Processamento de Linguagem Natural

Prof. Henrique Batista da Silva

PUC Minas Virtual

Introdução

Introdução

- Análise de sentimentos com algoritmo de VADER
- Utilizando TF-IDF para cálculo de similaridade (base para criação de um chatbot)

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

Análise de sentimentos

Análise de sentimentos

- Nós podemos usar os tokens extraídos das palavras, n-grams e as técnicas que vimos até o momento para construir um sistema de análise de sentimentos (uma das principais aplicações em NLP)

- A análise de sentimentos permite saber, por exemplo, o que as pessoas pensam sobre determinada entidade (produto, musica, game, filme, etc.)

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

Análise de sentimentos

- A maioria dos produtos são classificados por estrelas (e-commerce por exemplo)
- No entanto, uma forma mais natural é usar os comentários (muitas das plataformas de e-commerce possuem) em linguagem natural.
- O grande problema no uso de comentários para saber o feedback de um produto é que será necessário ler os comentários para uma análise.

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

Análise de sentimentos

- Um pipeline de NLP pode processar uma grande quantidade de feedback do usuário de maneira rápida e objetiva, eliminando um possível viés que a análise humana possa ter.

- E um pipeline de NLP pode gerar uma classificação numérica sobre o comentário (negativo ou positivo).

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

Análise de sentimentos

- Outra aplicação é a análise de mensagem indesejadas e inadequadas para determinado contexto.

- Uma chatbot, por exemplo, pode ser capaz de medir o sentimento nas mensagens de chat processadas para responder adequadamente (ou analisar suas próprias mensagens antes de responder).

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

Análise de sentimentos

- Suponha que o objetivo seja medir o quão positivo um texto pode ser (texto sobre um determinado produto).
- Este algoritmo pode produzir um número de ponto flutuante entre +1 e -1 (+1 para texto com sentimentos positivos como "Absolutamente perfeito! Adoro! :-) :-) :-). ”
- E -1 para texto com sentimentos negativos como “Horrível! Completamente inútil. :(. “

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

Análise de sentimentos

- Existem duas abordagens para análise de sentimentos:
  - Um algoritmo de regras compostas por um ser humano
  - Um modelo de machine learning treinado com dados produzidos por computador

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

Análise de sentimentos

- A primeira abordagem usa regras criadas por seres humanos para medir sentimentos.
- Uma abordagem comum é encontrar palavras-chave no texto e mapear cada uma para pontuações ou pesos numéricos em um dicionário, por exemplo.
- Neste caso, usamos tokenização, stems, lemas ou n-gram no dicionário, em vez de apenas palavras.

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

Análise de sentimentos

- A “regra” do algoritmo seria somar essas pontuações para cada palavra-chave em um documento encontradas no dicionário de pontuações.

- Podemos usar o algoritmo VADER (sklearn) para produzir este dicionário de pontuações.

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

VADER

Análise de sentimentos baseado em regras

- Criado por Hutto e Gilbert na GA Tech, o VADER (Valence Aware Dictionary for sEntiment Reasoning) é um dos primeiros algoritmos baseados em regras.
- Muitos dos pacotes de NLP (NLTK) implementam este algoritmo.
- Para instalação em Python:

```
pip install vaderSentiment
```

VADER: http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf

Veja mais sobre o pacote vaderSentiment: https://github.com/cjhutto/vaderSentiment

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

VADER

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# procedimento que exibe a pontuação do dicionário de palavras
sa = SentimentIntensityAnalyzer()
print(sa.lexicon)
```

Contém o dicionário de tokens com a pontuação

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

VADER

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# procedimento que exibe a pontuação do dicionário de palavras
sa = SentimentIntensityAnalyzer()
print(sa.lexicon)

score = [(tok, score) for tok, score in sa.lexicon.items() if " " in tok]
print(score)
```

Observe as palavras com espaço no dicionário, são poucas.

```
[("( '}{' )", 1.6), ("can't stand", -2.0), ('fed up', -1.8), ('screwed up', -1.5)]
```


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

VADER

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# procedimento que exibe a pontuação do dicionário de palavras
sa = SentimentIntensityAnalyzer()
# print(sa.lexicon)

# procedimento que exibe a pontuação de palavras com espaço " "
score = [(tok, score) for tok, score in sa.lexicon.items() if " " in tok]
# print(score)

print(sa.polarity_scores(text="Python is very readable and it's great for NLP."))
```

```
{'neg': 0.0, 'neu': 0.661, 'pos': 0.339, 'compound': 0.6249}
```

O algoritmo retorna três pontuações diferentes (negativo, positivo e neutro) e então um resultado total combinado

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

VADER

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# procedimento que exibe a pontuação do dicionário de palavras
sa = SentimentIntensityAnalyzer()
# print(sa.lexicon)

# procedimento que exibe a pontuação de palavras com espaço " "
score = [(tok, score) for tok, score in sa.lexicon.items() if " " in tok]
# print(score)

print(sa.polarity_scores(text="Python is very readable and it's great for NLP."))

print(sa.polarity_scores(text="Python is not a bad choice for most applications."))
```

```
{'neg': 0.0, 'neu': 0.737, 'pos': 0.263, 'compound': 0.431}
```

O algoritmo retorna três pontuações diferentes (negativo, positivo e neutro) e então um resultado total combinado

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

VADER

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# procedimento que exibe a pontuação do dicionário de palavras
sa = SentimentIntensityAnalyzer()
# print(sa.lexicon)

corpus = ["Absolutely perfect! Love it! :-) :-) :-
)", "Horrible! Completely useless. :(", "It was OK. Some good and some bad things."]

for doc in corpus:
    scores = sa.polarity_scores(doc)
    print('{:+}: {}'.format(scores['compound'], doc))
```

```
+0.9428: Absolutely perfect! Love it! :-) :-) :-)
-0.8768: Horrible! Completely useless. :(
-0.1531: It was OK. Some good and some bad things.
```

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

Análise de sentimentos baseado em regras

- Uma desvantagem do VADER é não analisar todas as palavras de um documento, apenas cerca de 7.500.
- Seria possível criar o próprio dicionário, no entanto, seria muito difícil sem entender todo o idioma (praticamente impossível)

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

Ranking

PUC Minas Virtual

Representando o tópico

- Conforme estudado anteriormente sobre TF-IDF, é possível comparar facilmente dois vetores e obter a similaridade entre eles.
- Portanto, em cada vetor que representa o documento no espaço multidimensional, vamos calcular o TF-IDF para cada termo.
- Agora, os vetores refletirão mais detalhadamente o significado, ou tópico sobre o que se trata o documento

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

```
# calculando os tokens para cada um dos documentos
from nltk.tokenize import TreebankWordTokenizer
from collections import Counter
import copy

tokenizer = TreebankWordTokenizer()

docs = ["The faster Harry got to the store, the faster and faster Harry would get home."]
docs.append("Harry is hairy and faster than Jill.")
docs.append("Jill is not as hairy as Harry.")

doc_tokens = []
for doc in docs:
    doc_tokens += [sorted(tokenizer.tokenize(doc.lower()))]

all_doc_tokens = sum(doc_tokens, [])
lexicon = sorted(set(all_doc_tokens))
```


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

(continua do Código do slide anterior)

```
# calculando a distância dos cossenos para os vetores de cada documento
import math

def cosine_sim(vec1, vec2):
    vec1 = [val for val in vec1.values()]
    vec2 = [val for val in vec2.values()]

    dot_prod = 0
    for i, v in enumerate(vec1):
        dot_prod += v * vec2[i]

    mag_1 = math.sqrt(sum([x**2 for x in vec1]))
    mag_2 = math.sqrt(sum([x**2 for x in vec2]))

    return dot_prod / (mag_1 * mag_2)
```


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

```
# calculando td-idf de cada termo nos três documentos
from collections import OrderedDict
zero_vector = OrderedDict((token, 0) for token in lexicon)

document_tfidf_vectors = []
for doc in docs:
    vec = copy.copy(zero_vector)
    tokens = tokenizer.tokenize(doc.lower())
    token_counts = Counter(tokens)

    for key, value in token_counts.items():
        docs_containing_key = 0
        for _doc in docs:
            if key in _doc:
                docs_containing_key += 1
        tf = value / len(lexicon)
        if docs_containing_key:
            idf = len(docs) / docs_containing_key
        else:
            idf = 0
        vec[key] = tf * idf
    document_tfidf_vectors.append(vec)

print(document_tfidf_vectors)
```

(continua do Código do slide anterior)

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

Representando o tópico

- Assim, temos uma representação vetorial K-dimensional (K é o número de palavras do vocabulário) de cada documento no corpus.
- Podemos dizer que dois vetores, em um determinado espaço vetorial, são similares se tiverem um ângulo semelhante.
- Considerando cada vetor começando na origem (0, ..., 0), os que alcançam o mesmo ângulo são similares, mesmo que não alcancem a mesma distância (comprimento do vetor no espaço Euclideano).

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

Representando o tópico

- Dois vetores são considerados similares se a similaridade do cosseno for alta; portanto, você pode encontrar a similaridade entre dois vetores se estes minimizarem:

$$\cos \Theta = \frac{A \cdot B}{|A|\ |B|}$$

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

Representando o tópico

- Para calcular a similaridade entre dois documentos, podemos tratar a consulta em si como um documento e, portanto, obter a representação vetorial baseada em TF-IDF.
- A última etapa é encontrar os documentos cujos vetores têm os maiores valores de similaridades da distância de cosseno com a consulta e retorná-los como resultados da pesquisa.

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

```
# realizando cálculo por similaridade do coseno
query = "How long does it take to get to the store?"
query_vec = copy.copy(zero_vector)
query_vec = copy.copy(zero_vector)

tokens = tokenizer.tokenize(query.lower())
token_counts = Counter(tokens)

for key, value in token_counts.items():
    docs_containing_key = 0
    for _doc in docs:
        if key in _doc.lower():
            docs_containing_key += 1
        if docs_containing_key == 0:
             continue
        tf = value / len(tokens)
        idf = len(docs) / docs_containing_key
        query_vec[key] = tf * idf

print(cosine_sim(query_vec, document_tfidf_vectors[0]))
print(cosine_sim(query_vec, document_tfidf_vectors[1]))
print(cosine_sim(query_vec, document_tfidf_vectors[2]))
```

(continua do Código do slide anterior)

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

Representando o tópico

- O doc 0 possui a maior relevância para a consulta. E assim, é possível pode encontrar documentos relevantes em qualquer corpus, sejam artigos na Wikipedia ou tweets.

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

Representando o tópico

- Observe que o mecanismo de busca do Google funciona de forma diferente, baseado em índices dos vetores TF-IDF (não é nosso objetivo aqui).
- A maioria dos mecanismos de pesquisa pode responder em tempo constante (O (1)) porque eles usam um índice invertido.
- Para saber mais sobre índices:

https://pypi.org/project/Whoosh/

https://github.com/Mplsbeb/whoosh

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

Representando o tópico

- Esta aplicação de busca por similaridade é uma etapa importante de um pipeline para NLP.
- Alguns chatbots dependem exclusivamente de um mecanismo de busca como seu único algoritmo para gerar respostas.
- Você precisa executar uma etapa adicional para transformar seu índice de pesquisa simples (TF-IDF) em um chatbot.

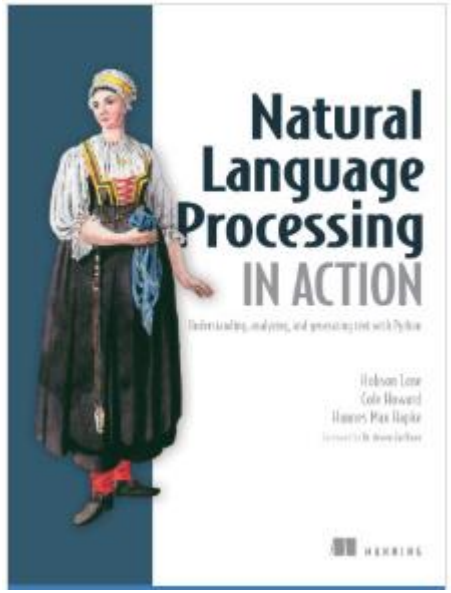
Ref.: Hobson Lane, Cole Howard, Hannes Hapke. **Natural Language Processing in Action** 2019

Crie o seu próprio chatbot

- Uma possibilidade para o chatbot, é necessário armazenar dados de treinamento em pares de perguntas e respostas apropriadas.
- Em seguida, calculamos o TF-IDF para pesquisar uma pergunta mais semelhante com o texto de entrada do usuário.
- Em vez de retornar a instrução mais similar de um banco de dados, retornamos a resposta associada a esta 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