"You shall know a word by the company it keeps" (J. R. Firth 1957:11)

Introdução a Processamento Natural de Linguagem

Eloiza Rossetto

DSBD

UFPR

Introdução

- Processamento de Linguagem Natural (PLN) é uma área destinada a tornar máquinas capazes de compreender textos escritos em linguagem natural (ou outros tipos de texto)
- Vem ganhando cada vez mais importância devido aos resultados recentes com modelos como GPT, BERT

Algumas das tarefas especializadas:

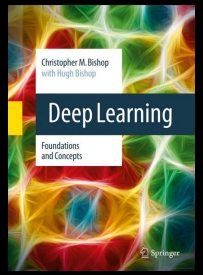
- Pergunta-Respostas
- Tradução
- Classificação
- Correção gramatical

Como representar texto?

Vimos até agora arquiteturas como MLP e CNNs que utilizam como entrada números. Sejam eles features criadas ou valores crus de pixels

Pensando nesses algoritmos e em outros vistos como podemos representar palavras, frases de maneira numérica?

“Bom dia” → | 1.231 | 0.812 | 0.001 | 1.00 |



→ | 1.412 | 0.973 | 0.401 | 0.301 |

Tokens, Corpus, Corpora,...

Definições fundamentais

Tokens: Unidade básica que compõe um corpus. Podem ser palavras mas não se limita a isso

Corpora: Conjunto de tokens. Pode ser o texto de um livro ou parágrafo ou outra fonte de origem.

Corpus: Plural de corpora, um conjunto de textos de determinada fonte.

Vocabulário: Tokens encontrados em determinado corpus

Pré-processamento

Antes de começarmos a trabalhar com texto é necessário passar por algumas etapas de pré-processamento para preparar esse dado.

Geralmente são feitas a seguintes etapas:

1. Tokenização
2. Normalização de palavras

Tokenização

A etapa de tokenização separa o texto em tokens. Nessa aula os tokens serão palavras porém podem ser outros segmentos de texto.

Uma das formas mais simples de fazer isso é utilizar pontuações e espaços em branco.

Vamos tokenizar algumas frases:

"Bom dia" Bom_dia

"A noite, ele acordou assustado" A_noite_ele_acordou_assustado

Tokenização

A tokenização com espaços e pontuação funciona muito bem até que você se depara com frases como

"Essa foi a gota d'agua para ele"

"Instale o mozilla firefox 131.0.2"

"Ligue para 9999-9999 e compre seu guarda-chuva por R$35,99!"

O tokenizador baseado em pontuação e espaços vai gerar:

"Essa_foi_a_gota_d_agua_para_ele"

"Instale_o_mozilla_firefox_131_0_2"

"Ligue_para_9999_9999_e_compre_seu_guarda_chuva_por_R_$_35_99_!"

Tokenização

Outro caso a ser considerado são conceitos com múltiplas palavras, como por exemplo:

- Belo Horizonte, Rio de Janeiro

Em tarefas como pergunta-resposta e reconhecimento de entidade essa outra forma de tokenização pode ajudar o modelo

Não existe certo ou errado, a escolha vai depender da aplicação e contexto do dataset!

Sempre verifique o resultado gerado pelo tokenizador escolhido

Algoritmos de Tokenização

A maioria deles vão ser baseados em expressões regulares (regex)

Prós

- Rápido
- Diversos padrões podem ser combinados
- Funciona na grande maioria dos casos

Contras

- Regex
- Só serão tokenizadas corretamente palavras com padrões já conhecidos

Algoritmos de Tokenização

Há também uma outra classe de algoritmos baseados em sub-strings

Dessa forma o tokenizador aprende a separar tokens sozinho reduzindo o número de unknown tokens

Esses tokenizadores são usados nos LLMs (Large Language Models)

Alguns dos algoritmos mais comuns:

- BytePair Enconding
- WordPiece Tokenization
- Unigram Tokenizer

Byte-pair Encoding

Usado no GPT, GPT-2, BART, RoBERTa

Ideia: Utilizar padrões de sub-string comuns para representar tokens e ao longo do treinamento juntar essas sub-strings em string maiores

Imagine um corpus com a seguinte frequência de palavras

Corpus:

[(“faz”, 10), (“paz”, 5), (“pai”, 12), (“pais”, 5)]

Byte-pair Encoding

Corpus:

[(“faz”, 10), (“paz”, 5), (“pai”, 12), (“pais”, 5)]

1 - Pré-tokenização: Reduzir para a menor substring possível. Inicializar o vocabulário

Corpus:

[(“f”, “a”, z”, 10), (“p”, “a”, z”, 5), (“p”, “a”, “i”, 12), (“p”, “a”, “i”, s”, 5)]

Vocabulário: [“f”, “a”, “z”, “p”, “i”, “s”]

Byte-pair Encoding

2- Encontrar token adjacentes mais comuns no texto

“fa”: [(“f”, “a”, z”, 10), (“p”, “a”, z”, 5), (“p”, “a”, “i”, 12), (“p”, “a”, “i”, s”, 5)] -› total 10

“az”: [(“f”, “a”, z”, 10), (“p”, “a”, z”, 5), (“p”, “a”, “i”, 12), (“p”, “a”, “i”, s”, 5)] -› total 15

“pa”: [(“f”, “a”, z”, 10), (“p”, “a”, z”, 5), (“p”, “a”, “i”, 12), (“p”, “a”, “i”, s”, 5)] -› total 22

3 - Criar nova regra para juntar “p” e “a”. Atualizar vocabulário e corpus

Corpus:

[(“f”, “a”, z”, 10), (“pa”, z”, 5), (“pa”, “i”, 12), (“pa”, “i”, s”, 5)]

Vocabulário: [“f”, “a”, “z”, “p”, “i”, “s”, “pa”]

Byte-pair Encoding

4 - Repetir 2 e 3 até ter um vocabulário de tamanho desejado. Faça você mesmo a próxima iteração!

Corpus:

[(“f”, “a”, z”, 10), (“pa”, z”, 5), (“pa”, “i”, 12), (“pa”, “i”, s”, 5)]

Vocabulário: [“f”, “a”, “z”, “p”, “i”, “s”, “pa”]

[(“f”, “a”, z”, 10), (“pa”, z”, 5), (“pa”, “i”, 12), (“pa”, “i”, s”, 5)]

Mais comum: “PAI” : 12 + 5 = 17 aparições

Byte-pair Encoding

[(“f”, “a”, z”, 10), (“pa”, z”, 5), (“pa”, “i”, 12), (“pa”, “i”, s”, 5)]

Mais comum: “PAI” : 12 + 5 = 17 aparições

Aplicando nova regra:

[(“f”, “a”, z”, 10), (“pa”, z”, 5), (“pai”, 12), (“pai”, s”, 5)]

Vocabulário: [“f”, “a”, “z”, “p”, “i”, “s”, “pa”, “pai”]

Byte-pair Encoding

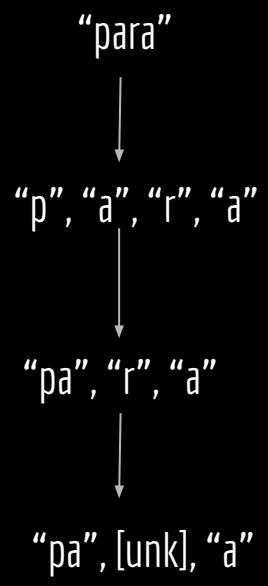
Com as regras que aprendemos e o vocabulário. Como seria a tokenização da palavra “para”?

Relembrando as regras:

Vocabulário: [“f”, “a”, “z”, “p”, “i”, “s”, “pa”, “pai”]

(p,a) -› “pa”

(pa, i) -› “pai”



Lematização e Stemming

Técnica para contrair palavras a sua forma base

Casas → Casa

Corações → Coração

Estive → Estar

Dessa forma não “alteramos” o significado de uma frase porém reduzimos drasticamente o vocabulário

Lematização e Stemming

A lematização é baseada em analisadores morfológicos que analisam a estrutura de uma frase identificando verbos, adjetivos, pronomes...

A partir dessa análise são criadas regras com regex para derivar formas base

Já o processo de stemming seria a versão mais simples da lematização. Muitas vezes algoritmos de stemming simplesmente cortam o final de palavras tentando encontrar a forma base

Faça você mesmo

Utilize o notebook e compare os tokenizadores, lematizadores e stimmers vistos em aula

Representações Vetoriais

Representação Vetorial

Com o que vimos até então palavras são apenas strings que compõem um vocabulário.

Essa representação não consegue extrair significado/sentido para essas palavras.

Para se entender o significado de uma palavra se deve compreender o ambiente em que a palavra está inserida.

Significado de palavras

Um sinonimo para a palavra água é H2O. Isso significa que podemos trocar a palavra água por H2O e ainda manter o mesmo significado para um frase. Exemplo:

Ele estava com sede e bebeu um copo de água -› Ele estava com sede e bebeu um copo de H2O

Mesmo que o sentido seja o mesmo o contexto em que utilizamos a palavra H2O é muitas vezes completamente diferente do que é utilizado o termo água.

Palavras com mesmo “significado” também podem ter conotações diferentes, como por exemplo nas palavras: falsificação, cópia, réplica.

TFIDF

Algoritmo para representação de texto a partir da contagem de palavras

Usando os dados do dataset newsgroup20

-Contém mais de 20 categorias de notícias

Selecionado as categorias de hardware, baseball, medicina e espaço. Essa é a contagem de algumas palavras em todo o dataset

	category_name	windows	computer	team	baseball	game	health	disease	space	lunar	science
0	comp.sys.ibm.pc.hardware	190	198	0	0	16	1	0	17	0	42
1	rec.sport.baseball	0	34	329	335	322	3	1	2	0	23
2	sci.med	26	154	3	4	1	264	210	17	0	296
3	sci.space	2	88	49	0	9	8	1	1400	229	174

TFIDF

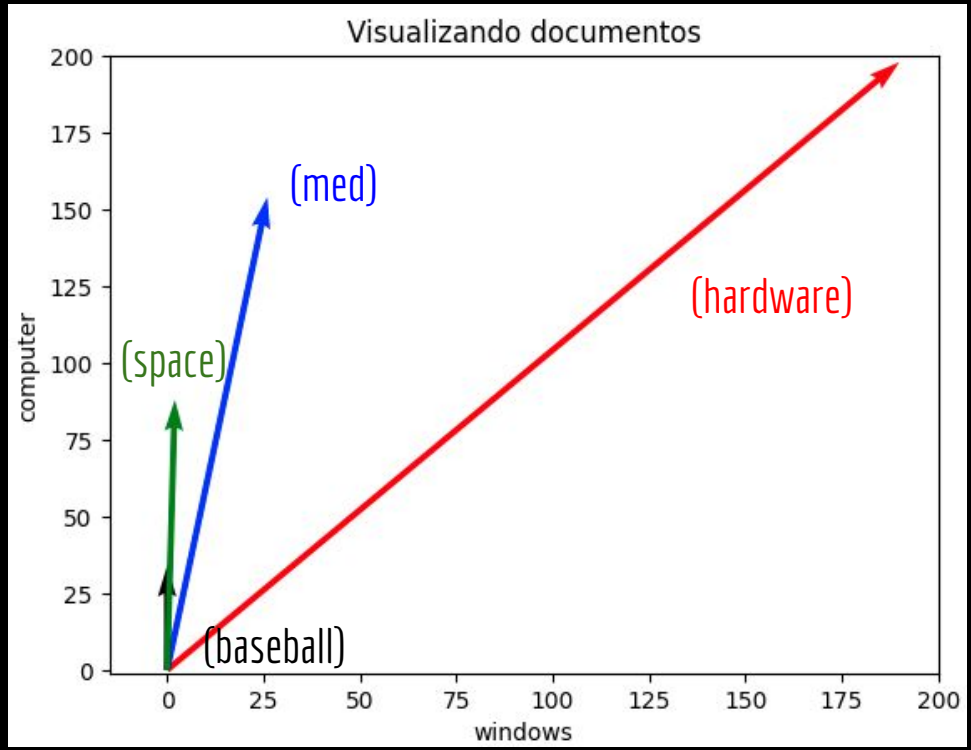
Podemos ter vetores para os tópicos olhando para a frequência das palavras

	category_name	windows	computer	team	baseball	game	health	disease	space	lunar	science
0	comp.sys.ibm.pc.hardware	190	198	0	0	16	1	0	17	0	42
1	rec.sport.baseball	0	34	329	335	322	3	1	2	0	23
2	sci.med	26	154	3	4	1	264	210	17	0	296
3	sci.space	2	88	49	0	9	8	1	1400	229	174

Se eu quero representar textos de hardware em um espaço de 10 dimensões. Os valores grifados seriam o valor para textos de hardware nesse espaço

TFIDF

Podemos visualizar parte desse espaço



TFIDF

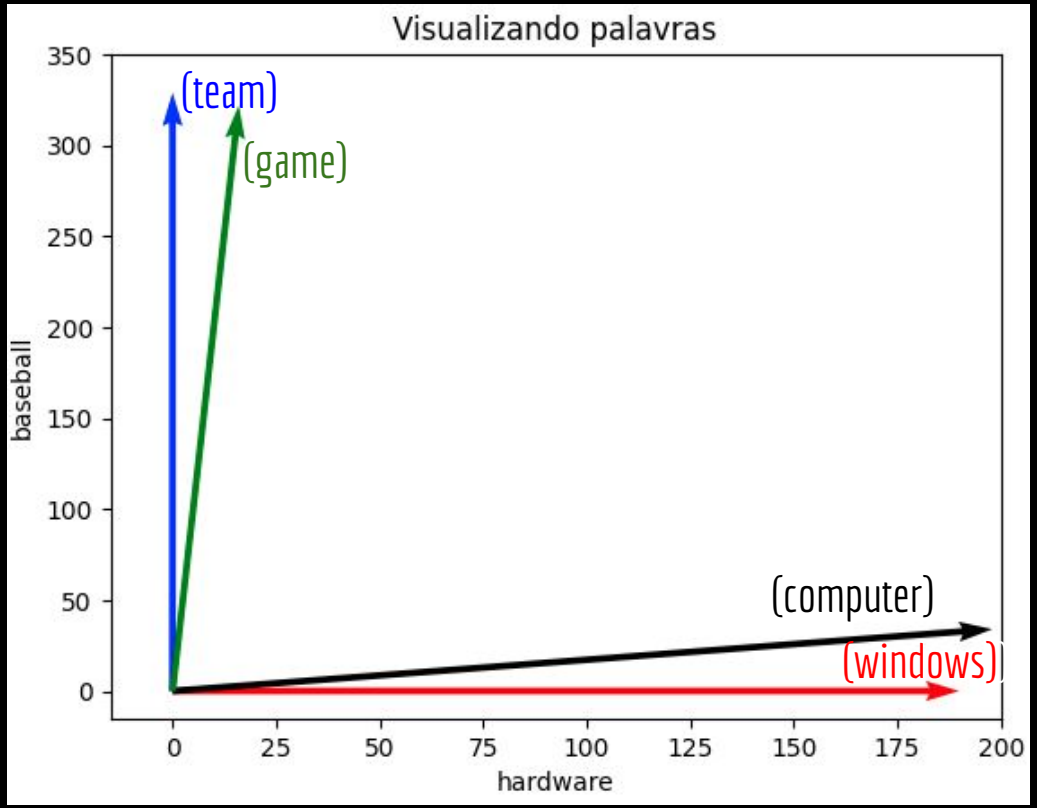
Podemos ter vetores para as palavras olhando para a frequência em cada tópico

	category_name	windows	computer	team	baseball	game	health	disease	space	lunar	science
0	comp.sys.ibm.pc.hardware	190	198	0	0	16	1	0	17	0	42
1	rec.sport.baseball	0	34	329	335	322	3	1	2	0	23
2	sci.med	26	154	3	4	1	264	210	17	0	296
3	sci.space	2	88	49	0	9	8	1	1400	229	174

Se eu representar palavras em um espaço de 4 coordenadas temos os valores grifados para a palavra windows.

TFIDF

Visualizando o espaço de palavras



TFIDF

Vetores baseados em frequência são uma boa ideia para representar palavras e documentos

Porém palavras muito frequentes não trazem informações relevantes para o nosso problema

Repare nas palavras mais comuns encontradas no dataset

Como balancear palavras de forma a manter as mais relevante?

the	25666
to	12880
of	12509
and	10815
in	9219

TFIDF

Term Frequency Inverse Document Frequency (TFIDF)

Term Frequency (TF) para um termo t em um documento d

$$TF(t, d) = \frac{\text{Número de vezes em que t aparece em d}}{\text{Número total de palavras em d}}$$

Inverse Document Frequency (IDF) para um termo t

$$IDF(t) = \log \frac{\text{Número total de documentos}}{\text{Número de documentos em que t aparece}}$$

$$TFIDF(t, d) = TF(t, d) * IDF(t)$$

Faça você mesmo

$$TFIDF(t, d) = TF(t, d) * IDF(t)$$

Calcule o TFIDF para o exemplo abaixo

palavra	doc1	doc2
the	10	6
ibm	3	0
game	0	20
total de palavras	100	200

$$TF(t, d) = \frac{\text{Número de vezes em que t aparece em d}}{\text{Número total de palavras em d}}$$

$$IDF(t) = \log \frac{\text{Número total de documentos}}{\text{Número de documentos em que t aparece}}$$

Faça você mesmo

Valores de TFIDF

palavra	doc1	doc2
the	0	0
ibm	0.0090	0
game	0	0.03
total de palavras	100	200

- O TFIDF gera uma representação esparsa do dados já que boa parte das colunas irá ter valor 0
- As matrizes geradas serão grandes pois dependem do tamanho do vocabulário e do total de documentos

Outras representações

Uma outra representação que podemos usar são representações densas em que:

- Os vetores são menores
- A maioria dos valores não é igual a 0
- Conseguem capturar melhor sinônimos

Esses vetores densos são muitas vezes chamados de **embeddings**

Alguns dos algoritmos:

- Word2Vec
- GloVe
- ELMo
- BERT

Vamos ver mais sobre o Word2Vec :)

Word2Vec

Ao invés de contarmos a frequência de uma determinada palavra em um texto vamos treinar uma rede para aprender correlações entre palavras

Para isso vamos treinar a rede para prever uma palavra em uma frase

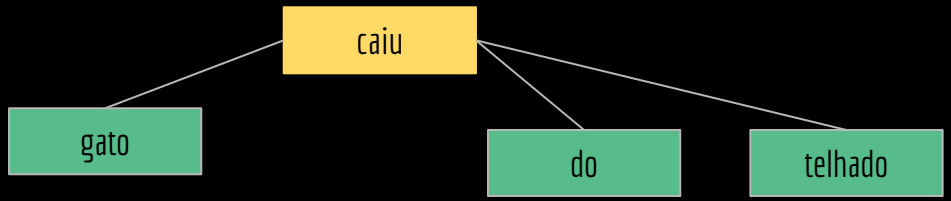
Nós não nos importamos com a tarefa de prever palavras apenas usamos essa tarefa para gerar embeddings por meio do aprendizado auto-supervisionado

Olha que paz: o modelo aprende sozinho sem a necessidade de dados rotulados, ou seja, podemos usar qualquer texto que existe na face da Terra para treinar o modelo

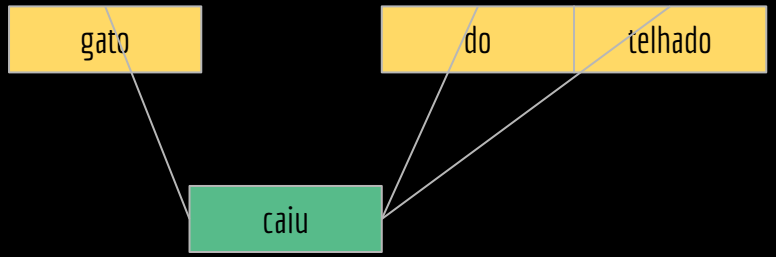
Word2Vec

Existem duas formas em que se pode montar o Word2Vec

Skip-gram Model



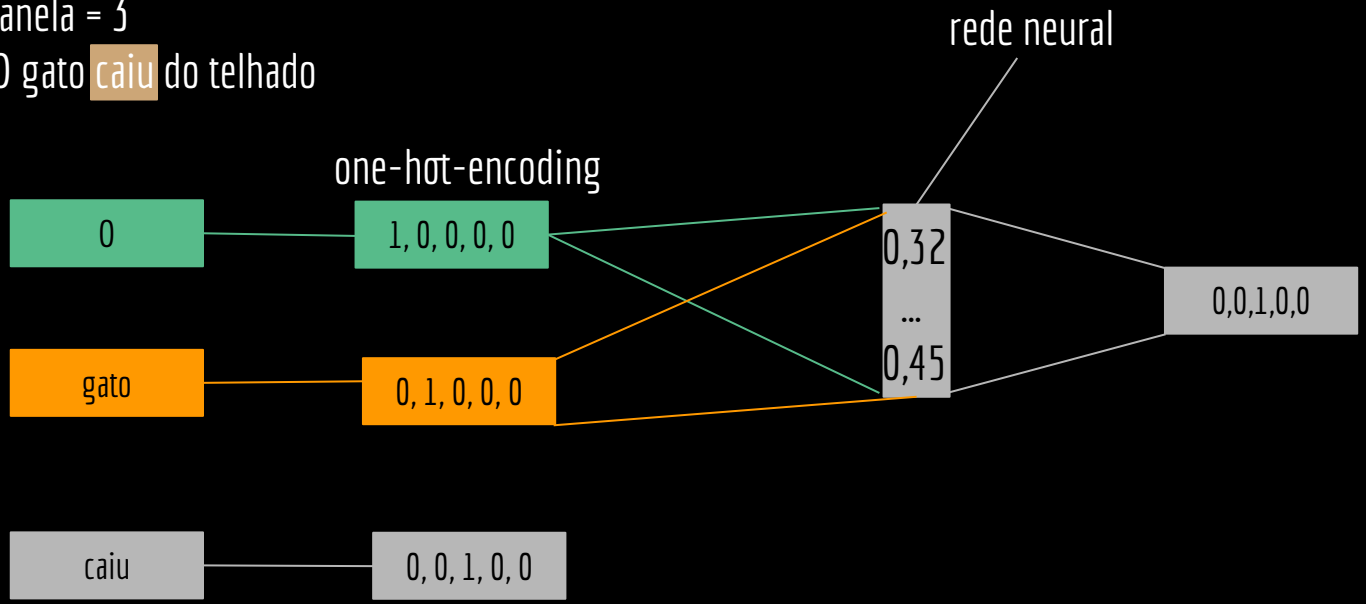
Continuous Bag of Words (CBOW)



Word2Vec (CBOW)

janela = 3

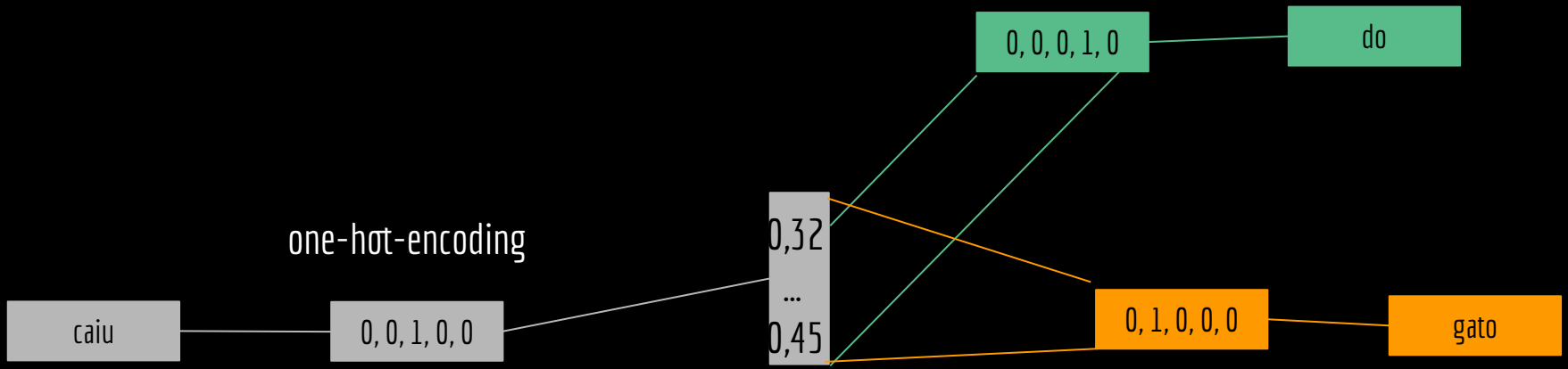
O gato caiu do telhado



Word2Vec (SkipGram)

janela = 1

O gato caiu do telhado



Word2Vec

O problema de treinamento do word2vec é maximizar a probabilidade de uma palavra P dado o contexto C. Ou seja:

maximizar(p(c|P))

Como calcular a probabilidade?

A intuição por trás do skipgram é basear a probabilidade na similaridade de dois embeddings. Uma palavra tem mais chances de ocorrer próximo ao seu objetivo se o embedding é similar ao objetivo.

Podemos definir p(c|P) com a similaridade de cosseno e a função softmax

$$p(c|P) = \frac{e^{(ut*v)}}{somatorio(e^{(ut*v)})}$$

Visualizando Embeddings

Ao final do processo teremos vetores que representam palavras
Uma das características mais interessantes é que palavras com significados parecidos ficam próximas nesse espaço vetorial que criamos

Visualizador de embeddings: https://projector.tensorflow.org/

Faça você mesmo

Treine o word2vec com o notebook disponibilizado

Faça você mesmo

1 - Faça seu regex para tratar emails como uma palavra só e aplique no dataset do word2vec. Exemplo: “O meu email favorito é meuemail@provedor.com” -› [O, meu, email, favorito, é, meuemail@provedor.com]

2 - Utilize a implementação do TFIDF do scikit-learning e aplique a função ao corpus do notebook de embedings. Compare os valores gerados pelo TFIDF com os gerados pelo word2vec

3-Compare os embeddings do treino do notebook com a implementação/treino do gensim(ref: https://radimrehurek.com/gensim/models/word2vec.html)

Referências

JURAFSKY, Daniel; MARTIN, James H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3. ed. 2024. Disponível em: https://web.stanford.edu/~jurafsky/slp3/. Acesso em 20 ago. 2024.

Licença