Encontro Python Sergipe #8

Introdução ao Processamento de Linguagem Natural com Python

Octávio Santana

Formação

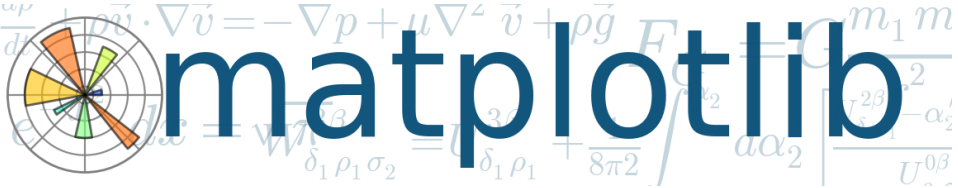
- Bacharel em Física pela Universidade Federal de Sergipe (2012);
- Mestrado em física pela Universidade Estadual de Campinas (2015);
- Doutor em física pela Universidade Estadual de Campinas (2019);

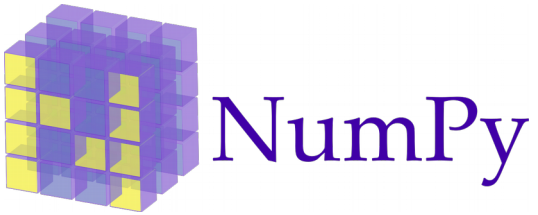
Alguns tópicos de pesquisa

Lasers Caóticos;

Sincronismo de Lasers Caóticos;

Vórtices Óticos;



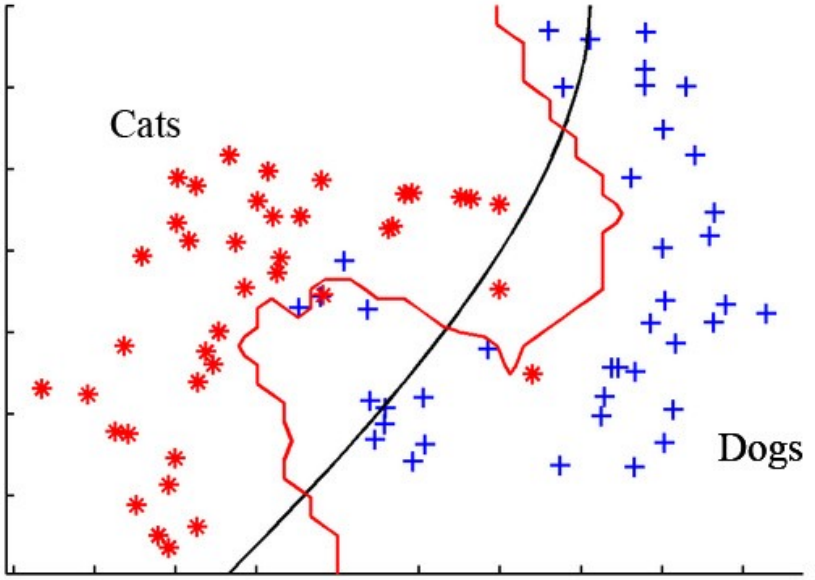


Introdução

Uma das tarefas em **Machine Learning** e **Data Mining** é a capacidade de comparar objetos.

Cluster

Classificação



Introdução

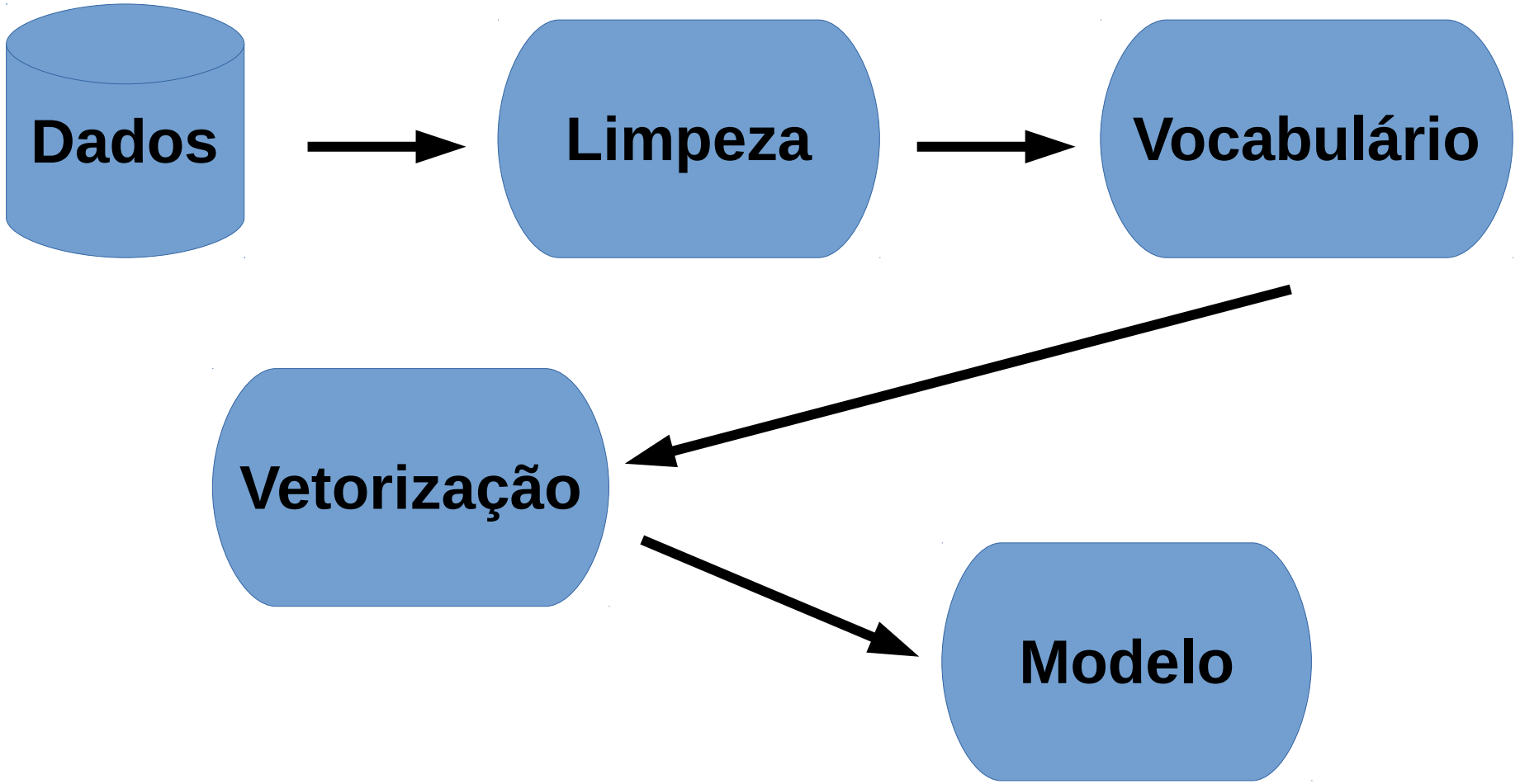
O que é Processamento de Linguagem Natural (NLP)?

A NLP refere-se a tarefas de análise que lidam com linguagem humana natural, na forma de texto ou fala. Essas tarefas geralmente envolvem algum tipo de aprendizado de máquina, seja para classificação de texto ou para geração de recursos, mas a NLP não é apenas aprendizado de máquina. Tarefas como o pré-processamento de texto e a limpeza também se enquadram em NLP.

Introdução

O quanto similar são duas frases?

Fluxograma



Dados

```
import re
import string
from math import sqrt
```

```
corpus = [
    'O importante não é vencer todos os dias, mas lutar sempre.',
    'Maior que a tristeza de não haver vencido é a vergonha de não ter lutado!',
    'É melhor conquistar a si mesmo do que vencer mil batalhas.',
    'Oi eu sou o goku!',
    'Oi eu não sou o goku!!!',
    'Oi seu nome é goku(?)'
]
```

Limpeza nos dados de texto

```python
def clear_text(text):
    """
    Limpar os textos, eliminando pontuações e converter
    todo o texto com letras minusculas.
    """
    pattern = "[{}]".format(string.punctuation)
    text = [word.lower() for word in text]
    text = [[re.sub(pattern, "", word) for word in words.split()] for words in text]
    text = [[word for word in words if len(word)>1] for words in text]
    text = [' '.join(words) for words in text]
    return text
```

```
pattern
'[!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~]'
```

```
text
'Oi eu sou o goku!!!'

text.lower()
'oi eu sou o goku!!!'
```

Limpeza nos dados de texto

```python
def clear_text(text):
    """
    Limpar os textos, eliminando pontuações e converter
    todo o texto com letras minusculas.
    """
    pattern = "[{}]".format(string.punctuation)
    text = [word.lower() for word in text]
    text = [[re.sub(pattern, "", word) for word in words.split()] for words in text]
    text = [[word for word in words if len(word)>1] for words in text]
    text = [' '.join(words) for words in text]
    return text
```

```
text.split()
['oi', 'eu', 'sou', 'o', 'goku!!!']

[re.sub(pattern, "", word) for word in text.split()]
['oi', 'eu', 'sou', 'o', 'goku']
```

Limpeza nos dados de texto

```
corpus_clear = clear_text(corpus)

print('-- Antes --')
[print(frase) for frase in corpus]
print()
print('-- Depois --')
[print(frase) for frase in corpus_clear]
```

```
-- Antes --
O importante não é vencer todos os dias, mas lutar sempre.
Maior que a tristeza de não haver vencido é a vergonha de não ter lutado!
É melhor conquistar a si mesmo do que vencer mil batalhas.
Oi eu sou o goku!
Oi eu não sou o goku!!!
Oi seu nome é goku(?)

-- Depois --
importante não vencer todos os dias mas lutar sempre
maior que tristeza de não haver vencido vergonha de não ter lutado
melhor conquistar si mesmo do que vencer mil batalhas
oi eu sou goku
oi eu não sou goku
oi seu nome goku
```

Construindo o vocabulário

```
def text_all(text):
    """
    Armazena em um vetor todas as palavras dos textos sem repetições.
    """
    text_set = set()
    for w in [words.split() for words in text]:
        text_set.update(w)
    return list(text_set)
```

```
vocabulary = text_all(corpus_clear)
print(vocabulary)
```

```
['todos', 'não', 'melhor', 'tristeza', 'os', 'oi', 'mesmo', 'vencer', 'ter', 'seu', 'sou',
'si', 'vencido', 'maior', 'batalhas', 'sempre', 'que', 'lutado', 'eu', 'haver', 'mil', 'con
quistar', 'vergonha', 'lutar', 'do', 'goku', 'importante', 'de', 'dias', 'mas', 'nome']
```

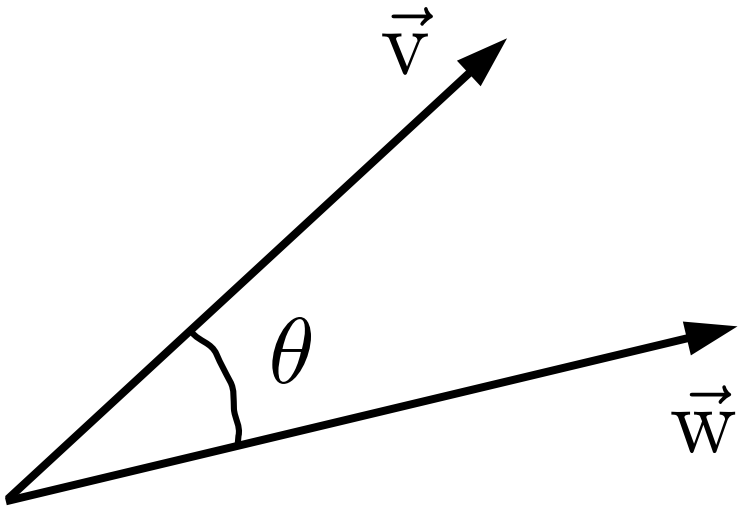
Vetorização booleana

```
def fit_transform(text, words=vocabulary):
    """
    Converte o texto em um vetor, onde compara se cada
    palavra obtida no vetor de
    todas as palavras contém ou não em cada texto.
    Insere 1 se sim e 0 se não.
    """

    return [int(word in text.split()) for word in words]
```

```
features = list(map(fit_transform, corpus_clear))

for feature in features:
    print(feature)
```

```
[0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1]
[0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Modelo - Produto escalar



$$\vec{v} \cdot \vec{w} = |v||w| \cos\theta$$

$$\vec{v} \cdot \vec{w} = \sum_{i=1}^{n} v_i w_i$$

```
def dot(v, w):
    return sum([vi*wi for vi, wi in zip(v,w)])

def cosine_similarity(v, w):
    return dot(v, w)/sqrt(dot(v, v)*dot(w, w))
```

Similaridade nas frases

```
def text_simillarities(id_text, features=features,
                       text=corpus, n_text=3):
    """
    Dado o texto a ser analisado, a função retorna em
    ordem descrecente quais os demais textos são
    similares ao analisado. A função retorna matriz
    de 2 por n_text, onde a primeira e a segunda coluna
    refere-se ao texto analisado e a similaridade do
    texto analisado, respectivamente.
    """
    simillarity = [[cosine_similarity(features[id_text], feature), int(i)] \
                   for i, feature in enumerate(features)]

    simillarity = list(sorted(simillarity,
                              key=lambda sim: sim[0],
                              reverse=True))

    return [[text[indice], sim] for sim, indice in simillarity[1:]][:n_text]
```

Similaridade nas frases

```
simillarity
[[0.2, 2], [0.75, 0], [0.7, 1], [0.9, 3]]

list(sorted(simillarity, key=lambda sim: sim[0], reverse=True))
[[0.9, 3], [0.75, 0], [0.7, 1], [0.2, 2]]
```

```
def text_simillarities(id_text, features=features,
                       text=corpus, n_text=3):
    """
    simillarity = [[cosine_similarity(features[id_text], feature), int(i)] \
                   for i, feature in enumerate(features)]

    simillarity = list(sorted(simillarity,
                              key=lambda sim: sim[0],
                              reverse=True))

    return [[text[indice], sim] for sim, indice in simillarity[1:]][:n_text]
```

Similaridade nas frases

```
print('Texto analisado -> ',corpus[3], '\n')
for t, s in text_simillarities(3):
    print('Texto: {} | Similaridade: {}'.format(t, round(s, 2)))
```

```
Texto analisado ->  Oi eu sou o goku!

Texto: Oi eu não sou o goku!!! | Similaridade: 0.89
Texto: Oi seu nome é goku(?) | Similaridade: 0.5
Texto: O importante não é vencer todos os dias, mas lutar sempre. | Similaridade: 0.0
```

Conclusões

Vimos como construir um algoritmo que analisa o quanto similar são duas frases utilizando apenas o conhecimento de vetores e as bibliotecas nativas do python.

Algumas lib’s para NLP

- NLTK (pip install nltk) → https://www.nltk.org/
- Scikit learning (pip install scikit-learn) → https://scikit-learn.org/stable/
- TextBlob (pip install textblob) → https://textblob.readthedocs.io/en/dev/