Tópico 2: Linguística computacional

Materiais	Vide
Due date	27 juillet 2023
Status	Done

Vídeos da aula 3: 3.1 a 3.6

Normalização de texto

- conjunto de tarefas computacioinais de pré-processmanto
- uniformiza os textos para serme processados de forma mais conveniente (contagem de palavras, sentenças, verificarmos os radicais,...) → tokenização, lematixação, radicalização e tokenização de sentenças

Tokenização

- visa distinguir as diferentes **unidades linguísticas** de um texto (tokens)
- tokenização em palavras é a forma mais comum

- “o que é uma palavra?”
  - fala do espaço como um delimitador:

```
texto = "No meio do caminho tinha uma pedra."

texto.split()
```

```
['No', 'meio', 'do', 'caminho', 'tinha', 'uma', 'pedra.']
```

Usando função em Python

  - pedra. não é uma palavra. por isso, usar a pontuação como um delimitador também:

```
texto = "No meio do caminho tinha uma pedra."

re.sub(r"(\b)", r" \1", texto).split()
```

```
['No', 'meio', 'do', 'caminho', 'tinha', 'uma', 'pedra', '.']
```

A primeira expressão \b vai ser substituída por “ ”.

  - ainda assim, essa tokenização não está perfeita
  - lembrando aqui que nem todas as línguas são delimitadas por espaços
- tokenizadores são baseados **em regras**
- ou seja, baseados em regras linguísticas de uma língua
- essas regras são codificadas em uma linguagem de programação a partir de expressões regulares
- exemplo de um tokenizador um pouco mais complexo:

```
texto = "Eu paguei R$456,00 pelo setup. O que acha?"
regex = r"R?\$?[\d\.\,]+|\w+|\S+"
re.findall(regex, texto)
```

```
['Eu', 'paguei', '456,00', 'pelo', 'setup', '.', 'Você', 'gostou', '?']
```

Tokenizador baseado em três regras codificadas por um expressão regular:

1. Buscar todas as ocorrências de valores numéricos e financeiros (R$1,00; $46; etc.)
2. Buscar todas as ocorrências de sequências de 1 ou mais caracteres
3. Buscar todas as ocorrências de sequências sem espaço

Exemplo de um tokenizador e a explicação das suas três regras

- Tokenizador do framwork NLTK (em Python)
    - dá pra fazer o import nltk no colab
    - lida com português, inglês, espanhol,...

Vídeo 2

- **types** vs tokens
    - fazer a contagem de types e tokens em um texto
    - ex: "eu sou eu"
        - types: "eu", "sou" (conta o número de palavras **únicas** em um texto)
        - tokens: "eu", "sou", "eu"
    - com essa contagem, é possível verificar a **variedade linguística** de um texto
- tokenizadores baseados em regras têm algumas limitações:
    - alto esforço manual para definir as regras
    - inadequado para algumas aplicações:
        - obter tokens que não estão em um vocabulário pré-estabelecido
        - ter um vocabulário muito extenso
- por isso, a ideia é fazer a tokenização em **sub palavras**
    - palavras muito frequentes não devem ser separadas
    - palavras raras podem ser segmentadas em subpalavras **frequentes**: frequentemente = frequente + mente
    - essa tokenização é baseada em dados e no estado da arte

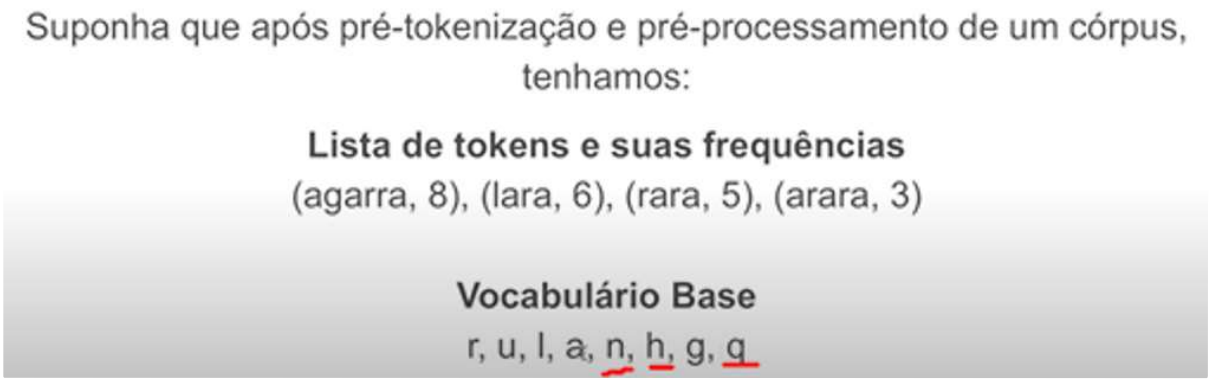
Alguns desses tokenizadores de subpalavras são:

Byte-Pair Encoding (BPE)

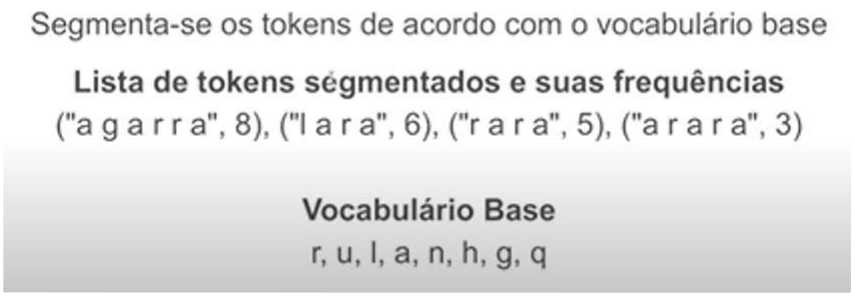
- proposto em 2015
- como são data driven, seu vocabulário é **treinado**
- seu funcionamento é baseado em sistema de compressão de arquivos:
- precisamos de um córpus de textos de treinamento
- esse córpus é pré-tokenizado (segmenta em palavras cada texto do córpus)
- a partir disso, uma lista de tokens únicos e suas frequências é criada
- um vocabulário base é criado com todos os símbolos (caracteres) que compõem os tokens

- por fim, aprende-se as regras de junção até que um vocabulário de tamanho pré-estabelecido seja obtido

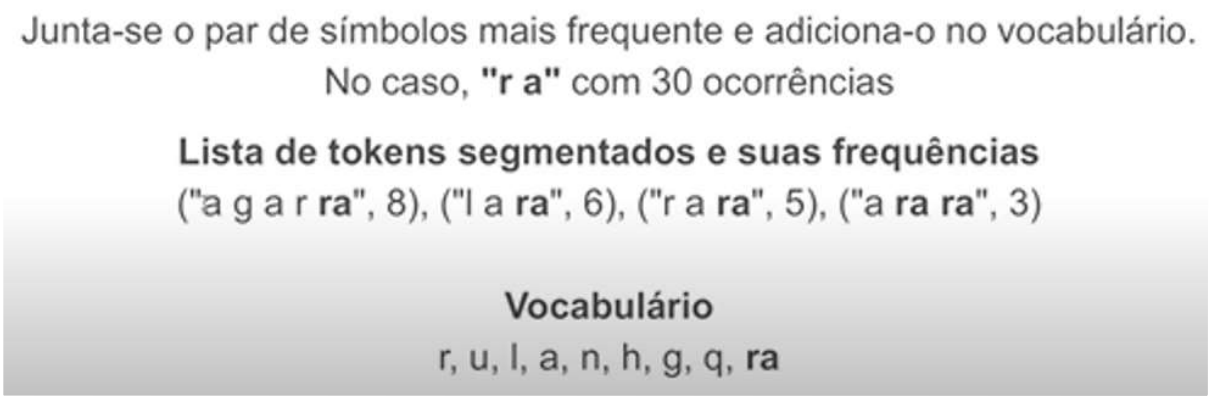
exemplo:



- os tokens são segmentados a partir do vocabulário base:



- contamos os pares de símbolos mais frequentes e aquele mais frequente é adicionado ao vocabulário:



- o processo continua, até que obtenhamos o vocabulário de tamanho desejado:

Continua-se com o processo de junção até obter um vocabulário de tamanho desejado (e.g. 10 tokens).
O próximo par de símbolos mais frequente é **"a ra"** com 14 ocorrências

Lista de tokens segmentados e suas frequências
("a g a r ra", 8), ("l **ara**", 6), ("r **ara**", 5), ("**ara** ra", 3)

Vocabulário
r, u, l, a, n, h, g, q, ra, **ara**

- agora que houve o treinamento, o vocabulário pode analisar novos textos:

Uma vez terminado o treinamento, o vocabulário obtido pode ser utilizado para tokenizar novos textos

Vocabulário
r, u, l, a, n, h, g, q, ra, ara

Texto
a garra rara

Texto Tokenizado
a g #a #r #ra r #ara

> ℹ️ importante notar que os subtokens são sinalizados por um símbolo especial (no caso, "#")

- no Python, precisamos do transformers

Limitação do BPE

- tokenizadores por subpalavras surgiram para resolver dois problemas dos tokenizadores tradicionar (vocabulário enxuto, que o BPE resolve, definimos o tamanho do vocabulário; no entanto, ainda acontece de alguns símbolos não estarem presentes no vocabulário)
- nesses casos (de símbolos não presentes), eles são marcados como desconhecidos (<unk>), que é uma operação muito custosa

- uma solução para isso é o Byte-level BPE:
    - todos os caracteres unicode são inseridos num vocabulário base
    - MAS existem mais de 100 mil caracteres assim
    - por isso, a ideia é representar todos os caracteres por bytes. assim, o vocabulário base tem tamanho de 256 bytes (e o treinamento é feito por bytes, e não caracteres):

Character		Binary code point	Binary UTF-8	Hex UTF-8
$	U+0024	010 0100	00100100	24
¢	U+00A2	000 1010 0010	11000010 10100010	C2 A2
ह	U+0939	0000 1001 0011 1001	11100000 10100100 10111001	E0 A4 B9
€	U+20AC	0010 0000 1010 1100	11100010 10000010 10101100	E2 82 AC
한	U+D55C	1101 0101 0101 1100	11101101 10010101 10011100	ED 95 9C
𐍈	U+10348	0 0001 0000 0011 0100 1000	11110000 10010000 10001101 10001000	F0 90 8D 88

    - importar o ByteLevelBPETokenizer em Python

WordPiece

- é utilizado no BERT
- outro tokenizador, é similar ao BPE:
- é pré-tokenizado, computa-se a frequência de cada um de seus tokens e é feita a junção de dois em dois símbolos. a diferença é o **método usado para juntar esses dois símbolos:**

$$\frac{P(x_2 \mid x_1)}{P(x_2)}$$

- import WordPiece e WordPieceTrainer

Unigram

- mais um tokenizador de palavras
- é diferente dos outros dois
- inicializa o vocabulário base com muitas palavras, subpalavras e caracteres (leva em conta símbolos com mais de um caractere)
- aos poucos, vai **reduzindo** o vocabulário até obter somente os símbolos mais importantes (até o tamanho desejado)

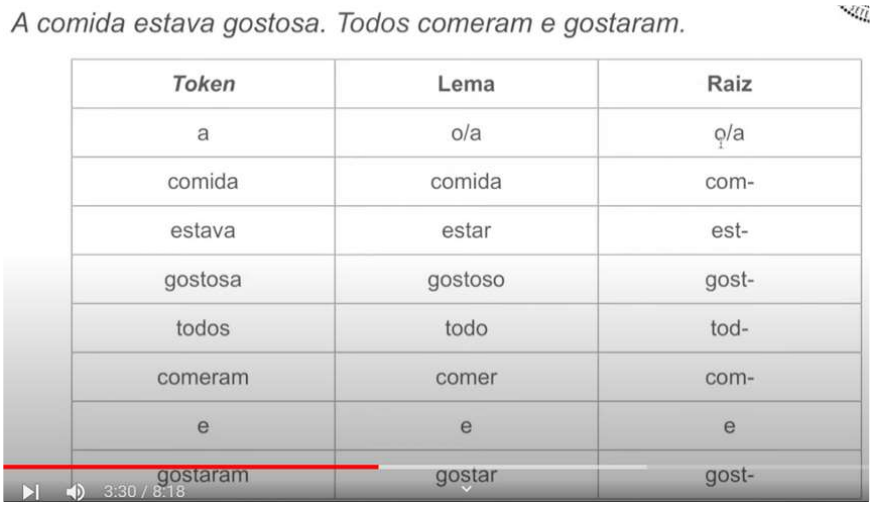
- o texto pode ser segmentados de múltiplas formas no treinamento
- SentencePiece
    - os outros tokenizadores usam pre-tokenizadores, que delimitam palavras por espaço
    - mas nem todas as línguas são delimitadas por espaço
    - o SentencePiece não usa um pré-tokenizador

Capitalização e tokenização de sentenças

- Capitalização: colocar os tokens em letra minúscula para a **normalização** do texto (.lower())
- Segmentação de sentenças: podemos levá-las em consideração como unidades linguísticas, e usá-las para contagem (tendo ponto de exclamação ou interrogação como delimitador, por exemplo). nltk.sent_tokenize();

Lexema, lema e raiz

- lexema: unidade abstrata de significado, corresponde a um conjunto de gormas relacionadas. {comeu, comido, comendo}
- lema: forma canônica. {comer}
- raiz: morfema, não se constitui uma palavra. {com}



pode cometer alguns erros. e se comida fosse um verbo?

- reparar que, com esses processos, os types diminuem
- RSLPStemmer: removedor de sufixos da língua portuguesa

Exercícios

- tokenização e ajuste de capitalização (lower)

- contagem da quatidade de types e tokens do texto (remover as pontuações)
- fazer rank dos 20 types mais frequentes (mostrando a frequência)
- fazer a lematização e a radicalização
- fazer rank dos 20 lemas e das 20 raízes mais frequentes

Seção 2.4.2 do livro An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition

Tokenização de palavra

- fala que na sequência de comando Unix todos os números e pontuações foram removidos, mas na maioria das vezes eles devem, sim, ser mantidos na tokenização
- pontuação muitas vezes são tokens separados, pontos finais ajudam a indeicar fins de sentenças
- fala sobre os preços, datas e os caracteres especiais que vêm com eles
- fala sobre a diferença do uso de vírgulas e pontos em diferentes localidades para demarcar separadores de milhar ou de decimais