UNIDADE DE APRENDIZAGEM: Representação vetorial de textos – bag of words

Apresentação

O processamento de linguagem natural envolve diversas etapas e técnicas no processo de extração de conhecimento a partir de textos escritos ou falados. Uma pessoa pode aprender, ao longo da vida, a falar e a se comunicar em diversas línguas. Fazer com que um programa de computador compreenda alguma língua e seja capaz de extrair conhecimento a partir dessa língua é um desafio, pois computadores trabalham com base em sequências numéricas. Para que um algoritmo de aprendizagem de máquina seja capaz de fornecer conhecimento a partir de um texto, é preciso que esse texto seja tratado e manipulado.

Nesta Unidade de Aprendizagem, você aprenderá como extrair características de textos por meio do algoritmo *bag of words*. Também aprenderá como representar textos vetorialmente e aplicar essas técnicas em um texto.

Bons estudos.

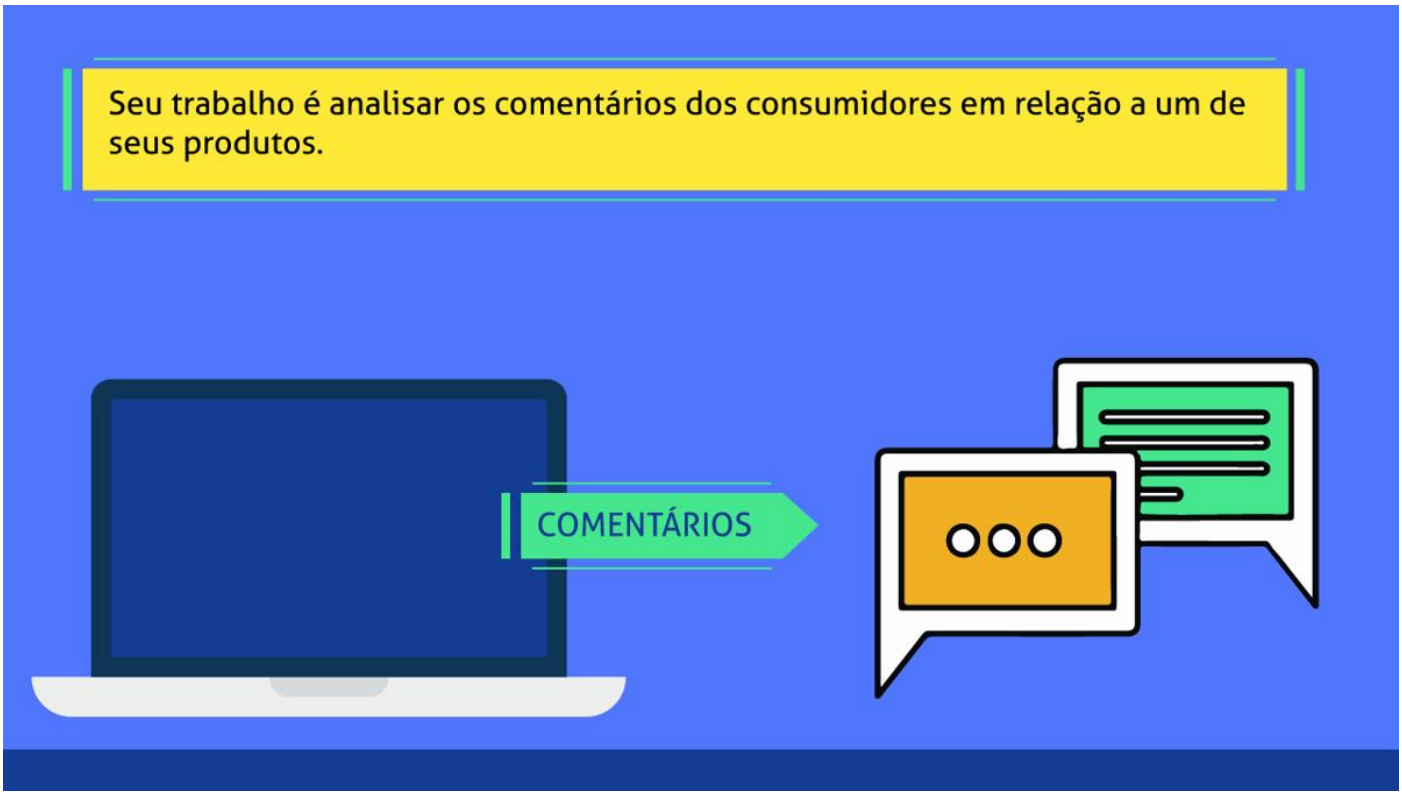
Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

- Descrever como o computador realiza a interpretação de dados textuais por conversão numérica.
- Definir o conceito de vetorização de palavras.
- Analisar o método *bag of words*.

Desafio

Extrair conhecimento de um texto envolve várias etapas. Diversas empresas e organizações têm apostado em aplicar técnicas de inteligência artificial para otimizar seus processos. Nesse contexto, identificar o comportamento do consumidor é fundamental para definir estratégias de mercado. Nos *sites* de *e-commerce*, é comum os consumidores registrarem suas avaliações acerca dos produtos adquiridos. Identificar e analisar essas avaliações podem fornecer conhecimentos importantes para as empresas.

Imagine que você foi contratado por uma empresa que vende *notebooks*.



Clique no botão a seguir para ver os comentários:

Clique aqui

Diante dessa situação, você deve aplicar o algoritmo *bag of words* utilizando a linguagem Python 3 para extrair características do texto. Essas características serão usadas futuramente como entrada para algoritmos de aprendizagem de máquina.

Infográfico

O processo de descoberta de conhecimento tem, basicamente, cinco etapas, entre as quais está a fase de pré-processamento dos dados, que é a fase na qual se prepara o conjunto de dados para o algoritmo de aprendizagem de máquina que irá recebê-lo. Os algoritmos de aprendizagem de máquina, geralmente, não apresentam boa acurácia quando recebem dados que estão fora dos padrões esperados. Para tratar esses dados, existem diversos algoritmos, cada qual para uma finalidade. Nesse contexto, o algoritmo *bag of words* apresenta algumas características importantes, como ser um algoritmo simples e de fácil implementação.

Neste Infográfico, você conhecerá algumas características do *bag of words*, como ele pode ser aplicado e também algumas desvantagens dele.

BAG OF WORDS

A aprendizagem de máquina requer várias técnicas aplicadas em conjunto para obtenção dos melhores resultados. Cada algoritmo de aprendizagem de máquina **requer um tratamento anterior** para que os dados de entrada possam produzir uma boa acurácia nos resultados. Nesse contexto, o algoritmo *bag of words* funciona como um algoritmo capaz de transformar textos em vetores numéricos, que podem ser usados por diferentes algoritmos de aprendizagem de máquina.

O ALGORITMO *BAG OF WORDS*

- Fornece uma maneira de **representar dados de texto** para algoritmos de aprendizado de máquina.
- Simples de entender.
- Fácil implementação.

NUVEM DE PALAVRAS

- Na prática, o algoritmo *bag of words* é usado para **extrair características de um texto**. Uma das características mais comuns é a frequência com que as palavras ocorrem no texto.
- Com a identificação da frequência, pode-se **apresentar a informação visualmente**, por meio de nuvens de palavras.

PROBLEMAS DO *BAG OF WORDS*

- A abordagem do algoritmo **não considera o significado semântico** das palavras no texto e ignora totalmente o contexto.
- Para documentos grandes, o tamanho do vetor pode ser enorme, resultando em **muito tempo de computação**.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Conteúdo do Livro

O objetivo do processamento da linguagem natural aplicada a textos é o processamento de informação textual, extraindo índices numéricos significativos a partir do texto, para então tornar essa informação acessível para os programas disponíveis nos sistemas de mineração de dados. Máquinas não compreendem linguagem como as pessoas são capazes de compreender, as máquinas precisam que a informação esteja em formato numérico para trabalhar eficientemente sobre ela. Por esse motivo, para trabalhar com processamento de linguagem natural, os textos precisam ser tratados e convertidos para formatos numéricos. A representação numérica de textos é chamada de representação vetorial, em que o texto é convertido em uma sequência de números que representa as características retiradas do texto.

No capítulo Representação vetorial de textos – *bag of words*, da obra *Processamento de linguagem natural*, base teórica desta Unidade de Aprendiagem, você aprenderá como usar o processo de vetorização para aplicar o algoritmo *bag of words* a fim de extrair as características de um texto.

Boa leitura.

PROCESSAMENTO DE LINGUAGEM NATURAL

Raiza Artemam de Oliveira

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Representação vetorial de textos — *bag of words*

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Descrever como o computador realiza a interpretação de dados textuais por conversão numérica.
- Definir o conceito de vetorização de palavras.
- Analisar o método *bag of words*.

Introdução

O algoritmo *bag of words* é uma técnica de processamento de linguagem natural usado para extrair características de um texto/documento, a partir da contagem da frequência das palavras em um documento. Nesse contexto, um documento pode ser definido conforme necessário como uma frase única ou toda a Wikipédia. A saída desse algoritmo é um vetor de frequência dos *tokens* no vocabulário.

Para implementar e aplicar esse algoritmo, é preciso compreender alguns passos, como limpar o texto, definir e extrair os *tokens* e construir um vocabulário. Neste capítulo, observaremos como os algoritmos de aprendizagem de máquina esperam receber os dados de entrada, o que é a vetorização de textos e sua importância para o bom funcionamento de alguns algoritmos, bem como aprender a executar o método de vetorização *bag of words*.

1 Interpretação de dados textuais

Computadores trabalham com bases numéricas, mais especificamente sequências binárias que indicam ou não a passagem de corrente elétrica por seus componentes de *hardware*. Quando usamos algum programa ou navegamos na internet, existem várias camadas que convertem a informação que nós vemos para a informação que a máquina é capaz de compreender, ou seja, números e corrente elétrica.

Em aplicações de processamento de linguagem natural, ocorre algo semelhante: o objeto do processamento da linguagem natural reside no fato de que a máquina consiga compreender e se comunicar com pessoas por meio da linguagem comum aos seres humanos; para isso, a linguagem como nós conhecemos deve ser tratada e convertida para a linguagem que a máquina possa compreender.

Geralmente, no processamento de linguagem natural temos um algoritmo de aprendizagem de máquina para extrair conhecimento dos dados passados a ele, aprendendo, de modo geral, a realizar um mapeamento de um valor de entrada para determinado valor de saída.

Os algoritmos de aprendizado de máquina são descritos como o aprendizado de uma função de destino (f) que mapeia melhor as variáveis de entrada (X) para uma variável de saída (Y) (BISHOP, 2006).

$$Y = f(X)$$

Essa é uma tarefa geral de aprendizado em que gostaríamos de fazer previsões no futuro (Y), dados novos exemplos de variáveis de entrada (X). Como não sabemos como é a função (f) ou sua forma, usamos um algoritmo de aprendizagem de máquina para descobri-las.

Nos algoritmos de aprendizagem de máquina, também existe um erro (e) independente dos dados de entrada (X).

$$Y = f(X) + e$$

Esse erro pode não ter atributos suficientes para caracterizar da melhor forma de mapeamento de X para Y, sendo chamado de erro irredutível porque, por melhor que seja a estimativa da função de destino (f), não podemos reduzir esse erro. Tanto a função e o erro são valores numéricos.

Embora cada algoritmo de aprendizagem de máquina tenha implementações diferentes das funções de aprendizado, geralmente se baseiam em valores numéricos, alguns em cálculos estatísticos e outros usando medidas de distância, mas sempre necessitando de valores numéricos para funcionar corretamente.

Assim, podemos dizer que inicialmente em um projeto de processamento de linguagem natural, o objetivo consiste em transformar textos em números, ou seja, em índices significativos, que podem, então, ser incorporados em outras análises, como classificação supervisionada ou não supervisionada.

2 Vetorização de textos

Os algoritmos de aprendizado de máquina operam em um espaço de recurso numérico, esperando entrada como uma matriz bidimensional em que linhas são instâncias e colunas, recursos ou características (BISHOP, 2006). Para realizar o aprendizado de máquina em texto, precisamos transformar nossos documentos em representações vetoriais, a fim de poder aplicar o aprendizado de máquina numérico, processo que leva o nome de extração de características ou vetorização e compreende um primeiro passo essencial para a análise sensível ao idioma (BENGFORT; BILBRO; OJEDA, 2018).

Ao processar o texto em linguagem natural, para extrair informações úteis de determinadas palavras usando técnicas de aprendizado de máquina, a palavra, ou o texto, deve ser convertida em um conjunto de números reais, ou seja, um vetor. Representar documentos numericamente nos permite executar análises significativas e cria as instâncias nas quais os algoritmos de aprendizado de máquina conseguem trabalhar e extrair conhecimento.

Na análise de texto, as instâncias são documentos ou enunciados inteiros, que podem variar em comprimento, mas cujos vetores têm sempre tamanho uniforme (BENGFORT; BILBRO; OJEDA, 2018). Cada propriedade da representação vetorial é uma característica. Para o texto, as características representam atributos e propriedades dos documentos, incluindo seu conteúdo e meta atributos, como comprimento do documento, autor, fonte e data da publicação. Quando considerados juntos, as características de um documento descrevem um espaço multidimensional no qual os métodos de aprendizado de máquina podem ser aplicados.

Para compreender melhor como os algoritmos de aprendizado de máquina funcionam em relação ao processamento de textos, precisamos mudar a maneira como pensamos sobre a linguagem, de uma sequência de palavras para pontos que ocupam um espaço semântico. Os pontos no espaço podem estar próximos ou distantes, bem agrupados ou distribuídos uniformemente. O espaço semântico é, portanto, mapeado de tal maneira que documentos com significados semelhantes estão mais próximos e aqueles que são diferentes estão mais afastados. Ao codificarmos a similaridade como a distância, podemos começar a derivar os componentes principais dos documentos e traçar limites de decisão em nosso espaço semântico.

A codificação mais simples do espaço semântico consiste no modelo de saco de palavras, cuja ideia principal reside no fato de que o significado e a semelhança são codificados no vocabulário — por exemplo, os artigos da Wikipédia sobre futebol e Pelé são provavelmente muito semelhantes; não apenas muitas das mesmas palavras aparecerão em ambas, como também não compartilharão muitas palavras em comum com artigos sobre caçarolas ou flexibilização quantitativa. Embora simples, esse modelo é extremamente eficaz.

Saiba mais

Muitas vezes, para programar em alguma linguagem e testar algumas de suas funcionalidades, temos certo trabalho para encontrar todos os pacotes necessários e configurar o ambiente de desenvolvimento. Algumas ferramentas podem ajudar nesse processo, como é o caso da Anaconda, uma distribuição gratuita e de código aberto das linguagens de programação Python/R para computação científica, que visa a simplificar o gerenciamento e a implantação de pacotes. Para testar o pacote nltk e sua implementação do algoritmo *bag of words* no Python, basta acessar o *link* e instalar a Anaconda.

https://qrgo.page.link/Dp2LL

3 Algoritmo *bag of words*

Método usado para extrair características e informações de um texto, geralmente é empregado em conjunto com outros algoritmos no processo de aprendizagem de máquina, já que as características fornecidas por ele são utilizadas na fase de treinamento de algoritmos de aprendizagem de máquina, como o Naive Bayes (SARKAR, 2016). Resumidamente, o algoritmo *bag of words* ("saco de palavras") gera um conjunto de palavras de um texto, sendo amplamente utilizado na recuperação de informações de documentos, classificação de documentos e processamento de linguagem natural de forma geral (JURAFSKY, 2000). Pode-se dividir em quatro etapas:

1. **Limpar o texto:** as palavras sem relevância para o conteúdo são removidas, como as *stopwords*, artigos, verbos de ligação ou o que o programador definir como não relevantes. Nessa etapa, também é removida a pontuação do texto.
2. **Extrair os *tokens*:** o texto é separado em *tokens*, conforme a necessidade da aplicação. Geralmente, cada palavra é considerada um *token*, mas podemos considerá-los também frases inteiras ou sílabas.
3. **Construir o vocabulário:** após a limpeza do texto e a extração de *tokens*, construímos o vocabulário com os *tokens* extraídos.
4. **Gerar os vetores:** são gerados os vetores com as características do texto. Para cada *token*, associa-se sua frequência no texto.

Para compreender melhor esses conceitos e como são aplicados, analisaremos o texto a seguir.

> Paulo e Cintia foram ao cinema sem comprar ingressos. Não havia mais ingressos à venda, então Paulo comeu bolo e Cintia comeu pipoca.

Essas frases podem ser representadas como uma coleção de palavras da seguinte forma:

```
['Paulo', 'e', 'Cintia', 'foram', 'ao', 'cinema', 'sem', 'comprar', 'ingressos', 'Não', 'havia', 'mais' ,'ingressos', 'à', 'venda,', 'então', 'Paulo', 'comeu', 'bolo', 'e', 'Cintia', 'comeu', 'pipoca']
```

Após termos uma coleção de palavras, devemos remover aquelas repetidas e contar a ocorrência de cada uma delas. O resultado desta operação é mostrado a seguir.

Palavra	Quantidade de ocorrências
Paulo	2
Cintia	2
e	2
foram	1
ao	1
cinema	1
sem	1
comprar	1
ingressos	2
Não	1
Havia	1
mais	1
à	1
venda	1
então	1
comeu	2
bolo	1

Neste pequeno texto, vemos que as palavras com mais frequência são: Paulo, e, Cintia e comeu; todas aparecendo duas vezes no texto. No formato de vetor, a representação do resultado de ocorrências é:

```
{"Paulo": 2,  'e': 2, 'Cintia': 2,  'foram': 1, 'ao': 1,
'cinema': 1, 'sem': 1, 'comprar': 1, 'ingressos': 2, 'Não':
1,  'havia': 1, 'mais': 1 'à': 1, 'venda,': 1, 'então': 1,
'comeu': 2, 'bolo': 1,'pipoca': 1}
```

Essa estrutura na linguagem Python compreende um dicionário no qual a palavra indica a chave e a quantidade aponta o valor associado à chave. A essa estrutura que criamos com todas as palavras e a e sua respectiva contagem, damos o nome de vocabulário, por meio do qual podemos criar vetores para cada frase do texto. Por padrão, o tamanho do vetor gerado é sempre igual ao tamanho do vocabulário; nesse caso, o vetor terá o tamanho 18. O vetor deve ser inicializado com todos os índices com valores zero:

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

O vetor gerado para a primeira frase "Paulo e Cintia foram ao cinema sem comprar ingressos." é:

```
[2,2,2,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0]
```

Nesse vetor, observamos que existem muitos elementos iguais a zero, o que ocorre sempre que o tamanho do vocabulário for muito grande ou mesmo quando houver variações das palavras. Essa quantidade de elementos iguais a zero acaba gerando vetores que chamamos de esparsos, o que pode configurar um problema para armazenar e manipular esses dados. Uma das maneiras de reduzir o tamanho do vocabulário consiste em remover as palavras sem relevância para o significado geral, como os artigos **o**, **a**, **os**, **as**, **e**, **ao**. No nosso exemplo, podemos diminuir o vocabulário para o tamanho quinze removendo essas palavras.

A linguagem de programação Python fornece diversos pacotes com métodos para aplicações científicas, como os pacotes scipy e nltk; o último fornece um conjunto de bibliotecas com métodos para processamento de linguagem natural para inglês, no entanto também é possível usá-lo para a língua portuguesa.

A biblioteca Scikit-Learn é uma biblioteca de aprendizado de máquina gratuita e de código aberto escrita na linguagem Python, a qual oferece diversos métodos para implementar algoritmos de aprendizagem de máquina e inteligência artificial, como regressão linear, classificadores, SVM, redes neurais convolucionais, etc. (BENGFORT; BILBRO; OJEDA, 2018), além de dispor de alguns conjuntos de dados com amostras que podem ser usados diretamente para treinamento e teste dos algoritmos.

Essa biblioteca também fornece métodos para vetorização de textos, por meio dos quais é possível aplicar as etapas do *bag of words* de maneira eficiente e mesmo aplicar pré-processamento e regras sobre o número e a frequência dos termos. O Scitkit-Learn oferece três principais tipos de vetorizadores de textos (SARKAR, 2016):

1. **CountVectorizer:** o mais simples, conta o número de vezes que um termo aparece no documento e usa esse valor como peso.
2. **HashVectorizer:** oferece boa eficiência em relação ao uso da memória. Em vez de armazenar as palavras como *strings*, o vetorizador aplica um *hash* para codificá-los como índices numéricos. A desvantagem desse método reside no fato de que, uma vez vetorizado, os nomes das características não podem mais ser recuperados.
3. **TF-IDFVectorizer:** TF-IDF significa "frequência de documento inversa à frequência do termo", indicando que o peso atribuído a cada termo não depende apenas de sua frequência em um documento, mas também de sua recorrência em todo um conjunto de documentos.

O método CountVectorizer pode receber os seguintes parâmetros (PEDREGOSA *et al*., 2011):

- `Input: {'filename', 'file', 'content'}`: se `filename`, espera-se que a sequência passada como um argumento adequado seja uma lista de nomes de arquivos que precisam ser lidos para buscar o conteúdo bruto a ser analisado. Se `file`, os itens da sequência devem ter um método de leitura chamado para buscar os bytes na memória. Caso contrário, espera-se que a entrada seja uma sequência de itens do tipo *string* ou byte.
- `Encoding`: por padrão, o tipo de `encoding` é o `utf-8`. Se forem fornecidos bytes ou arquivos para análise, usaremos essa codificação para decodificar.
- `decode_error {'strict', 'ignore', 'replace'}`: instruções sobre o que fazer se for fornecida uma sequência de bytes para analisar que contém caracteres que não fazem parte da codificação especificada. Por padrão, o valor é `strict`, o que significa que um `UnicodeDecodeError` será gerado. Outros valores são `ignore` e `replace`.

- `strip_accents {'ascii', 'unicode', None}`: remove os acentos e executa a normalização de outros caracteres durante a etapa de pré-processamento. `ASCII` é um método rápido que funciona apenas em caracteres com um mapeamento `ASCII` direto. Já o `Unicode` é um método um pouco mais lento que funciona em qualquer caractere. O `None` não faz nada.
- `Lowercase boolean`: por padrão, é verdadeiro. Converte todos os caracteres em minúsculas antes de tokenizar.
- `Preprocessor callable` ou `None` (padrão): pré-processador programável ou substitui o estágio de pré-processamento (transformação de cadeia), preservando as etapas de geração de *token* e n-gramas. Aplica-se apenas se o analisador não puder ser chamado.
- `Tokenizer callable` ou `None` (padrão): substitui a etapa de tokenização de cadeia, preservando as etapas de pré-processamento.
- `stop_words`: se for para língua inglesa, existe uma lista de *stop words* predefinida. Se for uma lista, presume-se que ela contenha *stopwords*, removidas dos *tokens* resultantes.
- `token_pattern`: expressão regular que denota o que constitui um *token*. O regexp padrão seleciona *tokens* de dois ou mais caracteres alfanuméricos; a pontuação é completamente ignorada e sempre tratada como um separador de *token*.
- `ngram_rangetuple (min_n, max_n)`: por padrão, é (1, 1), indicando o limite inferior e superior do intervalo de valores `n` para diferentes palavras n-gramas ou n-gramas de caracteres a serem extraídos. Todos os valores de `n` tais que `min_n <= n <= max_n` serão usados. Por exemplo, um intervalo de n-grama de (1, 1) significa apenas unigramas, (1, 2) significa unigramas e bigramas e (2, 2) significa apenas bigramas. Aplica-se apenas se o analisador não puder ser chamado.
- `analyzerstring, {'word', 'char', 'char_wb'}`: a opção `char_wb` cria caracteres n-gramas apenas a partir do texto dentro dos limites das palavras; n-gramas nas bordas das palavras são preenchidos com espaço. Se uma chamada for aprovada, ela será usada para extrair a sequência de características da entrada bruta e não processada. Está presente, desde a versão 0.21, se a entrada for nome de arquivo ou arquivo; os dados são lidos primeiro a partir do arquivo e, depois, passados para o analisador de chamada especificado.

- `max_dffloat` no intervalo `[0,0, 1,0]` ou `int, padrão = 1,0`: ao criar o vocabulário, ignora os termos que têm uma frequência de documento estritamente maior que o limite fornecido. Se `float`, o parâmetro representa uma proporção de documentos, número absoluto de contagens.
- `min_dffloat` no intervalo `[0,0, 1,0]` ou `int, padrão = 1`: ao criar o vocabulário, ignora os termos que tenham uma frequência de documento estritamente menor que o limite especificado. Esse valor também é chamado de corte na literatura. Se `float`, o parâmetro representa uma proporção de documentos, número absoluto de contagens.
- `max_features int` ou `None`: por padrão, é `None`; caso contrário, cria um vocabulário que considera apenas as principais características máximas ordenadas por frequência do termo no texto. Esse parâmetro será ignorado se o vocabulário não for `None`.
- `Vocabulary`: um mapeamento (p. ex., um ditado) em que chaves são termos e valores, índices na matriz de características ou uma iterável sobre os termos. Se não for fornecido, um vocabulário é determinado a partir dos documentos de entrada. Os índices no mapeamento não devem ser repetidos e apresentar nenhum intervalo entre 0 e o maior índice.
- `Binary boolean`: por padrão, é `False`; se o valor for `True`, todas as contagens diferentes de 0 são definidas como 1. Isso é útil para modelos probabilísticos discretos que modelam eventos binários em vez de contagens inteiras.
- `Dtype type`: indica o tipo da matriz retornada por `fit_transform ()` ou `transform ()`.

Para compreender como o `CountVectorizer` funciona no Python, implementaremos um método. Para usar o método `CountVectorizer`, devemos importá-lo da seguinte forma:

```
from sklearn.feature_extraction.text import CountVectorizer
```

Com a biblioteca e o método importado, definiremos o texto que será vetorizado. Na prática, em um processo de análise de linguagem natural, o texto geralmente estará em *sites* na internet, em documentos e arquivos de diferentes formatos e fontes, mas, para exemplificar como aplicar o `CountVectorizer`, usaremos um texto curto:

```
texto = ['No verão muito há mais chuva.','Neste inverno não houve chuva intensa.']
```

Primeiro, devemos instanciar o método:

```
countvec = CountVectorizer(lowercase=False)
```

O parâmetro `lowercase` recebe o valor `False` para indicar que todas as letras devem permanecer com o mesmo *case*, ou seja, as maiúsculas permanecem maiúsculas. Se o valor de `lowercase` fosse `True`, todas as letras seriam convertidas para minúsculas. Após a instanciação do objeto para a vetorização do texto por meio do método `CountVectorizer`, podemos gerar a matriz termo-documento por meio do método `fit_transform`.

```
matrix = countvec.fit_transform(texto)
```

Para recuperar as características do texto, basta aplicar o método `get_features_names()`:

```
tokens = countvec.get_feature_names()
```

Com esses três passos, temos os vetores gerados com a frequência de cada termo. A esses passos, acrescentamos um método para apresentar os vetores em uma matriz. Na Figura 1, você pode ver o código completo.

```python
#importa as bibliotecas
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

#cria um dataframe para apresentar a matriz com o resultado
def geradf(wm, feat_names):
    #cria um indice para cada linha
    doc_names = ['Doc{:d}'.format(idx) for idx, _ in enumerate(wm)]
    df = pd.DataFrame(data=wm.toarray(), index=doc_names,columns=feat_names)
    return(df)

#incializa os documentos para serem analisados
documentos =['No verão muito há mais chuva.','Neste inverno não houve chuva intensa.']

#instancia o objeto
countvec = CountVectorizer(lowercase=False)

#converte os documentos em uma matriz termo-documento
matrix = countvec.fit_transform(documentos)

#recupera os termos encontrados nos documentos
tokens = countvec.get_feature_names()

#cria um dataframe a partir da matriz termo-documento
print(geradf(matrix, tokens))
```

Figura 1. Código em Python com exemplo de utilização do `CountVectorizer`.

A matriz termo-documento gerada é apresentada a seguir.

	Neste	No	chuva	houve	há		intensa	inverno	mais	muito	não	verão
Doc0	0	1	1	0	1		0	0	1	1	0	1
Doc1	1	0	1	1	0		1	1	0	0	1	0

Referências

BENGFORT, B.; BILBRO, R.; OJEDA, T. *Applied text analysis with python*: enabling language-aware data products with machine learning. [*S. l.*]: O'Reilly Media, 2018.

BISHOP, C. M. *Pattern recognition and machine learning*. [*S. l.*]: Springer, 2006.

JURAFSKY, D. *Speech & language processing*. [*S. l.*]: Pearson Education, 2000.

PEDREGOSA, F. *et al.* Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

SARKAR, D. *Text analytics with Python*: a practical real-world approach to gaining actionable insights from your data. Bangalore: Apress, 2016.

Fique atento

Os *links* para *sites* da Web fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Dica do Professor

A linguagem de programação Python é uma das linguagens mais populares para o desenvolvimento de aplicações de métodos de

inteligência artificial e de ciência de dados. Uma das razões pela qual essa linguagem é popular neste meio se deve aos seus pacotes e as suas bibliotecas, que fornecem diversos métodos e algoritmos já implementados. O pacote NLTK (Natural Language Toolkit) é um pacote que fornece a implementação de algoritmos de processamento de linguagem natural em Python.

Nesta Dica do Professor, você vai estudar como utilizar os métodos fornecidos pelo NLTK para aplicar o algoritmo *bag of words* em um texto minerado da Web.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Exercícios

1) **Uma das etapas do algoritmo *bag of words* consiste em analisar a frequência das palavras em seu vocabulário.**

Nesse contexto, assinale a alternativa que indica qual seria o resultado desta etapa para o seguinte texto:

Maria comprou pão e leite na padaria.

Marcos foi à padaria e comprou leite e queijo.

A) {"Maria":1, "comprou":2,, "pão":1, "leite":2, "padaria":2, "queijo":1, "Marcos":1,"e":3,"na":1, "foi":1, "a":1}

B) {"Maria":1, "comprou":2,, "pão":1, "leite":2, "padaria":2, "queijo":2, "Marcos":1, "foi":1}

C) {"Maria":1, "comprou":1, "pão":1, "leite":2, "padaria":2, "queijo":2, "Marcos":1,"comprou":1,"e":3,"na":1, "foi":1, "a":1}

D) {"Maria":1, "comprou":1,"na":1, "pão":1, "e":1,"leite":2, "padaria":1} e {"queijo":1, "Marcos":1,"e":2, "foi":1, "a":1, "comprou":1, "leite":1, "padaria":1}

E) {"Maria":1, "comprou":2,"na":1, "pão":2, "e":3,"leite":2, "padaria":2} e {"queijo":1, "Marcos":1,"e":3, "foi":1, "a":1, "comprou":2, "leite":2, "padaria":2}

2) **Uma das primeiras etapas do algoritmo *bag of words* é a definição do vocabulário a ser analisado a partir do texto. Considere o seguinte texto:**

Todos foram ao cinema ontem.

Lucas não gostou do filme, mas gostou do cinema.

O filme era de terror.

Assinale a alternativa que apresenta o vocabulário desse texto.

A) ["Todos", "foram", "ao", "cinema", "ontem"]; e ["Lucas", "não", "gostou", "do", "filme"," mas", "gostou", "do", "cinema"] e ["O", "filme", "era", "de", "terror"]

B) ["Todos", "foram", "cinema", "ontem", "Lucas", "não", "gostou", "filme", "gostou", "cinema", "filme", "terror"]

C) ["Todos", "foram", "ao", "cinema", "ontem", "Lucas", "não", "gostou", "do", "filme"," mas", "gostou", "do", "cinema", "O", "filme", "era", "de", "terror"]

D) ["Todos", "foram", "ao", "cinema", "ontem", "Lucas", "não", "gostou", "do", "filme"," mas", "O", "era", "de", "terror"]

E) ["cinema", "gostou", "do", "filme", "gostou", "do", "cinema", "filme", "Todos", "era" , "terror"]

3) O algoritmo *bag of words* tem quatro grandes etapas para a extração de características do texto de um documento.

Assinale a alternativa que apresenta quais são essas etapas.

A) As etapas do *bag of words* são: limpar o texto, definir as *stopwords*, construir o vocabulário e criar o vetor de características.

B) As etapas do *bag of words* são: limpar o texto, extrair os *tokens*, construir o vocabulário e gerar os vetores de características.

C) As etapas do *bag of words* são: limpar o texto, remover os *tokens*, construir o vocabulário e criar o vetor de características.

D) As etapas do *bag of words* são: limpar o texto, construir o vocabulário, contar a frequência das palavras e criar o vetor de características.

E) As etapas do *bag of words* são: limpar o texto, definir as palavras, construir os vetores individuais e criar o vetor de características.

4) No contexto do algoritmo *bag of words*, é possível se referir aos elementos do vocabulário como *tokens*.

Assinale a alternativa que define corretamente o que é um *token* no *bag of words*.

A) *Token* é a palavra com maior frequência em um vocabulário, pode-se definir um *token* como palavras, frases completas ou até mesmo sílabas.

B) *Token* é a menor unidade de um vocabulário, pode-se definir um *token* como palavras, frases completas ou até mesmo sílabas.

C) *Token* é a frequência da unidade de um vocabulário, pode-se definir um *token* como palavras, frases completas ou até mesmo sílabas.

D) *Token* é o resultado de um vocabulário, pode-se definir um *token* como palavras, frases completas ou até mesmo sílabas.

E) *Token* é a maior unidade de um vocabulário, pode-se definir um *token* como palavras, frases completas ou até mesmo sílabas.

5) **A linguagem de programação Python fornece diversos pacotes que auxiliam o desenvolvedor por meio de métodos que implementam algoritmos para diversos fins.**

Assinale a alternativa que apresenta o pacote em Python que oferece métodos para o processamento de linguagem natural.

A) Scipy.

B) Beautiful Soup.

C) scikit-learn.

D) NLTK.

E) NumPy.

Na prática

O processamento de linguagem natural auxilia diversas áreas do conhecimento a produzirem melhores resultados e a otimizarem processos. Analisar e extrair conhecimento de documentos são uma das principais atividades do processamento de textos.

Neste Na Prática, você irá conhecer como Marcela aplicou o algoritmo *bag of words* para analisar documentos jurídicos e extrair conhecimentos, que melhoraram o resultado dos processos.

EXTRAINDO INFORMAÇÕES DE DOCUMENTOS

CONTEXTUALIZAÇÃO

Marcela é analista de dados em uma empresa que oferece, para organizações de diferentes áreas, soluções baseadas em aprendizagem de máquina. Recentemente, a equipe em que Marcela trabalha foi designada para auxiliar um escritório jurídico a avaliar as petições escritas por seus advogados e **identificar quais as características textuais** das petições que obtiveram resultados positivos nos tribunais.

ANÁLISE

Para realizar a análise, Marcela e sua equipe implementaram o **algoritmo *bag of words* utilizando Python**. Eles optaram por utilizar esse algoritmo por se tratar de uma análise inicial e pelo qual poderiam apresentar resultados rápidos para os seus clientes. A equipe de Marcela, juntamente com um consultor jurídico, definiram um conjunto de palavras como "não relevantes" e aplicaram esse conjunto de palavras na etapa de limpeza dos documentos. Após a limpeza, eles extraíram o vocabulário de cada uma das petições, definiram cada palavra como um token e, por fim, aplicaram o processo de vetorização do texto.

FINALIZANDO A ANÁLISE

- Marcela salvou o **texto vetorizado em arquivos CSV** para facilitar a análise. Veja a seguir, um trecho da matriz de termo e documento gerada:

	Destinatário	Excelso	Federal	Na	Nada	Os	Pretório	Que	Supremo	Tribunal	ab	academias	acessível
Doc0	0	1	1	1	0	0	1	0	1	1	1	0	0
Doc1	0	0	0	0	0	0	0	0	0	0	0	0	0
Doc2	0	0	0	0	0	0	0	1	0	0	0	0	0
Doc3	1	0	0	0	1	1	0	0	0	0	0	1	1

CONCLUSÃO

Marcela e sua equipe identificaram algumas sentenças e palavras comuns nas petições com resultados positivos. Assim, ficou claro que **é produtivo aplicar um algoritmo simples para extrair conhecimento de documentos.**

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Saiba mais

Para ampliar o seu conhecimento a respeito desse assunto, veja abaixo as sugestões do professor:

Proposta de *bag of visual words* por meio de redes complexas

A aplicação de técnicas de inteligência artificial vem auxiliando processos em diversas áreas do conhecimento. No contexto da agricultura, técnicas de reconhecimento são aplicadas desde a escolha das sementes no plantio até o controle de pragas em plantações. Neste trabalho, a autora desenvolveu uma pesquisa para identificar boas sementes de soja por meio da adaptação do algoritmo *bag of words* para a visão computacional.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Inteligência artificial aplicada à detecção de *fake news*

O processo de extrair informações e conhecimentos de um texto requer várias etapas. Neste processo, diferentes algoritmos podem ser aplicados de formas combinadas. Neste artigo, os autores investigaram aplicações de inteligência artificial para detectar *fake news* em redes sociais. Os autores aplicaram o algoritmo *bag of words* para a extração de características dos textos analisados.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Extrair conhecimento em comentários gerados em mídias sociais utilizando análise de sentimentos

Identificar como os consumidores se relacionam, ou como avaliam cada produto, é um passo importante para as organizações desenvolverem suas estratégias de mercado. Nesse contexto, a avaliação de sentimentos em comentários na Internet pode fornecer conhecimento valioso para empresas. Neste artigo, os autores apresentam como o processo de análise de sentimentos pode

obter a avaliação de consumidores em relação a produtos por meio da análise de comentários. Os autores propuseram um modelo preditivo, utilizando o algoritmo *bag of words* para extração de características e o algoritmo de Bayes.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.