Processamento de Linguagem Natural

Prof. Henrique Batista da Silva

PUC Minas Virtual

Agenda

- Bag of words
- Vetorização
- Espaço vetorial

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

Introdução

Introdução

- A detecção de palavras no contexto de NLP é útil para tarefas como obter estatísticas sobre o uso de palavras ou fazer pesquisas de palavras-chave
- Isso fará com que um detector de spam seja menos propenso a se enganar com uma única palavra.

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

Introdução

- Ou até mesmo poder avaliar o quão positivo um tweet é quando há uma ampla variedade de palavras com vários graus de pontuação de "positividade“
- A frequência com que essas palavras aparecem em um documento em relação ao restante dos documentos pode ser usado para refinar ainda mais a "positividade" do documento

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

Introdução

- A ideia aqui é estudar medidas mais diferenciadas e seu uso em um documento.
- A abordagem que analisaremos aqui tem sido a base para a geração de recursos da linguagem natural para mecanismos de busca comerciais e filtros de spam por décadas.

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

Introdução

- As técnicas de tokenização transformam palavras em números inteiros por representar a ocorrência de cada uma em documentos (vetores binários)
- A ideia agora é representar as palavras em um espaço contínuo.

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

Introdução

- Iremos analisar três técnicas de representar palavras e sua importância em um documento.
  - Bags of words: vetores de frequências de palavras
  - Bags of n-grams: Contagem de pares (n) de palavras
  - TF-IDF vectors: pontuação da palavra que melhor representa sua importância

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

Bag of words

Introdução

- Assumimos que, quanto mais vezes uma palavra ocorre no documento, maior deve ser o significado da palavra para o documento.
- Vejamos um exemplo em que a contagem de ocorrências de palavras é útil:

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

Bag of words

```
from nltk.tokenize import TreebankWordTokenizer
from collections import Counter

sentence = """The faster Harry got to the store, the faster Harry, the faster, would get home."""

tokenizer = TreebankWordTokenizer()
tokens = tokenizer.tokenize(sentence.lower())

print(tokens)

bag_of_words = Counter(tokens)

print(bag_of_words)
```

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

Introdução

- Para documentos pequenos como este, a lista não ordenada de palavras pode conter muitas informações sobre a intenção original da frase.
- E estas informações são suficientes para por exemplo, detectar spam, calcular sentimentos e até detectar sarcasmo.

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

Introdução

- Este número de vezes em que uma palavra aparece em um documento é chamado (em inglês) de term frequency (TF).
- Se normalizado, ela será dividido pelo número de termos no documento (que será no máximo 1, se todas as palavras do documento forem iguais)

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

Introdução

- A normalização é importante porque, dependendo do tamanho do documento, uma palavra aparecer 100 vezes pode indicar muita relevância para um documento de 1000 (0.1) palavras ou indicar baixa relevância para um documento de 1000000 (0.001).
- Assim, para cada palavra podemos calcular a importância relativa do documento neste termo.

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

Bag of words

```
from nltk.tokenize import TreebankWordTokenizer
from collections import Counter

sentence = """The faster Harry got to the store, the faster Harry, the faster, would get home."""

tokenizer = TreebankWordTokenizer()
tokens = tokenizer.tokenize(sentence.lower())
print(tokens)

bag_of_words = Counter(tokens)
print(bag_of_words)

bag_of_words_most_common = bag_of_words.most_common(4)
print(bag_of_words_most_common)
```

Vamos alterar o programa para incluir a contagem das palavras mais comuns

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

Introdução

- Assim, obtemos os 4 principais termos daquele documento.
- Sendo que termos como “the” e pontuações não são muito úteis para o documento e serão descartados (stop words).
- Agora vamos calcular o TF da palavra “harry”.

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

Bag of words

```
from nltk.tokenize import TreebankWordTokenizer
from collections import Counter

sentence = """The faster Harry got to the store, the faster Harry, the faster, would get home."""

tokenizer = TreebankWordTokenizer()
tokens = tokenizer.tokenize(sentence.lower())
print(tokens)

bag_of_words = Counter(tokens)
print(bag_of_words)

bag_of_words_most_common = bag_of_words.most_common(4)
print(bag_of_words_most_common)

times_harry_appears = bag_of_words['harry']
num_unique_words = len(bag_of_words)
tf = times_harry_appears / num_unique_words
print(round(tf, 4))
```

Calculando o TF da palavra “harry”

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

Introdução

- Vamos utilizar um texto maior da wikipedia sobre pipas (kite_text da biblioteca “nlpia”)

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

Bag of words

```
from collections import Counter
from nltk.tokenize import TreebankWordTokenizer
from nlpia.data.loaders import kite_text

tokenizer = TreebankWordTokenizer()

tokens = tokenizer.tokenize(kite_text.lower())
token_counts = Counter(tokens)
print(token_counts)
```


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

Introdução

- Observe que há muitos stop words neste texto.
- Iremos então eliminar estes stop words.

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

Bag of words

```
from nltk.tokenize import TreebankWordTokenizer
from nlpia.data.loaders import kite_text
import nltk
from collections import Counter

tokenizer = TreebankWordTokenizer()

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

nltk.download('stopwords', quiet=True)
stopwords = nltk.corpus.stopwords.words('english')

tokens = [x for x in tokens if x not in stopwords]
kite_counts = Counter(tokens)

print(kite_counts)
```

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

Introdução

- Observe agora que os termos, “kite(s)”, “wing” e “lift” são todos importantes.
- E até mesmo para nós, fica fácil saber do que o documento se trata.

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

Vetorização

PUC Minas Virtual

Vetorização

- Agora, ao invés de manter a descrição do documento com um dicionário de frequência, vamos produzir um vetor dessa contagem de palavras.

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

Vetorização

```
from nltk.tokenize import TreebankWordTokenizer
from nlpia.data.loaders import kite_text
import nltk
from collections import Counter

tokenizer = TreebankWordTokenizer()

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

nltk.download('stopwords', quiet=True)
stopwords = nltk.corpus.stopwords.words('english')

tokens = [x for x in tokens if x not in stopwords]
kite_counts = Counter(tokens)

document_vector = []
doc_length = len(tokens)
for key, value in kite_counts.most_common():
    document_vector.append(value / doc_length)

print(document_vector)
```

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

Vetorização

- Cada dimensão deste vetor é o TF normalizado de cada palavra neste (específico) documento.
- Mas observe que, como estamos tratando de apenas um único documento, todas as posições correspondem às palavras que aparecem no documento.
- Normalmente, com vários documentos, como que nosso vetor se comportaria?

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

Vetorização

- Para comparar vários documentos, precisamos que todos tenham vetores do mesmo tamanho (dimensão), sendo que o valor desta dimensão é a quantidade total de palavras do vocabulário.
- Vamos a um exemplo com mais de um documento.

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

Vetorização

```
from nltk.tokenize import TreebankWordTokenizer
from collections import Counter

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()))]
print(len(doc_tokens[0]))

all_doc_tokens = sum(doc_tokens, [])
print(len(all_doc_tokens))

lexicon = sorted(set(all_doc_tokens))
print(len(lexicon))

print(lexicon)
```

Nome da coleção de palavras no vocabulário (léxico)

Número total de tokens (sem repetição)

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

Vetorização

- Agora veja que cada um dos três vetores de documentos precisará ter 18 posições, mesmo que um documento para esse vetor não contenha todas as 18 palavras no seu vocabulário (léxico).
- Cada token recebe uma posição nos vetores, correspondente à sua posição no seu léxico. Assim, algumas dessas posições de token no vetor terão valor zeros.

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

Vetorização

```
# calculando o vetor de mesma dimensão para cada document
from collections import OrderedDict
zero_vector = OrderedDict((token, 0) for token in lexicon)
print(zero_vector)

import copy
doc_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():
        vec[key] = value / len(lexicon)
    doc_vectors.append(vec)

print (len(doc_vectors))
print (doc_vectors[0])
```

(continua do Código do slide anterior)

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

Vetorização

- Observe os três vetores, um para cada documento.
- Todos eles possuem o mesmo número de dimensões. Além disso, observe que em várias destas dimensões os valores para alguns vetores é zero. Ou seja, a palavra correspondente à aquela dimensão não aparece naquele vetor.

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

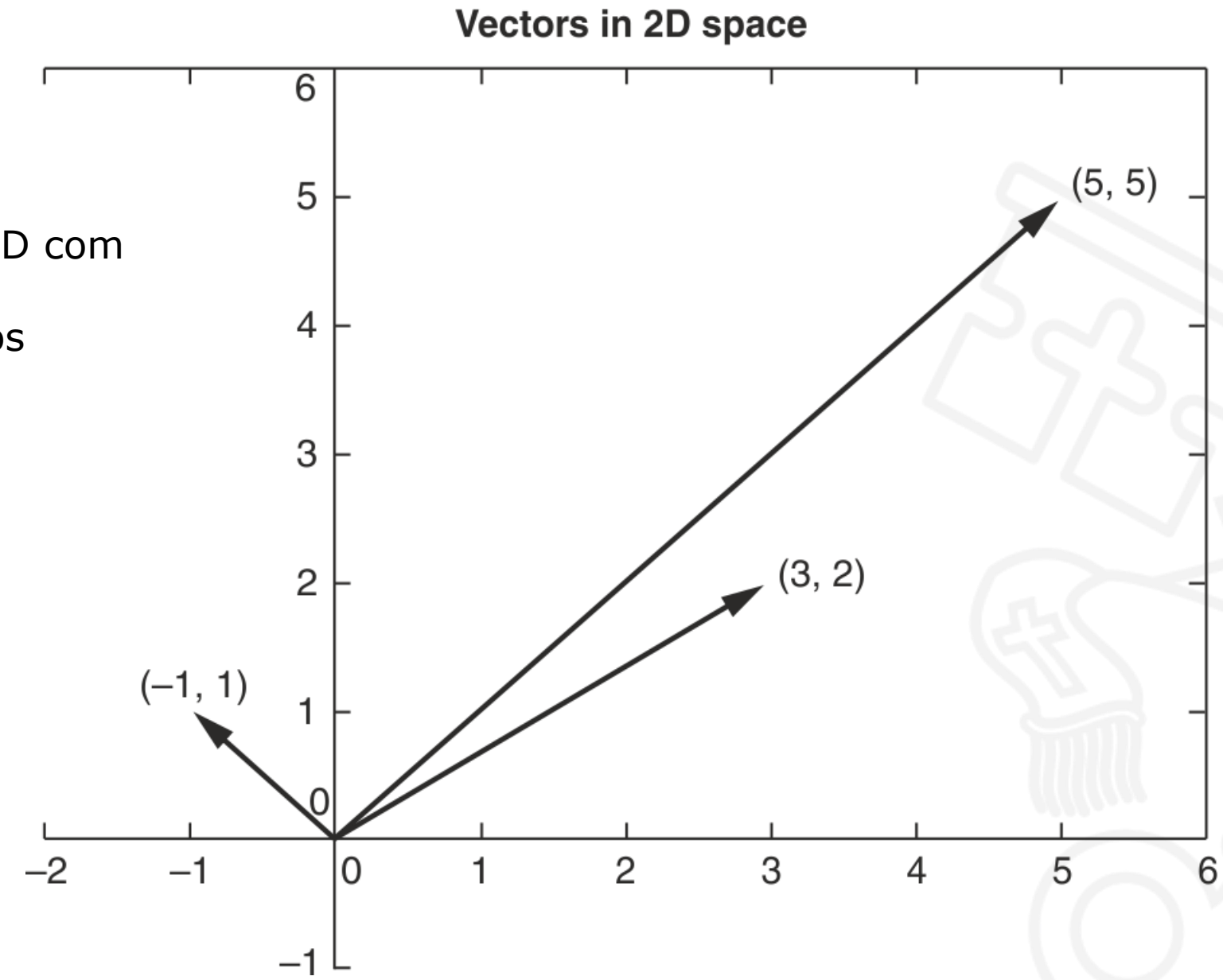
Espaço vetorial

Espaço vetorial

- Estes vetores que calculamos para cada documentos são coordenadas em um espaço vetorial. Eles descrevem um local ou posição nesse espaço.
- Um espaço é a coleção de todos os vetores (documentos) possíveis que podem aparecer nesse espaço.
- Portanto, um vetor com dois valores estaria em um espaço vetorial 2D, um vetor com três no espaço vetorial 3D e assim por diante.

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

Um espaço 2D com três vetores representados



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

Espaço vetorial

- Mas podemos representar mais do que 2 ou 3 dimensões em um espaço vetorial (apesar de não ser possível representar graficamente)
- O número de dimensões dos nossos vetores será o número de palavras no léxico (vocabulário, definido pela letra k), ou seja, podemos ter milhares de dimensões.

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

A maldição da dimensionalidade

- Um dos problemas em representar vetores com grandes dimensões é a necessidade de maior poder computacional para as operações sobre estes vetores.
- Além disso, a maldição da dimensionalidade é um problema, pois em espaços de muitas dimensões, todos os vetores tendem a ficar equidistantes. Isto dificulta muito o cálculo de distância (Euclidean distance) entre eles.

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

Espaço vetorial

- Então, agora que temos os vetores para cada documentos, podemos medir a distância entre eles.
- No espaço vetorial, dois vetores são similares se compartilham direções similares (aproximadamente o mesmo comprimento)
- Nosso objetivo é encontrar documentos que usam as mesmas palavras em uma proporção semelhante

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

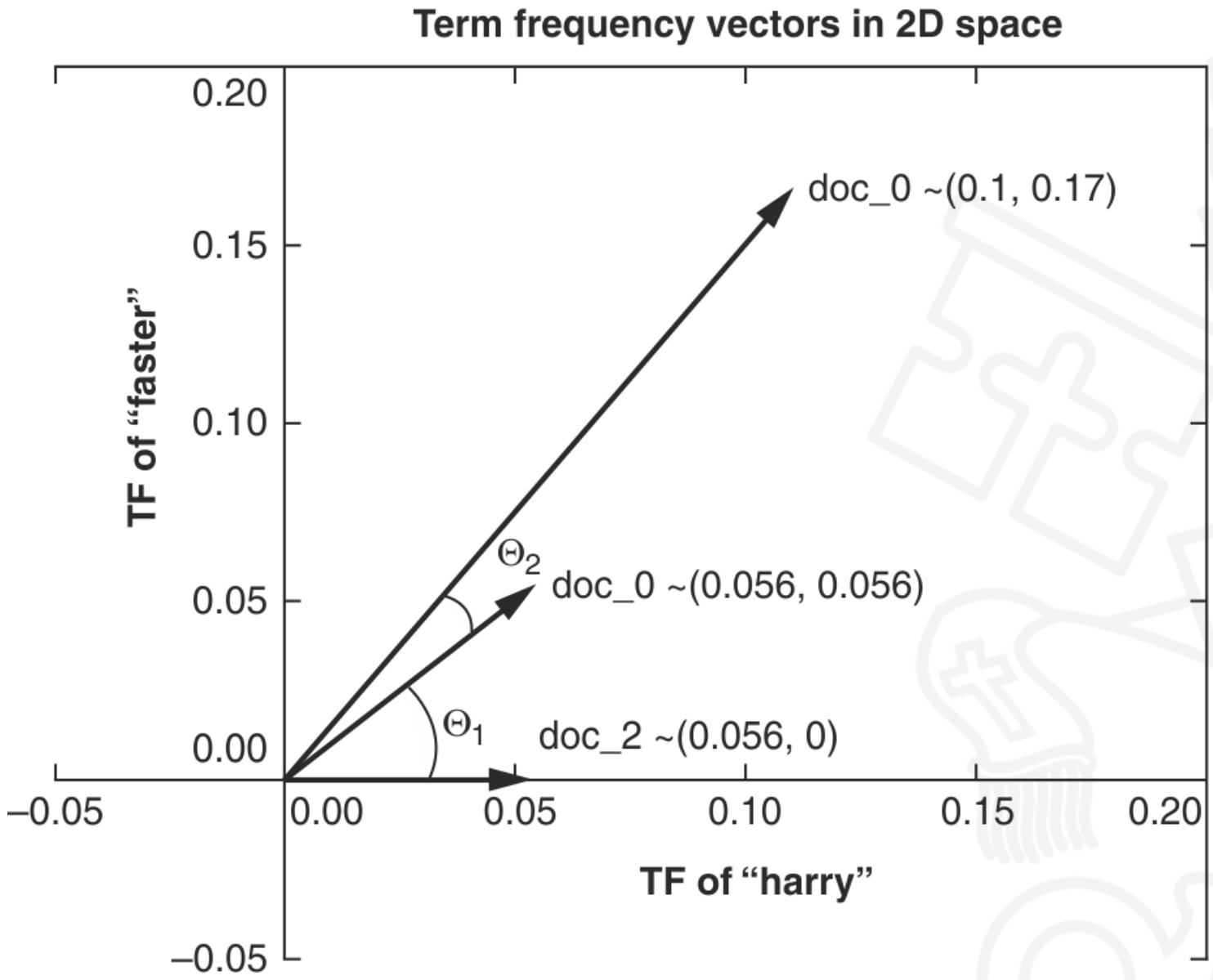
Espaço vetorial

- Uma das métricas que podemos utilizar é a distância do cosseno. Ou seja, é o cosseno do ângulo entre dois vetores (valores de -1 a +1).

$$A \cdot B = |A|\ |B| * \cos \Theta$$

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

Representação 2D da distância do cosseno entre três vetores



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

Vetorização

```
# 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)

d = cosine_sim(doc_vectors[0], doc_vectors[1])
print(d)
```

(continua do Código do slide anterior)

Convertendo os vetores em listas

Cálculo do produto escalar (multiplicação de cada elemento dos veores em pares)

Cálculo da norma (magnitude) de cada vetor. A normal é a dist. Euclideana (raiz quadrada da soma dos quadrados de seus elementos) do eixo zero até o seu ponto na cordenada.

Como a saída da função cosseno, terá um valor entre -1 e +1

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

Espaço vetorial

- O valor retornado pela função é o cosseno do ângulo entre os dois vetores.
- Quanto mais próximo o valor de similaridade do cosseno for 1, mais próximos os dois vetores estão do ângulo.
- Para vetores de documentos de linguagem natural que têm uma similaridade do cosseno próxima a 1, significa que os documentos estão usando palavras semelhantes em proporção semelhante.

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

Espaço vetorial

- Portanto, os documentos cujos vetores de documentos estão próximos um do outro, provavelmente, estão abordando o mesmo assunto.

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

TF-IDF

TF-IDF

- A contagem de palavras, apesar de útil, ainda não informa muito sobre a importância dessa palavra naquele documento em relação ao restante dos documentos do corpus (coleção de documentos)
- Vamos analisar o IDF (Inverse Document Frequency). Para calcular esta medida, vamos contar tokens e agrupá-los de duas maneiras:
  - por documento e em todo o corpus.

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

Introdução

- Vamos voltar a utilizar o texto da wikipedia sobre pipas (kite_text da biblioteca “nlpia”), mas agora pegando outra seção (histórico) como o segundo documento no corpus.

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

TF-IDF

```
from nlpia.data.loaders import kite_text, kite_history
from nltk.tokenize import TreebankWordTokenizer

tokenizer = TreebankWordTokenizer()

kite_intro = kite_text.lower()
intro_tokens = tokenizer.tokenize(kite_intro)
kite_history = kite_history.lower()
history_tokens = tokenizer.tokenize(kite_history)
intro_total = len(intro_tokens)

print(intro_total)

history_total = len(history_tokens)
print(history_total)
```


Tokens de kite-text

Tokens de history

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

TF-IDF

- Agora, com alguns documentos tokenizados, vejamos TF da palavra “kite" em cada documento.

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

TF-IDF

(continua do Código do slide anterior)

```
# calculando o TF de "kite" em cada documento
from collections import Counter

intro_tf = {}
history_tf = {}

intro_counts = Counter(intro_tokens)
intro_tf['kite'] = intro_counts['kite'] / intro_total

history_counts = Counter(history_tokens)
history_tf['kite'] = history_counts['kite'] / history_total

print('Term Frequency of "kite" in intro is: {:.4f}'.format(intro_tf['kite']))
print('Term Frequency of "kite" in history is: {:.4f}'.format(history_tf['kite']))
```

```
Term Frequency of "kite" in intro is: 0.0441
Term Frequency of "kite" in history is: 0.0202
```

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

TF-IDF

- Ao analisar o resultado vimos um número duas vezes maior do que o outro.
- A palavra “kite” é duas vezes mais relevante na seção de “intro”? Não é.
- Então, vamos ver como esses números se relacionam com alguma outra palavra, tipo “and":

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

TF-IDF

(continua do Código do slide anterior)

```
# calculando o TF da palavra "and"
intro_tf['and'] = intro_counts['and'] / intro_total
history_tf['and'] = history_counts['and'] / history_total
print('Term Frequency of "and" in intro is: {:.4f}'.format(intro_tf['and']))

print('Term Frequency of "and" in history is: {:.4f}'.format(history_tf['and']))
```

Term Frequency of "and" in intro is: 0.0275
Term Frequency of "and" in history is: 0.0303

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

TF-IDF

- Ao analisar o resultado, veja que não ajuda muito saber sobre o que o documento se trata, pois “and” parece ser altamente relevante.
- Ideia da frequência inversa (IDF): Quão estranho é esse token neste documento? Se um termo aparece em um documento várias vezes, mas ocorre raramente no restante do corpus, pode-se supor que seja importante para este documento especificamente.

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

TF-IDF

- O IDF de um termo é proporção do: (n) número total de documentos para o (m) número de documentos em que o termo aparece (n/m).
- No caso de “and" e “kite" no exemplo atual, a resposta é a mesma para ambos:

2 (documentos no total) / 2 (documentos que contêm “and“) = 2/2 = 1
2 (documentos no total) / 2 (documentos contêm “kite“) = 2/2 = 1

Não ajuda muito. Então, vamos olhar para outra palavra "China".
2 (documentos no total) / 1 (documento contém "China“) = 2/1 = 2

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

TF-IDF

(continua do Código do slide anterior)

```
# calculando o número de documentos em que cada um dos três termos aparecem
num_docs_containing_and = 0
for doc in [intro_tokens, history_tokens]:
    if 'and' in doc:
        num_docs_containing_and += 1

num_docs_containing_kite = 0
for doc in [intro_tokens, history_tokens]:
    if 'kite' in doc:
        num_docs_containing_kite += 1

num_docs_containing_china = 0
for doc in [intro_tokens, history_tokens]:
    if 'china' in doc:
        num_docs_containing_china += 1
```


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

TF-IDF

(continua do Código do slide anterior)

```
# calculando TF de "china"
intro_tf['china'] = intro_counts['china'] / intro_total
history_tf['china'] = history_counts['china'] / history_total

print('Term Frequency of "china" in intro is: {:.4f}'.format(intro_tf['china']))
print('Term Frequency of "china" in history is: {:.4f}'.format(history_tf['china']))
```

Term Frequency of "china" in intro is: 0.0000
Term Frequency of "china" in history is: 0.0101

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

TF-IDF

(continua do Código do slide anterior)

```
# calculando o IDF para todos os três termos nos dois documentos
num_docs = 2
intro_idf = {}
history_idf = {}
intro_idf['and'] = num_docs / num_docs_containing_and
history_idf['and'] = num_docs / num_docs_containing_and
intro_idf['kite'] = num_docs / num_docs_containing_kite
history_idf['kite'] = num_docs / num_docs_containing_kite
intro_idf['china'] = num_docs / num_docs_containing_china
history_idf['china'] = num_docs / num_docs_containing_china

print(intro_idf['and'])
print(history_idf['and'])
print(intro_idf['kite'])
print(history_idf['kite'])
print(intro_idf['china'])
print(history_idf['china'])
```



```
1.0
1.0
1.0
1.0
2.0
2.0
```

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

TF-IDF

(continua do Código do slide anterior)

```
# calculando o resultado para o documento "intro"
intro_tfidf = {}
intro_tfidf['and'] = intro_tf['and'] * intro_idf['and']
intro_tfidf['kite'] = intro_tf['kite'] * intro_idf['kite']
intro_tfidf['china'] = intro_tf['china'] * intro_idf['china']

# calculando o resultado para o documento "history"
history_tfidf = {}
history_tfidf['and'] = history_tf['and'] * history_idf['and']
history_tfidf['kite'] = history_tf['kite'] * history_idf['kite']
history_tfidf['china'] = history_tf['china'] * history_idf['china']

print(intro_tfidf['and'])
print(intro_tfidf['kite'])
print(intro_tfidf['china'])

print(history_tfidf['and'])
print(history_tfidf['kite'])
print(history_tfidf['china'])
```

```
0.027548209366391185
0.0440771349862259
0.0

0.030303030303030304
0.020202020202020204
0.020202020202020204
```

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

TF-IDF

- O IDF é muito útil, pois se pensarmos em um corpus de 1000 documentos e se a palavra ocorresse em todos (ou quase todos) os documentos, o valor de seu IDF seria próximo de 1.
- Uma palavra mais rara, teria o IDF mais próximo de 1000.
- Observe que temos uma forma de penalizar palavras que ocorrem em quase todos os documentos e, portanto, ajudam muito pouco a discriminar um documento do outro.

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

TF-IDF

- Assim, a métrica TF-IDF leva em consideração tanto a frequência de ocorrência de um termo em um documento bem como a frequência de ocorrência do termo em todos os documentos (o quão raro um termo é)
- Ou seja, quanto mais vezes uma palavra aparecer no documento, o TF (e, portanto, o TF-IDF) aumentará.
- Ao passo que, à medida que o número de documentos que contêm essa palavra aumenta, o IDF (e, portanto, o TF-IDF) dessa palavra diminui.

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

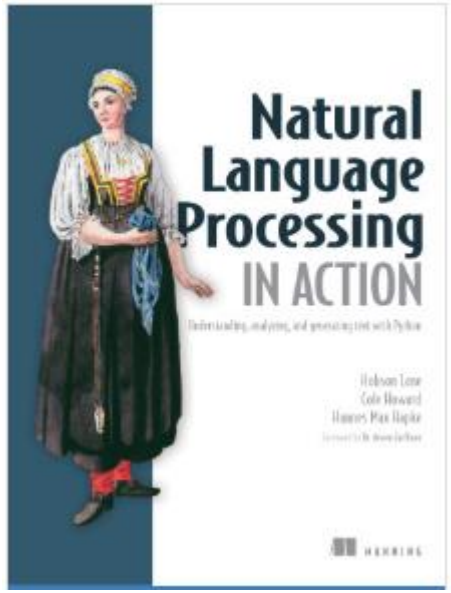
Considerações Finais

Tópicos Estudados

- Bag of words
- Vetorização
- Espaço vetorial
- TF-IDF

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