	1	0	507
digital	0	0	...	1670	1683	5
information	0	0	...	3982	3982	5

Embeddings (2/4)

- **O que são embeddings?**
  - **A matriz termo-contexto** cria um modelo semântico distribucional baseado em **vetores longos e esparsos**...
    - ... porém os embeddings que são realmente **usados na prática** são os **vetores curtos e densos**
      - Word2Vec
      - Glove
      - FastText
      - Wang2Vec
      - Sense2Vec
      - ELMO
      - BERT
      - GPT
    - ... Em especial, os **vetores densos dinâmicos** como ELMO, BERT e GPT.
    - ... eles são criados através de métodos bem mais complexos (redes neurais)

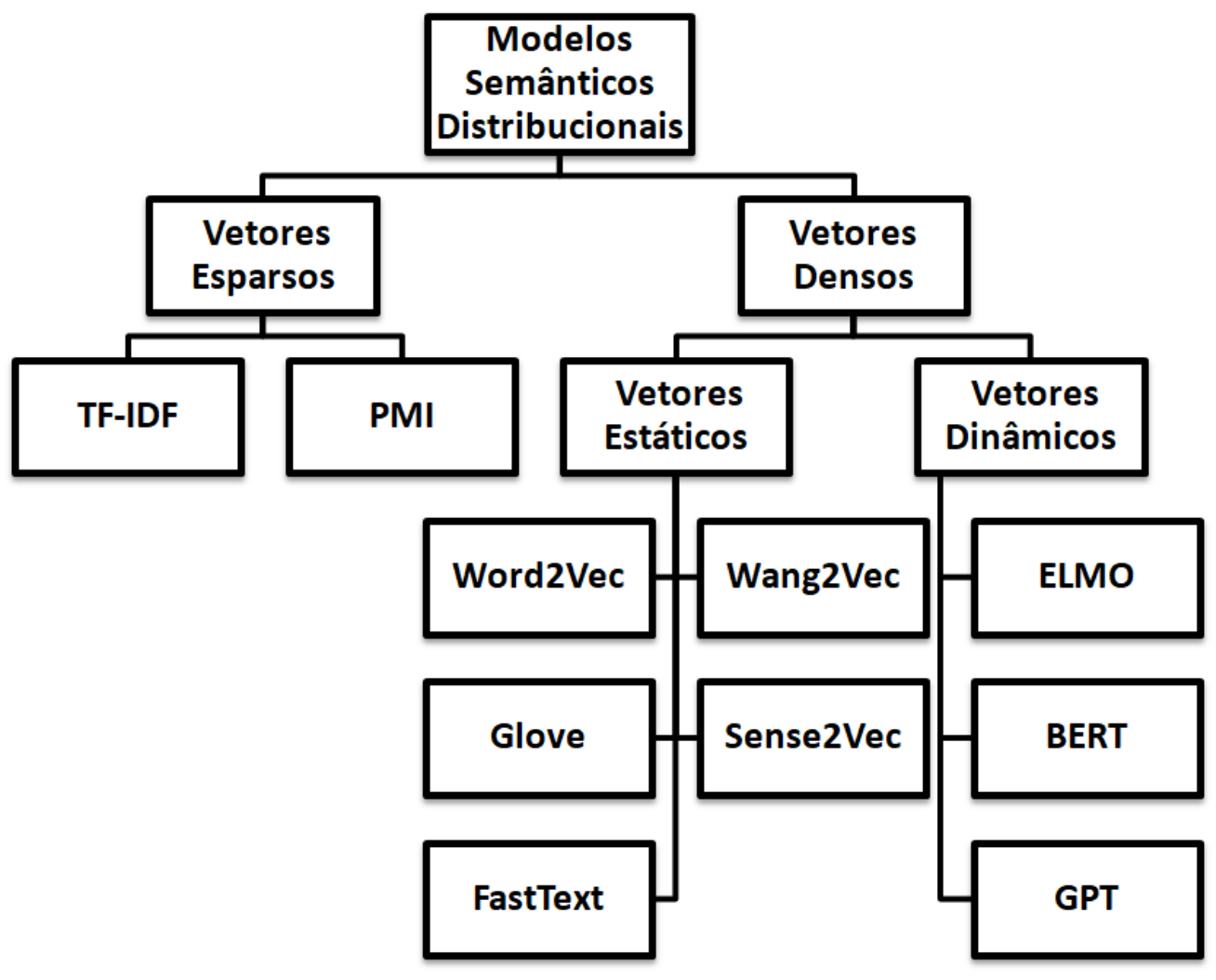
Embeddings (3/4)

- **Exemplo –** palavra "hora" representada em um embedding de 300 dimensões gerado pelo algoritmo Word2vec (implementação NILC (2017)).

hora = [-0.062545, -0.126489, 0.113524, -0.073889, 0.006757, -0.027824, 0.286723, -0.010828, -0.032953, 0.348775, -0.212591, 0.223134, 0.136070, -0.256647, 0.072911, 0.021405, -0.058130, 0.191910, -0.113131, 0.072543, 0.030004, -0.056953, -0.115852, 0.029836, -0.156374, -0.249230, 0.112478, 0.126747, -0.004281, 0.014156, -0.407328, 0.108729, 0.349468, -0.035015, -0.328122, -0.051109, 0.233072, 0.244711, 0.108190, 0.297134, -0.024735, 0.127016, -0.336429, 0.049182, 0.155089, 0.093664, -0.121086, -0.182766, 0.118789, -0.002741, -0.031592, 0.052193, 0.081095, -0.183748, 0.036250, 0.388335, -0.240749, -0.047723, -0.078823, 0.156688, 0.122014, 0.589334, -0.021866, 0.185283, -0.018192, 0.077296, 0.030083, -0.398973, -0.241083, 0.038450, 0.206490, 0.435420, 0.011289, -0.019493, -0.119840, 0.155041, 0.104586, 0.434048, -0.140947, -0.038420, 0.518088, -0.158477, 0.140301, 0.155320, -0.020636, -0.149190, 0.069280, -0.247552, -0.115318, 0.103906, 0.173289, -0.111075, 0.061043, 0.180365, -0.092135, -0.105528, 0.360078, -0.303879, -0.054269, -0.121504, 0.132912, -0.047233, -0.052588, 0.018701, -0.005428, -0.007330, -0.021943, -0.000987, -0.105333, 0.168648, -0.243485, 0.020719, -0.246331, -0.297241, 0.041788, 0.105140, -0.289586, 0.023823, -0.122424, -0.064835, -0.158461, 0.068709, -0.205565, 0.173924, 0.126506, -0.244338, -0.052390, 0.174168, 0.338677, -0.287746, -0.020279, -0.022267, 0.199124, 0.058610, -0.107843, 0.057305, 0.104822, 0.062944, 0.231721, -0.244904, -0.015706, 0.173425, -0.010017, -0.227111, 0.137199, 0.222197, -0.154434, 0.189224, 0.250377, 0.396699, -0.026948, 0.053932, -0.246668, 0.108724, 0.095354, 0.184613, 0.077367, 0.080714, -0.000680, -0.071466, -0.324820, -0.163270, 0.142114, -0.176308, -0.040815, 0.277833, -0.025246, 0.053511, 0.409635, -0.213821, -0.142188, 0.384299, 0.130455, 0.324051, -0.025810, -0.086175, -0.207605, 0.018398, 0.193626, -0.160563, -0.280345, 0.413318, -0.126175, 0.034025, -0.356754, 0.078491, 0.164689, 0.037496, 0.147475, 0.086021, 0.190551, -0.296555, -0.348180, -0.122142, 0.032044, 0.189108, -0.346120, 0.272183, 0.128225, 0.173495, -0.135757, -0.063786, 0.125983, 0.215387, -0.112882, 0.043228, -0.060512, -0.021911, 0.014611, 0.083601, 0.019959, 0.346055, 0.014662, 0.098009, 0.094097, 0.017535, -0.075331, -0.095261, -0.218828, 0.101617, -0.235558, 0.193594, -0.027302, -0.003942, 0.039598, -0.080496, -0.067256, 0.299944, 0.143414, -0.136063, -0.136811, -0.205818, -0.068009, 0.059440, 0.135297, -0.135802, -0.078074, -0.186340, -0.101303, 0.081397, 0.039796, 0.052596, -0.071562, 0.075672, -0.194597, 0.058802, -0.361304, -0.041861, 0.011388, -0.070250, 0.132716, -0.193836, -0.148685, 0.039821, 0.124415, -0.011547, 0.319272, 0.163388, 0.280234, -0.146267, 0.323278, -0.065721, -0.091195, 0.189873, 0.160153, 0.053370, 0.223984, -0.151240, 0.163842, -0.071410, 0.039385, 0.066962, 0.329253, -0.067227, 0.298047, -0.055179, 0.081621, 0.102730, -0.131966, -0.272213, 0.083461, 0.224348, 0.002535, 0.003754, 0.267152, 0.076403, 0.074424, 0.145011, -0.153571, 0.041310, 0.122828, 0.064069, 0.075629, 0.195021, -0.181588, -0.140206, 0.093154, -0.127453, -0.153316, -0.021281]

Embeddings (4/4)

- **Taxonomia de Modelos Semânticos Distribucionais**



Fonte: Seno et al. (2024)

Embeddings no Python: spaCy (1/8)

- **spaCy**
  - A ferramenta spaCy é junto com NLTK a mais popular ferramenta de PLN no Python
    - https://spacy.io/
  - Uma de suas mais interessantes características é o fato de que ela disponibiliza **embeddings fastText** para diversos idiomas, incluindo o Português.
    - fastText é um embedding da família: “curto denso estático”.
  - É bem fácil de usar !!! Os próximos passos apresentam o roteiro básico para trabalhar com embeddings na spaCy.

Embeddings no Python: spaCy (2/8)

- **fastText da spaCy: importando o modelo de linguagem**
  - Para usar o fastText na spaCy, primeiro temos que importar um dos modelos de linguagem para o Português que disponibilize embeddings.
  - No exemplo abaixo, importamos o "pt_core_news_lg", contendo 500.000 vetores de 300 posições (código para o Google Colab).
  - Após a importação, é preciso criar uma variável para armazenar o modelo de linguagem, como mostra o código abaixo.

```
# passo 1: importar o modelo de linguagem em Português
#          nesse caso: pt_core_news_lg (lg pq é large !)
import spacy
!python -m spacy download pt_core_news_lg

nlp = spacy.load("pt_core_news_lg")
```

Embeddings no Python: spaCy (3/8)

- **Modelo “pt_core_news_lg”**

LANGUAGE	PT Portuguese
TYPE	CORE Vocabulary, syntax, entities, vectors
GENRE	NEWS written text (news, media)
SIZE	LG 541 MB
COMPONENTS	`tok2vec`, `morphologizer`, `parser`, `lemmatizer`, `senter`, `attribute_ruler`, `ner`
PIPELINE	`tok2vec`, `morphologizer`, `parser`, `lemmatizer`, `attribute_ruler`, `ner`
VECTORS	500k keys, 500k unique vectors (300 dimensions)
DOWNLOAD LINK	pt_core_news_lg-3.8.0-py3-none-any.whl
SOURCES	UD Portuguese Bosque v2.8 </> (Rademaker, Alexandre; Freitas, Cláudia; de Souza, Elvis; Silveira, Aline; Cavalcanti, Tatiana; Evelyn, Wograine; Rocha, Luisa; Soares-Bastos, Isabela; Bick, Eckhard; Chalub, Fabricio; Paulino-Passos, Guilherme; Real, Livy; de Paiva, Valeria; Zeman, Daniel; Popel, Martin; Mareček, David; Silveira, Natalia; Martins, André) WikiNER (Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, James R Curran) Explosion fastText Vectors (cbow, OSCAR Common Crawl + Wikipedia) (Explosion)

Embeddings no Python: spaCy (4/8)

- **Exemplo 1: obtendo o vetor de uma palavra –** para acessar os vetores semânticos (embeddings), basta acessar a propriedade vector

```
v1 = nlp.vocab['cachorro'].vector
v2 = nlp.vocab['gosta'].vector
v3 = nlp.vocab['de'].vector
v4 = nlp.vocab['bolinha'].vector
# imprime apenas as 10 primeiras dimensões de cada vetor
# (mas são 300)
print('vetor "cachorro":', v1[:10], "\n")
print('vetor "gosta":', v2[:10], "\n")
print('vetor "de":', v3[:10], "\n")
print('vetor "bolinha":', v4[:10], "\n")
```

vetor "cachorro": [0.96452 -1.9547 0.50567 2.118 0.88336 -2.4307 0.57607 1.3785 -1.2825 1.2342]
vetor "gosta": [1.951 -4.1664 0.65357 3.6018 -2.7861 -0.80177 0.72013 1.8136 -0.13742 -4.2307]
vetor "de": [3.9609 4.5764 -2.5355 2.7397 -7.2996 -3.3364 -2.1117 -2.8205 -0.54067 -4.2687]
vetor "bolinha": [0.52471 -3.0086 -1.0966 0.89346 -0.48732 -1.9214 -0.13132 -0.59334 -0.52669 0.27043]

Embeddings no Python: spaCy (5/8)

- **<u>Exemplo 2</u>: Computando o vetor de uma frase ou texto**
  - O vetor de uma frase pode ser a média ou soma dos embeddings que compõem a frase
  - No exemplo abaixo, foi feita a média

```
# e qual é o vetor da frase "cachorro gosta de bolinha"?
# pode ser a média dos 4 vetores !!!
vetor_frase = (v1 + v2 + v3 + v4) / 4
print('vetor "cachorro gosta de bolinha":',
vetor_frase[:10], "\n")
```

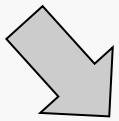
vetor "cachorro gosta de bolinha": [1.719105 -0.38617504 -0.344065 2.114875
-2.300585 -1.6422175 -0.203875 0.09290004 -0.49014747 -1.8163]

Embeddings no Python: spaCy (6/8)

- **Exemplo 3: e se eu quiser saber se o vetor de uma determinada palavra existe?**
  - Quando a palavra não é encontrada, a spaCy retorna 0 em todas as posições do vetor.

```
# quando palavra não é encontrada, retorna 0 em todas as posições
v5 = nlp.vocab['ence'].vector
print('vetor "ence":', v5[:10], "\n")

# você pode testar dessa forma
import numpy as np
if np.all(v5==0): print("v5 só tem zeros")
```



vetor "ence": [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

v5 só tem zeros

Embeddings no Python: spaCy (7/8)

- **<u>Exemplo 4</u>: similaridade entre 2 palavras**
  - É preciso aplicar o modelo sobre as palavras que você quer comparar

```
# similaridade entre duas palavras
palavras = 'arroz feijão estatística probabilidade matemática'
tokens = nlp(palavras)

print(tokens[0].similarity(tokens[1])) # arroz x feijão
print(tokens[0].similarity(tokens[2])) # arroz x estatística
print(tokens[0].similarity(tokens[3])) # arroz x probabilidade
print(tokens[2].similarity(tokens[3])) # estatística x probabilidade
print(tokens[2].similarity(tokens[4])) # estatística x matemática
```

0.8557679653167725
0.05946008116006851
0.11741837859153748
0.6280221939086914
0.7076966762542725

Retorna a similaridade de cosseno, que é um valor entre 0.0 e 1.0.

Quanto mais próximo de 1.0, mais similar o par de palavras.

Embeddings no Python: spaCy (8/8)

- **<u>Exemplo 5</u>: Obtendo os *n* vetores mais similares**
  - Utiliza-se o método most_similar. O código abaixo foi retirado do manual da spaCy.

```
import numpy as np

palavra = "estatística"

ms = nlp.vocab.vectors.most_similar(
    np.asarray([nlp.vocab.vectors[nlp.vocab.strings[palavra]]]),
                n=10)
words = [nlp.vocab.strings[w] for w in ms[0][0]]
distances = ms[2]
print(words)
print(distances)
```

['estatística', 'bioestatística', 'geoestatística', 'Geoestatística', 'estatístico', 'Bioestatística', 'estatistica', 'estatísticos', 'estatísticamente', 'Estatística']

[[1. 0.863 0.8248 0.8138 0.8103 0.8032 0.7915 0.7905 0.7734 0.7596]]

Embeddings no R

- **Alguns pacotes muito mencionados:**
  - text2vec
  - wordVectors
  - word2vec
  - text
  - embeddings do NILC-USP
    - “arquivão” com mais de 3,8 bilhões de embeddings gerados de textos de várias fontes em Português (jornais, livros etc.)
    - você pode baixar e usar como quiser.

Referências

- ALAMMAR, J. **The Illustrated Word2vec**. Github.io, 2019. Disponível em: < https://jalammar.github.io/illustrated-word2vec >. Acesso em: 26 mai. 2025.

- JURAFSKY, D.; MARTIN, J. H. Vector Semantics and Embeddings. *In*: JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing**. 3. ed. *draft* [*s.l.*] : [*s.n.*], 2025. cap 6. Disponível em: https://web.stanford.edu/~jurafsky/slp3/. Acesso em: 26 mai. 2025.

- NILC, **Repositório de Word Embeddings do NILC**. nilc.icmc.usp.br, 2017. Disponível em: < http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings-do-nilc >. Acesso em: 26 mai. 2025.

- PARRISH, A. Understanding word vectors. **Github.com**, 2018. Disponível em: < https://gist.github.com/aparrish/2f562e3737544cf29aaf1af30362f469 >. Acesso em: 26 mai. 2025.

- SENO, E. R. M. *et al*. Semântica distribucional. *In*: CASELI, H. M.; NUNES, M. G. V. **Processamento de Linguagem Natural: Conceitos, Técnicas e Aplicações em Português**. 3. ed. [*s.l.*]: BPLN, 2024. cap 10. Disponível em: < https://brasileiraspln.com/livro-pln/3a-edicao/parte-significado/cap-semantica-distribucional/cap-semantica-distribucional.html >. Acesso em: 26 mai. 2025.

Obrigado !!!!!