Recuperação de Informações - Parte 1

Júlio César Batista

FURB - Universidade de Blumenau

Fevereiro, 2019

Agenda

- Expressões Regulares
- Web Scraping
  - HTTP
  - scrapy
- Motores de Busca
  - Matriz termo-documento
  - Índices
  - Consultas *booleanas*

Expressões Regulares - Introdução

Extrair informação estruturada de textos semi-estruturados.

Top livros Projeto Gutenberg

1. The Works of Edgar Allan Poe, The Raven Edition by Edgar Allan Poe (1525)
2. Pride and Prejudice by Jane Austen (1302)

Expressões Regulares - Introdução

Extrair informação estruturada de textos semi-estruturados.

Top livros Projeto Gutenberg

1. **The Works of Edgar Allan Poe, The Raven Edition** by **Edgar Allan Poe** (1525)
2. **Pride and Prejudice** by **Jane Austen** (1302)

Expressões Regulares - Introdução

Extrair informação estruturada de textos semi-estruturados.

Top livros Projeto Gutenberg

1. **The Works of Edgar Allan Poe, The Raven Edition** by **Edgar Allan Poe (1525)**
2. **Pride and Prejudice** by **Jane Austen (1302)**

Posição	Título	Autor	Visualizações
1	The Works of Edgar Al...	Edgar Allan Poe	1525
2	Pride and Prejudice	Jane Austen	1302

Web scraping

Motores de busca

- Otimizar a busca/consulta de documentos
- Busca simples pode ser feita com *grep*
- Como efetuar uma busca em bilhões de documentos e segundos?

Vocabulário

- *documento*: informação que será referenciada no índice
  - livro, capítulo, página, parágrafo, frase, página página da web?
  - indexar um livro pode gerar resultados indesejados
  - indexar um livro faz com que o usuário tenha que procurar no livro pela informação
  - indexar frases pode evitar que resultados sejam gerados porque as palavras não ocorreram na mesma frase

Vocabulário

- *documento*: informação que será referenciada no índice
  - livro, capítulo, página, parágrafo, frase, página página da web?
  - indexar um livro pode gerar resultados indesejados
  - indexar um livro faz com que o usuário tenha que procurar no livro pela informação
  - indexar frases pode evitar que resultados sejam gerados porque as palavras não ocorreram na mesma frase
- *token*: uma palavra em um documento

Vocabulário

- *documento*: informação que será referenciada no índice
  - livro, capítulo, página, parágrafo, frase, página página da web?
  - indexar um livro pode gerar resultados indesejados
  - indexar um livro faz com que o usuário tenha que procurar no livro pela informação
  - indexar frases pode evitar que resultados sejam gerados porque as palavras não ocorreram na mesma frase
- *token*: uma palavra em um documento
- *type/tipo*: classe (normalização) do *token*
  - *venda* é o tipo e *vender*, *vender* são normalizados para o tipo *venda*
  - *stemming* e lematização são exemplos comuns

Vocabulário

- *documento*: informação que será referenciada no índice
  - livro, capítulo, página, parágrafo, frase, página página da web?
  - indexar um livro pode gerar resultados indesejados
  - indexar um livro faz com que o usuário tenha que procurar no livro pela informação
  - indexar frases pode evitar que resultados sejam gerados porque as palavras não ocorreram na mesma frase
- *token*: uma palavra em um documento
- *type/tipo*: classe (normalização) do *token*
  - *venda* é o tipo e *vender*, *vender* são normalizados para o tipo *venda*
  - *stemming* e lematização são exemplos comuns
- *termo*: *token* que foi adicionado ao índice

Tokenização

Quebrar um documento em *sequências* de palavras

```
1 texto = "Only in the darkness can you see the stars."
2 texto.split()
3
4 # ['Only', 'in', 'the', 'darkness', 'can', 'you', 'see
    ', 'the', 'stars.']
5
```

Tokenização

Quebrar um documento em *sequências* de palavras

```
1 import re
2
3
4 texto = "Only in the darkness can you see the stars."
5 re.findall(r'(\b\w+\b)+', texto)
6
7 # ['Only', 'in', 'the', 'darkness', 'can', 'you', 'see
      ', 'the', 'stars']
8
```

Tokenização

Quebrar um documento em *sequências* de palavras

```
1  from nltk.tokenize import word_tokenize
2
3
4  # import nltk
5  # nltk.download('punkt')
6
7
8  texto = "Only in the darkness can you see the stars."
9  word_tokenize(texto)
10
11 # ['Only', 'in', 'the', 'darkness', 'can', 'you', 'see
       ', 'the', 'stars', '.']
12
```

Normalização

Define uma forma padrão para os *tokens*

- *Casing*
- Normalizar plurais
- Normalizar conjugação verbal

Lower case

```
1 texto = "Only in the darkness can you see the stars."
2 tokens = word_tokenize(texto)
3 [t.lower() for t in tokens]
4
5
6 # ['only', 'in', 'the', 'darkness', 'can', 'you', 'see
    ', 'the', 'stars', '.']
7
```

Lower case

```
1 texto = "Only in the darkness can you see the stars."
2 tokens = word_tokenize(texto)
3 [t.lower() for t in tokens]
4
5
6 # ['only', 'in', 'the', 'darkness', 'can', 'you', 'see
    ', 'the', 'stars', '.']
7
```

Em alguns casos, palavras maiúsculas podem ser importantes. Por exemplo, *IT'S YOUR LUCKY DAY. You won ...* é um exemplo de *spam* onde normalmente existem palavras em maiúsculo.

Stemming

Obtém o radical de uma palavra

```
1 from nltk.stem import PorterStemmer
2
3 stemmer = PorterStemmer()
4 [stemmer.stem(t) for t in tokens]
5
6
7 # ['onli', 'in', 'the', 'dark', 'can', 'you', 'see', '
    the', 'star', '.']
8
```

Lemmatisation

Tenta obter a forma no dicionário
Por exemplo: *saw* por ser tanto *see* (ver) ou *saw* (cerra) dependendo do contexto.

Stop words

Remove palavras/pontuação que não tem importância no texto

```
1  from nltk.corpus import stopwords
2
3  # import nltk
4  # nltk.download('stopwords')
5
6  sws = stopwords.words('english')
7
8  tokens = [t.lower() for t in tokens]
9  [t for t in tokens if t not in sws]
10
11
12 # ['darkness', 'see', 'stars', '.']
13
```

Stop words

Cuidado ao remover *stop words*. De forma geral, elas são mantidas no texto.

```
1  from nltk.corpus import stopwords
2
3  # import nltk
4  # nltk.download('stopwords')
5
6  sws = stopwords.words('english')
7
8  texto = "The Who is a rock band"
9  tokens = texto.split()
10 tokens = [t.lower() for t in tokens]
11 [t for t in tokens if t not in sws]
12
13 # ['rock', 'band']
14
```

Bag of words

- Remove *tokens* duplicados
- Documentos são conjuntos de palavras
  - A sequência das palavras não importa mais

```
1 texto = "Only in the darkness can you see the stars."
2 tokens = word_tokenize(texto)
3 tokens = [t.lower() for t in tokens]
4 set(tokens)
5
6 # {'.', 'can', 'darkness', 'in', 'only', 'see', 'stars
    ', 'the', 'you'}
7
```

Matriz termo-documento

Uma matriz $\mathbf{I}$ onde cada linha representa um *termo* e cada coluna um *documento*.

$$\mathbf{I}_{i,j} = \begin{cases} 1, & \text{se o termo } i \text{ aparece no documento } j \\ 0, & \text{caso contrário} \end{cases}$$

Consultas booleanas

- Consultas *AND*, *OR*, *NOT* são operações binárias em vetores

Consultas booleanas

- Consultas *AND*, *OR*, *NOT* são operações binárias em vetores
- Termo 01: $\mathbf{I}_{01,:} = [0, 0, 0, 1, 1, ..., 1, 0, 0, 0]^T$
- Termo 50: $\mathbf{I}_{50,:} = [1, 1, 0, 0, 1, ..., 1, 1, 1, 0]^T$

Consultas booleanas

- Consultas *AND*, *OR*, *NOT* são operações binárias em vetores
- Termo 01: $\mathbf{I}_{01,:} = [0, 0, 0, 1, 1, ..., 1, 0, 0, 0]^T$
- Termo 50: $\mathbf{I}_{50,:} = [1, 1, 0, 0, 1, ..., 1, 1, 1, 0]^T$
- Termo 01 *AND* Termo 50
- $[0, 0, 0, 1, 1, ..., 1, 0, 0, 0]^T$
- *AND*
- $[1, 1, 0, 0, 1, ..., 1, 1, 1, 0]^T$

Consultas booleanas

- Consultas *AND*, *OR*, *NOT* são operações binárias em vetores
- Termo 01: $\mathbf{I}_{01,:} = [0, 0, 0, 1, 1, ..., 1, 0, 0, 0]^T$
- Termo 50: $\mathbf{I}_{50,:} = [1, 1, 0, 0, 1, ..., 1, 1, 1, 0]^T$
- Termo 01 *AND* Termo 50
- $[0, 0, 0, 1, 1, ..., 1, 0, 0, 0]^T$
- *AND*
- $[1, 1, 0, 0, 1, ..., 1, 1, 1, 0]^T$
- Resultado
- $[0, 0, 0, 0, 1, ..., 1, 0, 0, 0]^T$
- As colunas com *1* são os ids dos documentos que contém os termos *Termo 01* e *Termo 50*

Índice invertido

- O problema de usar uma matriz termo-documento é que usaremos muita memória para armazenar muitos *0*s
- O índice invertido é um dicionário de termos para listas ordenadas de documentos

Índice invertido

```
1  documentos = [
2      "The Who is a rock band", # 1
3      "Only in the darkness can you see the stars." # 2
4  ]
5
6  I = {
7      'the': [1, 2],
8      'who': [1],
9      'is': [1],
10     # ...
11     # ...
12     'can': [2],
13     'see': [2],
14     'stars': [2]
15 }
16
```

Operação AND - Intersecção de listas

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

Figure: Intersecção de listas. Fonte: Introduction to Information Retrieval