Processamento de Linguagem Natural

Prof. Henrique Batista da Silva

PUC Minas Virtual

Agenda

- Tokenizando texto em palavras e n-gramas (tokens)
- Como lidar com pontuação e emoticons fora do padrão de como postagens nas redes sociais.
- Compactando vocabulário com stemming e lematização
- Construindo uma representação vetorial de uma instrução

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Construindo um vocabulário com tokenizador

PUC Minas Virtual

Tokenizador

- Primeiro passo para um sistema de NLP é um bom vocabulário
- Aqui iremos estudar algoritmos para separar uma string em palavras em pares, triplas, quádruplas e até quíntuplas. Estes pedaços são chamados de n-grams (n é igual ao tamanho dos pedaços)

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

- O uso de n-grams permite que sua máquina saiba sobre “ice cream”, bem como os “ice” e “cream”.

- No processamento de linguagem natural, compor um vetor numérico a partir de texto é um processo de extração de features "com perdas".

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

- No entanto, o bag-of-words (BOW), retêm o conteúdo de informações do texto suficiente para produzir modelos úteis de aprendizado de máquina.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

- A tokenização é um tipo específico de segmentação de documento. A segmentação divide o texto em pedaços ou segmentos menores.
- A segmentação pode incluir a quebra de um documento em parágrafos, parágrafos em sentenças, frases em tokens (geralmente palavras) e pontuação.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

- Em compiladores, um tokenizador usado para compilar linguagens de computador é chamado de lexer.
- O vocabulário (o conjunto de todos os tokens válidos) para uma linguagem de computador é de léxico.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

- Para um sistema de NLP, temos (equivalente aos compiladores):
  - Tokenizador: scanner, lexer, analisador lexical
  - Vocabulário: léxico
  - Analisador: compilador
  - token, termo, palavra ou n-gram: token, símbolo ou símbolo terminal

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

- A tokenização é a primeira etapa de um pipeline de NLP.
- Objetivo: dividir dados não estruturados (texto em linguagem natural) em chunks de informação discretos.
- Essas contagens de ocorrências de tokens em um documento podem ser usadas diretamente como um vetor que representa este documento

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

- Ideia: transforma uma sequência não estruturada (documento textual) em uma estrutura de dados numérico adequada para aprendizado de máquina.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

- Codificação: A maneira mais simples de tokenizar uma frase é usar espaços em branco em uma string como o "delimitador" das palavras (próximo slide).

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

```
>>> sentence = """Thomas Jefferson began building Monticello at the
...   age of 26."""

>>> sentence.split()
['Thomas',
 'Jefferson',
 'began',
 'building',
 'Monticello',
 'at',
 'the',
 'age',
 'of',
 '26.']
```

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

- Podemos agora criar uma representação vetorial numérica para cada palavra. Uma sequência desses vetores captura completamente o texto do documento original em uma tabela de números.

- Assim, resolvemos p primeiro problema da NLP, transformar palavras em números:

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

```
import numpy as np

sentence = """Thomas Jefferson began building Monticello at the age of 26."""

token_sequence = str.split(sentence)
vocab = sorted(set(token_sequence))

', '.join(vocab)

num_tokens = len(token_sequence)
vocab_size = len(vocab)
onehot_vectors = np.zeros((num_tokens, vocab_size), int)

for i, word in enumerate(token_sequence):
    onehot_vectors[i, vocab.index(word)] = 1
    ' '.join(vocab)

print(onehot_vectors)
```


Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

```
import numpy as np
import pandas as pd

sentence = """Thomas Jefferson began building Monticello at the age of 26."""

token_sequence = str.split(sentence)
vocab = sorted(set(token_sequence))

', '.join(vocab)

num_tokens = len(token_sequence)
vocab_size = len(vocab)
onehot_vectors = np.zeros((num_tokens, vocab_size), int)

for i, word in enumerate(token_sequence):
    onehot_vectors[i, vocab.index(word)] = 1
    ' '.join(vocab)

print(onehot_vectors)

print(pd.DataFrame(onehot_vectors, columns=vocab))
```

Permite atribuir rótulos de cada coluna

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

	26.	Jefferson	Monticello	Thomas	age	at	began	building	of	the
0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	0	1	0	0
4	0	0	1	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0
9	1	0	0	0	0	0	0	0	0	0

Nesta representação do seu documento, cada linha é um vetor para uma única palavra (veja o exemplo destacado para a palavra "age").

Podemos usar o vetor [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0] para representar a palavra "age" no pipeline.

Um "1" em uma coluna indica uma palavra de vocabulário que estava presente nessa posição no documento.

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Bag of Words

Bag of Words

- Até aqui criamos uma representação com apenas um único documento (a frase utilizada)
- Para ficar mais clara a ideia de bag-of-words precisamos trabalhar com vários documentos para termos um vocabulário (conjunto de palavras) significativamente maior.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Bag of Words

- Vamos agrupar em um DataFrame do Pandas para poder adicionar mais frases ao "corpus" de vetor binário de textos sobre Thomas Jefferson.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

```
import numpy as np
import pandas as pd

sentence = """Thomas Jefferson began building Monticello at the age of 26."""

df = pd.DataFrame(pd.Series(dict([(token, 1) for token in sentence.split()])), columns=['sent']).T

print(df)
```

```
Thomas  Jefferson  began  building  Monticello  at  the  age  of  26.
sent         1         1         1           1   1    1    1   1    1
```

Observe o resultado com todos os valores 1 para o primeiro documento (sentença). Isto ocorre porque nosso vocabulário está limitado (por enquanto) à apenas uma única sentença.

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Bag of Words

- Vamos adicionar mais alguns textos ao seu corpus para ver como um DataFrame se comporta.
- Um DataFrame indexa as colunas (documentos) e as linhas (palavras), para que possa ser um "índice invertido" para recuperação de documentos.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

```
import numpy as np
import pandas as pd

sentences = """Thomas Jefferson began building Monticello at the age of 26.\n"""
sentences += """Construction was done mostly by local masons and carpenters.\n"""
sentences += """He moved into the South Pavilion in 1770.\n"""
sentences += """Turning Monticello into a neoclassical masterpiece was Jefferson's obsession."
""


corpus = {}
for i, sent in enumerate(sentences.split('\n')):
    corpus['sent{}'.format(i)] = dict((tok, 1) for tok in sent.split())

df = pd.DataFrame.from_records(corpus).fillna(0).astype(int).T
print(df[df.columns[:]])
```


Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Tokenizador

	Thomas	Jefferson	began	...	masterpiece	Jefferson's	obsession.
sent0	1	1	1	...	0	0	0
sent1	0	0	0	...	0	0	0
sent2	0	0	0	...	0	0	0
sent3	0	0	0	...	1	1	1

[4 rows x 32 columns]

Observe o resultado (várias colunas foram omitidas). Cada documento (sentença) possui apenas o valor “1” na posição do vocabulário em que há ocorrência daquela palavra.

Observe que a dimensão do vetor é 32.

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Bag of Words

- Na resposta anterior, notamos pouca sobreposição no uso de palavras para os documentos (poucas palavras aparecem em mais de uma sentença).
- Agora iremos calcular essa sobreposição no pipeline para comparar documentos ou procurar documentos semelhantes, contando o número de tokens sobrepostos usando um produto escalar.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Produto escalar

Produto escalar

- O produto escalar utiliza vetores de dimensões (tamanhos) iguais.
- Sua operação produz um único valor escalar como saída.
- O valor escalar gerado pelo produto escalar pode ser calculado multiplicando todos os elementos de um vetor por todos os elementos de um segundo vetor.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Produto escalar

```
import numpy as np
import pandas as pd

v1 = np.array([1, 2, 3])
v2 = np.array([2, 3, 4])

print(v1.dot(v2))

print((v1 * v2).sum())

print(sum([x1 * x2 for x1, x2 in zip(v1, v2)]))
```

Operação mais eficiente

Bem menos eficiente

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Medindo o overlap entre bag-of-words

- Se pudermos medir a sobreposição de um conjunto de palavras para dois vetores, podemos obter uma boa estimativa de quão similares elas são.
- E esta é uma boa estimativa de quão similares elas são no significado.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Overlap entre bag-of-words

```
import numpy as np
import pandas as pd

sentences = """Thomas Jefferson began building Monticello at the age of 26.\n"""
sentences += """Construction was done mostly by local masons and carpenters.\n"""
sentences += """He moved into the South Pavilion in 1770.\n"""
sentences += """Turning Monticello into a neoclassical masterpiece was Jefferson's obsession."""

corpus = {}
for i, sent in enumerate(sentences.split('\n')):
    corpus['sent{}'.format(i)] = dict((tok, 1) for tok in sent.split())

df = pd.DataFrame.from_records(corpus).fillna(0).astype(int).T

df = df.T
print(df.sent0.dot(df.sent1))
print(df.sent0.dot(df.sent2))
print(df.sent0.dot(df.sent3))
```

Medindo a sobreposição (overlap) entre a sentença 0 e a sentença 1

Veja a resposta.

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Medindo o overlap entre bag-of-words

- Veja que um palavra foi usada em sent0 e sent2 (resposta 1).
- Da mesma forma, uma das palavras do vocabulário foi usada em sent0 e sent3.
- Essa sobreposição de palavras é uma medida de similaridade.
- A frase sent1 foi a única que não mencionou Jefferson ou Monticello diretamente, mas usou um conjunto de palavras completamente diferente.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Medindo o overlap entre bag-of-words

- Então, vamos a uma maneira de encontrar a palavra que é compartilhada por sent0 e sent3:

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Overlap entre bag-of-words

```
import numpy as np
import pandas as pd

sentences = """Thomas Jefferson began building Monticello at the age of 26.\n"""
sentences += """Construction was done mostly by local masons and carpenters.\n"""
sentences += """He moved into the South Pavilion in 1770.\n"""
sentences += """Turning Monticello into a neoclassical masterpiece was Jefferson's obsession."""

corpus = {}
for i, sent in enumerate(sentences.split('\n')):
    corpus['sent{}'.format(i)] = dict((tok, 1) for tok in sent.split())

df = pd.DataFrame.from_records(corpus).fillna(0).astype(int).T

df = df.T
print(df.sent0.dot(df.sent1))
print(df.sent0.dot(df.sent2))
print(df.sent0.dot(df.sent3))

print([(k, v) for (k, v) in (df.sent0 & df.sent3).items() if v])
```


Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Medindo o overlap entre bag-of-words

- Este modelo visto aqui é uma forma de extração de features de frases.
- Além de produtos escalares, outras operações de vetores podem ser usadas nestes casos: adição, subtração, OR, AND e assim por diante. Ou até mesmo distância euclidiana ou distância dos cossenos entre esses vetores.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Melhorando a extração de tokens

- Agora podemos melhorar a extração de tokens separando melhor as palavras usando expressão regular (até então usamos apenas espaços em branco)

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Overlap entre bag-of-words

```
import re

sentence = """Thomas Jefferson began building Monticello at the age of 26."""

pattern = re.compile(r"([-\s.,;!?])+")
tokens = pattern.split(sentence)

print(tokens)

tokens = [x for x in tokens if x and x not in '- \t\n.,;!?']

print(tokens)
```

Além de espaços em branco, usamos vários caracteres para fazer a separação dos termos.

Veja aqui todos os termos (com pontuação)

Veja aqui todos os termos (apenas os termos sem pontuação)

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Melhorando a extração de tokens

- Vamos a mais um exemplo, agora usando a biblioteca NLTK para extração de texto informais (como textos de rede sociais)

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Overlap entre bag-of-words

```
from nltk.tokenize.casual import casual_tokenize

message = """RT @TJMonticello Best day everrrrrrr at Monticello. Awesommmmmmeeeeeeee day :*)"""

tokens = casual_tokenize(message)
print(tokens)
```

Veja aqui todos os termos da forma bruta

```
tokens = casual_tokenize(message, reduce_len=True, strip_handles=True)
print(tokens)
```

Veja aqui todos os termos melhorados

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Ampliando o vocabulário com n-grams

PUC Minas Virtual

N-grams

- Um n-gram é uma sequência que contém até n elementos que foram extraídos de uma sequência. Em geral, os "elementos" de um n-gram podem ser caracteres, sílabas, palavras ou até símbolos como "A", "T", "G" e "C" usados para representar uma sequência de DNA.

- Aqui vamos focar nos n-gram de palavras (e não caractere).

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

N-grams

```
import re
from nltk.util import ngrams

sentence = """Thomas Jefferson began building Monticello at the age of 26."""
pattern = re.compile(r"([-\s.,;!?])+")

tokens = pattern.split(sentence)
tokens = [x for x in tokens if x and x not in '- \t\n.,;!?']

print(tokens)
```

Aqui está o tokenizador de 1-gram original

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

N-grams

```
import re
from nltk.util import ngrams

sentence = """Thomas Jefferson began building Monticello at the age of 26."""
pattern = re.compile(r"([-\s.,;!?])+")

tokens = pattern.split(sentence)
tokens = [x for x in tokens if x and x not in '- \t\n.,;!?']

print(tokens)

tokens = list(ngrams(tokens, 2))

print(tokens)
```


Aqui está o tokenizador de 2-gram

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

N-grams

- Os n-grams são obtidos na lista anterior como tuplas, mas podem ser unidos se você desejar que todos os tokens do pipeline sejam cadeias de caracteres.
- Isso permitirá que os estágios posteriores do pipeline esperem um tipo de dados consistente como entrada, sequências de strings:

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

N-grams

```
import re
from nltk.util import ngrams

sentence = """Thomas Jefferson began building Monticello at the age of 26."""
pattern = re.compile(r"([-\s.,;!?])+")

tokens = pattern.split(sentence)
tokens = [x for x in tokens if x and x not in '- \t\n.,;!?']

print(tokens)

two_grams = list(ngrams(tokens, 2))
tokens = [" ".join(x) for x in two_grams]

print(tokens)
```


Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stop Words

Stop words

- Stopwords são palavras comuns em qualquer idioma que ocorram com alta frequência, mas carregam muito menos informações substantivas sobre o significado de uma frase.
- Exemplos (língua inglesa): a, na, the, this, and, or, of, on
- Exemplos (língua portuguesa): a, o, de, do, da, em, etc.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stop words

- Stopwords são palavras comuns em qualquer idioma que ocorram com alta frequência, mas carregam muito menos informações substantivas sobre o significado de uma frase.
  - Não são bons discriminadores dos documentos relevantes para uma consulta.
- Exemplos (língua inglesa): a, na, the, this, and, or, of, on
- Exemplos (língua portuguesa): a, o, de, do, da, em, etc.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stop words

- Podemos filtrar arbitrariamente um conjunto de stopwords durante a tokenização.
- Aqui utilizamos alguns stopwords que serão ignorados ao percorrer a lista de tokens:

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stop Words

```
stop_words = ['a', 'an', 'the', 'on', 'of', 'off', 'this', 'is']

tokens = ['the', 'house', 'is', 'on', 'fire']
tokens_without_stopwords = [x for x in tokens if x not in stop_words]

print(tokens_without_stopwords)
```

Lista de stopwords

Tokens extraídos de uma sentença

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stop words

- Para obter uma lista completa de stopwords, a biblioteca NLTK é provavelmente uma lista mais aplicável. Veja a lista a seguir.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stop Words NLTK

```
import nltk

nltk.download('stopwords')
stop_words = nltk.corpus.stopwords.words('english')

print(len(stop_words))

stopwords = stop_words[:7]
['i', 'me', 'my', 'myself', 'we', 'our', 'ours']
```


Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stop Words Sklearn

```
import nltk
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS as skl
earn_stop_words

nltk.download('stopwords')
stop_words = nltk.corpus.stopwords.words('english')

print(len(stop_words))

print(len(sklearn_stop_words))
```

A lista de stopwords consideradas pelo sklearn é um pouco maior

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Normalização do vocabulário

PUC Minas Virtual

Normalização do vocabulário

- Técnica de redução de vocabulário: normalizar o vocabulário para que tokens que significam algo semelhante sejam combinados em uma única forma normalizada.
- Isso reduz o número de tokens.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Case folding

- A normalização de letras maiúsculas e minúsculas é uma maneira de reduzir o tamanho do seu vocabulário e generalizar o pipeline de NLP.
- Ajuda a consolidar palavras cujo objetivo é significar a mesma coisa (e que são escritas da mesma maneira) sob um único token.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Case folding

- A normalização de letras maiúsculas e minúsculas é uma maneira de reduzir o tamanho do seu vocabulário e generalizar o pipeline de NLP.
- Ajuda a consolidar palavras cujo objetivo é significar a mesma coisa (e que são escritas da mesma maneira) sob um único token.

```
tokens = ['House', 'Visitor', 'Center']
normalized_tokens = [x.lower() for x in tokens]
print(normalized_tokens)
```

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Case folding

- Para um mecanismo de pesquisa sem normalização, se pesquisasse “Age", obteria um conjunto de documentos diferente do que se pesquisasse “age".
- Ao normalizar o vocabulário no índice de pesquisa (assim como na consulta), garantimos que os dois tipos de documentos sobre “age" sejam retornados, independentemente da capitalização na consulta do usuário.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stemming

- Outra técnica comum de normalização de vocabulário é eliminar as pequenas diferenças de significado
- Ou seja, manter apenas a porção de uma palavra que resta após a remoção de prefixos e sufixos
- A ideia é identificar um radical comum entre as várias formas de uma palavra (radicalização - stemming).
- Por exemplo, as palavras “housing” e “houses” compartilham o mesmo porção, “house”.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stemming

- O stemming remove sufixos das palavras na tentativa de combinar palavras com significados semelhantes juntos sob o seu radical comum.
- Vamos a uma implementação simples de stemmer em Python puro que pode lidar com “s” à direita (língua inglesa):

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stemming

```
import re

def stem(phrase):
    return ' '.join([re.findall('^(.*ss|.*?)(s)?$', word)[0][0].strip("'") for word
in phrase.lower().split()])

print(stem('houses'))

print(stem("Doctor House's calls"))
```

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Stemming

- Dois dos algoritmos de stemming mais populares são o Porter e Snowball.
- Esses algoritmos implementam regras mais complexas do uma simples expressão regular. Isso permite que o stemmer lide com as complexidades das regras de ortografia e final de palavras em inglês:

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Algoritmo Porter

```
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()

text = ' '.join([stemmer.stem(w).strip("'") for w in "dish washer's washed dishes".split()])

print(text)
```

Saiba mais em: https://github.com/jedijulia/porter-stemmer/blob/master/stemmer.py

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Lemmatization

- Se tiver acesso a informações sobre conexões entre os significados de várias palavras, é possível associar várias palavras, mesmo que a grafia seja bem diferente.
- Essa normalização que considera semântica de uma palavra - seu lema - é chamada lematização.
- Mais precisa do que técnicas anteriores (usa base de conhecimento de sinônimos)

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Lemmatization

- Alguns lematizadores usam a tag part of speech (POS) da palavra, além da ortografia, para melhorar a precisão.
- A tag POS para uma palavra indica seu papel na gramática de uma frase ou frase.
- Por exemplo, o POS substantivo é para palavras que se referem a "pessoas, lugares ou coisas" em uma frase. Um POS adjetivo é para uma palavra que modifica ou descreve um substantivo.
- Um verbo se refere a uma ação.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Lemmatization

- O pacote NLTK fornece funções para identificar lemas de palavras.
- Observe que devemos informar ao WordNetLemmatizer em qual POS você está interessado (substantivo, adjetivo, etc.).
- O lematizador NLTK está usa um grafo de significado de palavras do Princeton WordNet.

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Lemmatization

```
import nltk
nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print(lemmatizer.lemmatize("better"))
print(lemmatizer.lemmatize("better", pos="a"))
print(lemmatizer.lemmatize("good", pos="a"))
print(lemmatizer.lemmatize("goods", pos="a"))
print(lemmatizer.lemmatize("goods", pos="n"))
print(lemmatizer.lemmatize("goodness", pos="n"))
print(lemmatizer.lemmatize("best", pos="a"))
```

Por default, usa-se 'n' para os substantivos

'a' indica adjetivo. Neste caso, veja que mudou o valor da palavra goods

Fonte: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Considerações Finais

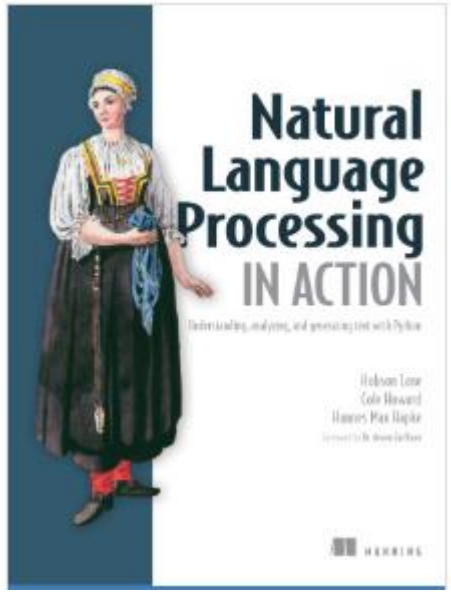
PUC Minas Virtual

Tópicos estudados

- Tokenizando texto em palavras e n-gramas (tokens)
- Como lidar com pontuação e emoticons fora do padrão de como postagens nas redes sociais.
- Compactando vocabulário com stemming e lematização
- Construindo uma representação vetorial de uma instrução

Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Principais Referências



Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action:Understanding, analyzing, and generating text with Python**. March 2019

PUC Minas
Virtual