

UNIDADE DE APRENDIZAGEM: Representação vetorial de textos – utilizando word embeddings em Python

Apresentação

Em um mundo rodeado de dispositivos conectados à Internet, o antes tão eficaz sistema de busca e pesquisa tem-se tornado um processo cada vez mais difícil. Nos dias de hoje, inúmeros são os dispositivos que podem acessar as informações da Internet ou enviar seus próprios dados para ela; daí a importância de se analisar e compreender essas informações e de todas as demais fontes que ininterruptamente fazem a Internet crescer a cada segundo.

A biblioteca Gensim pode criar modelos *GloVe*, *word2vec*, *TF-IDF*, *fastText* e muitos outros. Maior ênfase é dada, no entanto, à modelagem de tópicos, técnica que pode facilitar a correlação de páginas e documentos ao analisar e encontrar principalmente semelhanças de tema.

Nesta Unidade de Aprendizagem, você vai conhecer a biblioteca Gensim e ver como ela é utilizada para criar seus próprios modelos e até compartilhá-los ou, ainda, como é possível partir de modelos pré-treinados, alguns com quantidades imensas de palavras e dimensões.

Bons estudos.

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

- Identificar os recursos disponibilizados pela biblioteca Gensim.
- Usar modelos pré-treinados de *word2vec*.
- Desenvolver modelos pré-treinados de *word2vec*.

Desafio

O pré-processamento pode tanto melhorar o desempenho de um modelo quanto dificultá-lo se algumas precauções não forem tomadas. Por exemplo, em sistemas de análise de sentimentos, boa parte da informação está contida na pontuação, que em muitos é simplesmente removida. Já em outras situações, alguns símbolos podem ser responsáveis pela remoção de representação e também tornar mais difícil de comparar com outros vetores.

Na qualidade de profissional, considere o seguinte cenário:

Você está desenvolvendo um aplicativo para catalogar livros de uma biblioteca digital conforme o assunto de que tratam.

Para tanto, você vai utilizar o conjunto de dados *brown* (no idioma inglês) presente na biblioteca NLTK como base para um modelo *word2vec* da biblioteca Gensim. Nesse caso, suas tarefas incluem:



- remover as *stopwords*, palavras de alta frequência, como *I*, *if*, *the*, existentes no conjunto `nltk.corpus.stopwords('english')`;
- forçar todos os caracteres para letras minúsculas;
- filtrar o conjunto para que apenas um representante de cada lema das palavras apareça;
- montar um vocabulário em que estejam presentes apenas as 100 palavras de lemas distintos mais próximas da palavra *education* (já que o *corpus* está em inglês).

Para isso, você pode usar a função de similaridade por palavra do modelo *word2vec* criado (*similar_by_word*), buscando por um conjunto grande o suficiente para conter cada uma das 100 palavras de lemas distintos buscados.

Esse algoritmo será na linguagem Python e terminará apresentando a lista de palavras do dicionário criado segundo os procedimentos citados.

Diante do exposto, resolva as tarefas elencadas, **apresentando o algoritmo em Python**.

Lembre-se: a limitação de palavras se deve ao custo computacional envolvido, já que o sistema tem limitações de recurso. Assim, ao proceder com o tratamento das palavras, para evitar que vetores cuja palavra tenha o mesmo lema sejam replicados no conjunto final, tende-se a aumentar a eficiência do aplicativo.

Infográfico

O desempenho em algoritmos de processamento de linguagem natural depende diretamente das escolhas e das decisões tomadas durante o processo de modelagem e treinamento, o que inclui, ainda, a seleção e a adequação de textos para alimentar o processo. Elementos de pontuação, espaços, diferenças entre letras maiúsculas e minúsculas podem agregar informações úteis ao treinamento do modelo em alguns casos, mas, na maior parte deles, devem ser eliminados ou tratados devidamente. Passada a etapa de pré-processamento, o modelo poderá ser treinado e até mesmo compartilhado ao final do treinamento para que outros possam utilizá-lo e até mesmo aprimorá-lo.

No Infográfico, aproveite para conhecer as etapas de criação de um modelo *word2vec*.

ETAPAS DE CONSTRUÇÃO DE UM MODELO WORD2VEC

O processamento de linguagem natural passa por diversas etapas de tratamento, modelagem e, principalmente no caso de *word embeddings*, compartilhamento de modelos pré-treinados para aumentar a eficiência e a agilidade no desenvolvimento das aplicações. Veja como essas etapas aparecem durante a construção de um modelo do *word2vec*.



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

```
import nltk
from nltk.corpus import stopwords
from gensim.models import Word2Vec

arquivos = [ 'C:\\\\Sagah\\\\arquivo1.txt', 'C:\\\\Sagah\\\\arquivo2.txt' ]

nltk.download('stopwords')
stops = stopwords.words('english')

docs_tokens = []
for arq in arquivos:
    file = open(arq,'r', errors='ignore')
    texto = file.read().lower()

    tokens = nltk.word_tokenize(texto)
    tokens_sem_stops = [w for w in tokens if not w in stops]
    docs_tokens.append(tokens_sem_stops)

from gensim.models import Word2Vec

model = Word2Vec(tokens_sem_stops, min_count=1, size=10)

model.save('C:\\\\Sagah\\\\modelo.bin')
model = Word2Vec.load('C:\\\\Sagah\\\\modelo.bin')
```

1 PRÉ-PROCESSAMENTO

Consiste na preparação do *corpus* e dos documentos nele presentes. Inclui remoção de acentuações, padronização do texto em minúscula ou maiúscula, eliminação de palavras denominadas *stopwords*, que são comuns em qualquer contexto, *tokenização*, enfim, os textos são trabalhados e transformados visando a aumentar a eficiência no treinamento do modelo.



EXEMPLOS

- *word_tokenize*: cria um *token* para cada palavra distinta encontrada.
- *if not w in stops*: não incluirá as palavras que constam no vetor *stops* com a lista de *stopwords* carregada da biblioteca NLTK.

2 DESENVOLVIMENTO/TREINAMENTO

Nesta etapa, são configurados **hiperparâmetros**, como a taxa de aprendizado, que terá influência combinada com o tamanho do conjunto de dados disponível. Outros parâmetros incluem a quantidade de núcleos de processamento disponíveis, as limitações de memória e as configurações próprias do algoritmo de *word2vec*.



EXEMPLOS

- *min_count*: frequência mínima para a palavra não ser ignorada.
- *size*: dimensão dos vetores a serem criados.

3 COMPARTILHAMENTO

Tanto o modelo produzido quanto outros modelos são frequentemente distribuídos e compartilhados na Internet. Esses modelos, em geral, demandaram vasta quantidade de informações e tempo de treinamento, sendo muito importante a distribuição de bancos de dados que auxiliem as pessoas na construção de aplicações mais eficientes.



EXEMPLOS

- *model.save*: salva o modelo treinado em um arquivo.
- *Word2Vec.load*: carrega um modelo a partir de um arquivo.

Conteúdo do Livro

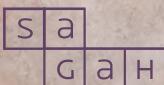
Implementar um algoritmo de processamento de linguagem natural pode ser uma tarefa complexa. É por isso que diversas bibliotecas e sistemas têm sido desenvolvidos para essa área, que cresce em ritmo acelerado. A biblioteca Gensim tem como finalidade a modelagem de tópicos, por exemplo, muito útil em tarefas de busca ou pesquisa.

No capítulo Representação vetorial de textos – utilizando *word embeddings* em Python, da obra *Processamento de linguagem natural*, você vai acompanhar a criação de um modelo *word2vec* com o auxílio da biblioteca Gensim, especialmente concebida para o desenvolvimento de modelagem de tópicos.

Boa leitura.

PROCESSAMENTO DE LINGUAGEM NATURAL

Maikon Lucian Lenz



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS

Representação vetorial de textos — utilizando *word embeddings* em Python

Objetivos de aprendizagem

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

- Identificar os recursos disponibilizados pela biblioteca Gensim.
- Usar modelos pré-treinados de word2vec.
- Desenvolver modelos pré-treinados de word2vec.

Introdução

O word2vec é um modelo relevante, mas complexo de implementar, exigindo a utilização de bibliotecas para diminuir o esforço necessário para seu treinamento, o que facilita o seu emprego.

Neste capítulo, utilizaremos como exemplo a biblioteca Gensim, já que dispõe de ferramentas de pré-processamento e leitura/escrita de arquivos de modelos pré-treinados.

A partir de agora, você conhecerá a biblioteca Gensim e a importância de suas funções de sintaxe simples, especialmente no mundo acadêmico. Ainda, verá que, mesmo com poucas linhas de código, é possível treinar um modelo ou realizar operações a partir de um modelo já treinado.

1 Biblioteca Gensim

Para analisar de maneira não supervisionada a imensidão de dados criados diariamente, foram desenvolvidos algoritmos de modelagem tópica, que utilizam métodos estatísticos para analisar a composição de palavras presentes nos documentos e tentar descobrir o tema abordado para melhorar, por exemplo, a maneira como estes são relacionados em uma busca. Assim, o sistema convencional de *links* e palavras-chave pode ser aprimorado para melhor atender

às reais expectativas do navegador (BLEI, 2012). Obviamente, tais técnicas não se restringem a mecanismos de busca, embora esse contexto confira exatamente a dimensão dos problemas e das soluções buscadas a eles associados.

Aproveitando o desenvolvimento desses métodos de modelagem tópica, deu-se origem à biblioteca Gensim, uma biblioteca de processamento de linguagem natural (PLN), de código aberto, originalmente desenvolvida para modelagem de tópicos por Radim Řehůřek.

A biblioteca foi criada na linguagem Python, o que requer o uso de um interpretador. Os exemplos deste capítulo fazem uso do sistema Jupyter integrado à plataforma Anaconda — para utilizá-lo, você poderá instalar a plataforma Anaconda, que já inclui o Jupyter e outros sistemas e bibliotecas.



Saiba mais

Faça uma busca pela biblioteca “Gensim” e procure pelo *link* oficial disponibilizado na página pessoal de Radim Řehůřek, na qual você poderá realizar o *download* da ferramenta e consultar a documentação integral da biblioteca. Contudo, ressaltamos que esta é apenas uma das formas de realizar o *download* da biblioteca.

Já a plataforma Anaconda + Jupyter pode ser encontrada na página oficial do aplicativo, bastando procurar pelo termo “Anaconda” na internet.

Uma vez instalado o Anaconda, as bibliotecas adicionais podem ser integradas ao sistema por meio do comando no console do Anaconda:

```
conda install [nome do pacote]
```

onde o nome do pacote deve ser substituído pela biblioteca que se deseja instalar, por exemplo:

```
conda install gensim
```

Você pode ainda instalar todos os pacotes/bibliotecas de uma única vez, separando cada pacote por um simples espaço.

A instalação da plataforma Anaconda e das bibliotecas necessárias e a criação de um novo arquivo de projeto podem ser realizadas por meio dos passos mostrados na Figura 1.

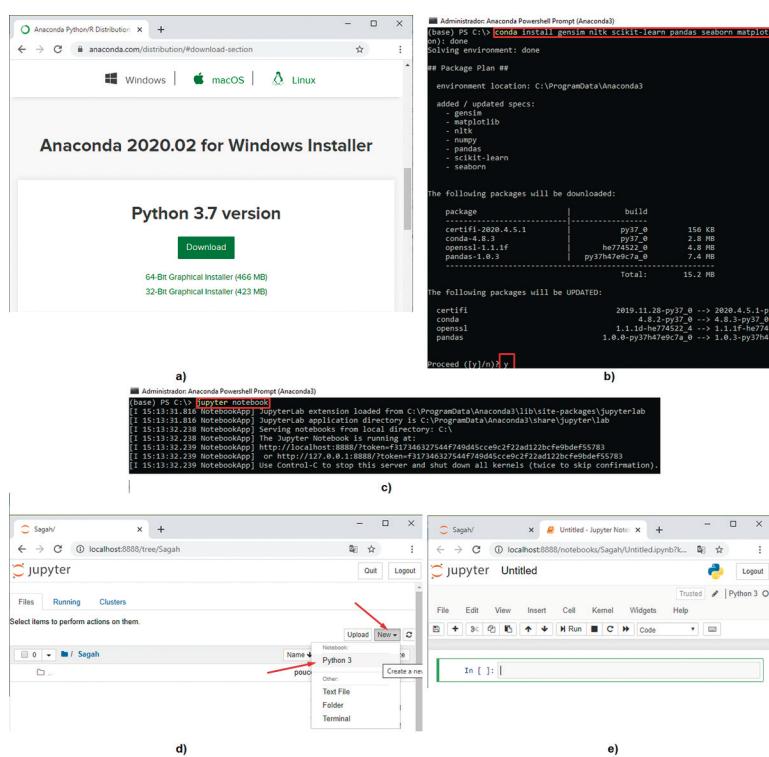


Figura 1. Sequência de passos para *download*, configuração e inicialização de um arquivo no ambiente Anaconda/Jupyter. (a) Página de *download*. (b) Console do Anaconda demonstrando comandos para instalação das bibliotecas necessárias. (c) Console do Anaconda demonstrando a inicialização do ambiente Jupyter. (d) Página inicial do Jupyter com destaque para os passos necessários para criar um arquivo Python. (e) Página de edição e simulação do arquivo criado: o retângulo no qual está presente o cursor é onde o código deve ser redigido, podendo ser testado pelo atalho Ctrl+Enter, ou Shift+Enter caso se queira criar um bloco de código adicional.

Para a biblioteca Gensim, podemos definir documento como um texto qualquer, *corpus* como uma coleção de documentos, vetor como uma representação matemática de um documento e modelo como o algoritmo que transforma representações de vetores (ŘEHŮŘEK, 2019a). Normalmente, o processamento inclui os documentos à medida que se fazem necessários, já que, muitas vezes, a quantidade de dados, se carregada para toda a memória ao mesmo tempo, poderia exceder a capacidade da maioria dos computadores convencionais (ŘEHŮŘEK, 2019a).

Antes de iniciar o processo de aprendizagem e modelagem, em geral os documentos são pré-processados, executando funções de:

- remoção de palavras de frequência excessiva, como artigos e preposições, normalmente denominadas *stop words*;
- tokenização, para indexar palavras distintas encontradas no texto e elaborar o seu vocabulário, etc.

Na biblioteca Gensim, existe uma função de pré-processamento acessível pelo módulo `utils` por `gensim.utils.simple_preprocess()`, que recebe o documento como parâmetro obrigatório e outros três parâmetros opcionais (ŘEHŮŘEK, 2019b):

- `deacc`, quando `True` remove as acentuações das palavras;
- `min_len`, que estabelece o comprimento mínimo de uma palavra;
- `max_len`, que estabelece o comprimento máximo que deve ter uma palavra. Um exemplo de uso dessa função pode ser visto a seguir.

Todos os exemplos podem ser executados no ambiente Jupyter seguindo as orientações de criação de um novo arquivo da Figura 1. Uma vez redigidos conforme os exemplos, os códigos podem ser executados a partir dos comandos `Ctrl+Enter` ou `Shift+Enter`, que crião um bloco de código após a execução.

A biblioteca `pandas`, utilizada no Exemplo 1, instalada pelo console do Anaconda conforme a Figura 1b, será útil para a criação e a manipulação de matrizes e vetores no formato de tabelas, que facilitam a visualização.

A função `simple_preprocess()` retorna uma lista de palavras de acordo com as regras especificadas nos parâmetros. No Exemplo 1, no entanto, a função é executada duas vezes, já que um laço `for` é utilizado para varrer o *corpus* e pré-processar os dois documentos nele existentes e incluí-los em uma nova lista, ou seja, o resultado esperado consiste na exibição de uma lista de documentos pré-processados, em que cada documento é representado também por uma lista, e seus membros correspondem às palavras com tamanho maior que (`min_len = 3`), transformadas para letras minúsculas e sem acento (`deacc = False`), conforme a imagem, na qual a parte superior exibe a lista como originalmente retornada pela função usada para criar uma tabela e facilitar a visualização e o entendimento dessas informações.

Os *tokens* obtidos podem ser visualizados na Figura 2, inicialmente da maneira como são retornados normalmente pelo método, seguidos da tabela criada pela biblioteca `pandas` para facilitar a sua visualização.



Exemplo

```
# Exemplo 1 - Definição de um corpus e pré-processamento
utilizando Gensim de cada um deles

# Importação das bibliotecas utilizadas
import pandas as pa
import gensim
from gensim.utils import simple_preprocess

# Definição de um corpus
corpus = [ 'Primeiro texto do capítulo', 'Segundo texto
do capítulo' ]

# Pré-processamento de cada documento do corpus
corpus_preprocessed = []
for doc in corpus:
    corpus_preprocessed.append(simple_preprocess(doc,
deacc=True,
min_len=3))

display(corpus_preprocessed)
df = pa.DataFrame(corpus_preprocessed, index=['Documento
1', 'Documento 2'],
columns=['Token 1', 'Token 2', 'Token 3'])
display(df)
```

```
[['primeiro', 'texto', 'capítulo'], ['segundo', 'texto', 'capítulo']]
```

	Token 1	Token 2	Token 3
Documento 1	primeiro	texto	capítulo
Documento 2	segundo	texto	capítulo

Figura 2. Lista e quadro que exibem os *tokens* criados para cada um dos documentos após a execução do pré-processamento com a função `simple_preprocess()` da biblioteca Gensim.

Caso deseje apenas tokenizar as palavras, o usuário pode utilizar a função `gensim.utils.tokenize()`, chamada internamente pela `simple_preprocess()`. Outras funções do módulo `utils` incluem salvar e abrir arquivos, alterar a codificação do texto e a reordenação de vetores, criação de dicionários, além de muitas outras funções que auxiliam na manipulação de *corpus* e documentos.

Na sequência, pode ser necessário criar um vocabulário, pela possibilidade de haver palavras repetidas entre os documentos, como as palavras “texto” e “capítulo” do Exemplo 1. No Exemplo 2 (como continuação do *script* do Exemplo 1), é utilizada a função `gensim.corpora.Dictionary()` para criar esse vocabulário. Para que seja possível executar o Exemplo 2, o Exemplo 1 deve antecederê-lo, já que faz uso de variáveis e bibliotecas utilizadas no Exemplo 1. Durante os seus testes, você pode, por exemplo, copiar o código do Exemplo 1 seguido do código do Exemplo 2, ou simplesmente executá-los na ordem em que estão dispostos.



Exemplo

```
# Exemplo 2 - Vocabulário criado a partir dos documentos
# tokenizados e tratados
from gensim.corpora import Dictionary as corpDict # nova
# função importada

vocab = corpDict(corpus_preprocessed)
print(vocab)
df2 = pa.DataFrame(vocab.values(), columns=['Palavra'])
display(df2)
```

O vocabulário é um tipo de arquivo próprio da biblioteca Gensim, similar e baseado nos dicionários da linguagem Python. Seu resultado pode ser visualizado na Figura 3, novamente em sua representação original e na forma de uma tabela usando o `pandas.DataFrame`.

```
Dictionary(4 unique tokens: ['capítulo', 'primeiro', 'texto', 'segundo'])
```

Palavra
0 capítulo
1 primeiro
2 texto
3 segundo

Figura 3. Vocabulário de palavras distintas entre todos os documentos pré-processados criado a partir da função `gensim.corpora.Dictionary()`.

Antes da criação de um modelo, os documentos precisam estar na forma vetorial, e não textual. Assim, cada documento será representado por um vetor em que as dimensões compreendem uma característica do documento, portanto um vetor denso, com valores diretamente relacionados a ele (ŘEHŮŘEK, 2019a).



Fique atento

Lembre-se de que, sempre que algumas dimensões dos vetores são desconhecidas, diz-se que o vetor é esparsa e todas as características faltantes são tratadas como zero (ŘEHŮŘEK, 2019a).

Como observado, a biblioteca Gensim disponibiliza, inclusive, métodos para facilitar a etapa de pré-processamento dos dados. Além dos métodos vistos, muitos outros estão disponíveis no módulo `gensim.utils` e podem ser conhecidos pela documentação na página da biblioteca. Na sequência, você poderá criar seus próprios modelos também utilizando a biblioteca Gensim.

Desenvolvimento de modelos

Apesar de a biblioteca Gensim ter vários modelos e algoritmos diferentes, como TF-IDF, word2vec, LDA, fastText, etc., a forma de utilização e os padrões de parâmetros e variáveis são muito similares para cada um deles. Aqui, daremos enfoque ao modelo word2vec.

A biblioteca implementa tanto os algoritmos de Skip-Gram quanto de CBOW (*continuous bag-of-words*) com função de *softmax* hierárquico ou amostragem negativa (ŘEHŮŘEK, 2019c).

Para iniciar e treinar um modelo word2vec, basta instanciar o modelo fornecendo os documentos como parâmetro, como no Exemplo 3, cujo código é uma sequência dos Exemplos 1 e 2, que devem ser executados antes dele para que consiga funcionar.



Exemplo

```
# Exemplo 3 - Treinamento de um modelo de word2vec
from gensim.models import Word2Vec

# Treina um modelo de Word2Vec
model = Word2Vec(corpus_preprocessed, min_count=1, size=10)

# Exibe uma tabela com os vetores
dic = {v: model[v] for v in vocab.values()}
df3 = pa.DataFrame(dic)
display(df3)
```

O parâmetro `min_count` do `Word2Vec()` permite ignorar palavras encontradas nos documentos com frequência menor que o parâmetro, enquanto o `Size` especifica a dimensão dos vetores.

O algoritmo do Exemplo 3 ainda utiliza o vocabulário anteriormente criado para servir de referência na exibição dos valores do modelo treinado na forma de uma `DataFrame` (Figura 4).

	capítulo	primeiro	texto	segundo
0	-0.014308	0.018883	-0.021456	0.002951
1	-0.028095	0.044876	0.015067	0.021536
2	-0.043606	-0.036506	0.015736	0.019706
3	-0.035784	-0.020879	0.026023	-0.023903
4	-0.031754	-0.005694	-0.045312	-0.005873
5	0.023746	0.039531	0.014962	0.049756
6	0.009919	0.013233	0.004083	-0.029429
7	-0.016283	-0.000541	-0.046770	-0.006237
8	0.014910	0.015645	0.010717	0.047384
9	0.037455	-0.007912	0.041591	0.018520

Figura 4. Representação vetorial de 10 dimensões das palavras encontradas no *corpus* para um modelo de word2vec.

Assim como especificado pelo parâmetro `size` do Exemplo 3, cada palavra é formada por um vetor de 10 dimensões. Muitos outros parâmetros podem ser modificados, como os listados a seguir.

- O parâmetro `corpus_file` substitui o uso de uma variável direta de sentenças, como o `corpus_preprocessed` do Exemplo 3, por um arquivo em que cada linha será interpretada como uma sentença. Assim, é possível economizar memória, já que o arquivo será lido conforme for necessário e outros parâmetros de restrição de memória configurados.
- O valor `sg` quando igual a 0 especifica o algoritmo de Skip-Gram; já quando igual a 1, será utilizado o algoritmo de CBOW.
- A janela máxima do contexto de uma palavra é especificada pelo parâmetro `window`.
- Também é possível optar pelo uso da função *softmax* hierárquica ou amostragem negativa, em que o parâmetro `hs` receberá 1 ou 0, respectivamente.
- O parâmetro `negative`, quando maior que 0, determina a quantidade de palavras de amostragem negativa que devem ser utilizadas no processo de treinamento.
- O valor de `alpha` determina a taxa de aprendizado.
- O parâmetro `seed` permite que se forneça um valor de base para os geradores de números aleatórios necessários durante o processo.



Saiba mais

Nenhum dos parâmetros de `gensim.models.Word2Vec()` é obrigatório, tornando-se possível instanciar um modelo sem qualquer treinamento. Ao fornecer uma variável contendo as sentenças, o nome do parâmetro pode ser ocultado desde que seja o primeiro valor fornecido. Do contrário, se utilizado um arquivo externo, será necessário explicitar o parâmetro `corpus_file='caminho do arquivo'`.

O modelo treinado pode ser salvo em um arquivo para utilização posterior ou, ainda, prosseguir com o treinamento — esta é, inclusive, uma das grandes vantagens dos modelos de *word embeddings*. Apesar do volume de dados e do tempo de treinamento requerido para modelos de linguagem natural, os algoritmos de *word embeddings* possibilitam o compartilhamento de modelos pré-treinados, usados diretamente para realizar alguma tarefa ou para ampliar o treinamento do modelo de maneira progressiva, como veremos a seguir.

2 Modelos pré-treinados

Após treinar um modelo, você pode salvá-lo em um arquivo para utilizá-lo em suas aplicações ou, ainda, para prosseguir com o aperfeiçoamento do modelo em novos treinamentos no futuro, conforme o Exemplo 4.



Exemplo

```
# Exemplo 4 - Armazenando um modelo de word2vec
import gensim
import nltk
from gensim.models import Word2Vec
from nltk.corpus import brown

nltk.download('brown')
corpus = brown.sents()
model_brown = gensim.models.Word2Vec(corpus, min_count=1)
model_brown.save('C:\\\\Sagah\\\\modelo_brown.bin')
```

No Exemplo 4, ainda é utilizado um *corpus* presente na biblioteca NLTK; caso você não tenha essa biblioteca, poderá instalá-la no console do Anaconda utilizando o comando:

```
conda install nltk
```

O Brown Corpus é um compilado de 500 textos publicados em inglês no ano de 1961 pela Universidade de Brown, no qual há pouco mais de 1 milhão de palavras diferentes, com textos de estilos diversos, mas todos escritos por nativos americanos (FRANCIS; KUCERA, 1979). Esse *corpus* tem sido frequentemente utilizado em modelos de linguagem natural pelo tamanho e pela qualidade dos textos. Ao final do Exemplo 4, o modelo gerado é salvo, pois se torna muito mais eficiente carregar o modelo em si no lugar de treinar um novo modelo de word2vec sempre que for utilizá-lo.

O nome e a pasta em que serão salvos os modelos são passados pelo primeiro parâmetro da função `save`, e não havendo qualquer restrição quanto ao nome ou à extensão do arquivo. Sempre que necessário, o modelo pode ser carregado novamente utilizando o método `load` de alguma instância de `gensim.models.Word2Vec()` ou acessando diretamente a função a partir da classe `Word2Vec`, como no Exemplo 5.



Exemplo

```
# Exemplo 5 - Carregando um modelo
from gensim.models import Word2Vec

model_carreg = Word2Vec.load('C:\\\\Sagah\\\\modelo_brown.
bin')
```

Contudo, a `load` não é utilizada apenas para aproveitar seus próprios modelos: você pode aproveitar modelos previamente treinados de outros desenvolvedores, alguns dos quais contendo quantidades imensas de dados que dificilmente um usuário comum conseguiria coletar ou filtrar e pré-processar para treinar um modelo adequado.

Modelos pré-treinados podem ser encontrados no repositório no Github dos desenvolvedores da biblioteca Gensim, caso em que os arquivos podem ser automaticamente carregados utilizando o método `gensim.downloader.download()` informando como parâmetro o nome do *corpus*.



Saiba mais

A lista de modelos pré-treinados localizados na biblioteca Gensim pode ser visualizada por meio do repositório no Github da RaRe Technologies (2018) ao buscar na internet o termo “gensim-data”.

O Exemplo 6 faz uso de um modelo `word2vec` pré-treinado chamado `word2vec-google-news-300`, um compilado de notícias contendo vetores de 300 dimensões e cerca de 3 milhões de palavras e frases, que, inteiro, tem pouco mais de 1,5 GB.



Exemplo

```
# Exemplo 6 - Carregando um modelo pré-treinado
import gensim.downloader as corpus_data

model = corpus_data.load('word2vec-google-news-300')
```

Com modelos pré-treinados, com grandes quantidades de palavras e documentos, espera-se encontrar relações mais complexas entre as palavras. Por exemplo, quando utilizado o modelo anterior para avaliar a similaridade entre as palavras `car` (“carro” em inglês) e `house` (“casa” em inglês), percebe-se que as 5 primeiras palavras retornadas para cada um seguem realmente um padrão. Para tanto, utilizou-se o algoritmo do Exemplo 7, em que um DataFrame é criado para listar na ordem da primeira até a quinta palavra mais similar às palavras `car` e `house`. Lembrando que o modelo (`model`) foi carregado no Exemplo 6, código do qual depende o Exemplo 7; portanto, deve ser executado por primeiro.



Exemplo

```
# Exemplo 7 - Palavras similares a car e house
import pandas as pa

dfC = pa.DataFrame(model.similar_by_word('car', topn=10),
columns=['Palavra', 'Similaridade'])
dfC = dfC.set_index('Palavra')
dfH = pa.DataFrame(model.similar_by_word('house', topn=10),
columns=['Palavra', 'Similaridade'])
dfH = dfH.set_index('Palavra')
display(dfC)
display(dfH)
```

No Exemplo 7, os vetores são colocados lado a lado, conforme podemos observar na Figura 5.

Similaridade-Car		Similaridade-House	
Palavra		Palavra	
vehicle	0.782110	houses	0.707239
cars	0.742383	bungalow	0.687856
SUV	0.716096	apartment	0.662900
minivan	0.690704	bedroom	0.649694
truck	0.673579	townhouse	0.638408
Car	0.667761	residence	0.619842
Ford_Focus	0.667320	mansion	0.605819
Honda_Civic	0.662685	farmhouse	0.585757
Jeep	0.651133	duplex	0.575794
pickup_truck	0.644144	apartment	0.569033

Figura 5. Dez palavras mais similares a car (à esquerda) e a house (à direita).

Para ambos os vetores (`car` e `house`), foram encontradas palavras com um contexto ou significado muito parecido com os da palavra original. Para o vetor casa (`house`), foram encontrados casas, bangalô, apartamento, quarto, etc. Já as palavras próximas a carro também apresentaram características muito similares.

Assim, uma vez criado, o modelo de representação vetorial pode ser utilizado diretamente em uma aplicação nas tarefas mais diversas, como classificação de textos, robôs de bate-papo, sumarização, tradução de idioma, etc. Bibliotecas como a Gensim e a NLTK auxiliam tanto nos processos de criação e treinamento de um modelo quanto nas fases de pré-processamento dos dados para remover palavras e outros elementos textuais indesejados. Além disso, por meio delas, você consegue compartilhar os modelos criados e usar outros modelos pré-treinados por terceiros para seus projetos, reduzindo, assim, o custo computacional e o tempo e o custo de desenvolvimento para obter soluções de alto desempenho.



Referências

- BLEI, D. M. Surveying a suíte of algorithms that offer a solution to managing large document archives. *Communication of the ACM*, [s. l.], v. 55, n. 2, p. 77–84, 2012. Disponível em: <http://www.cs.columbia.edu/~blei/papers/Blei2012.pdf>. Acesso em: 28 abr. 2020.
- FRANCIS, W. N.; KUCERA, H. *Brown corpus manual*. 1979. Disponível em: <http://korpus.uib.no/icame/manuals/BROWN/INDEX.HTM>. Acesso em: 28 abr. 2020.
- RARE TECHNOLOGIES. *Gensim-data*. [2018]. Disponível em: <https://github.com/RaRe-Technologies/gensim-data>. Acesso em: 28 abr. 2020.
- ŘEHŮŘEK, R. *Core concepts*. [2019a]. Disponível em: https://radimrehurek.com/gensim/auto_examples/core/run_core_concepts.html#sphx-glr-auto-examples-core-run-core-concepts-py. Acesso em: 28 abr. 2020.
- ŘEHŮŘEK, R. *Models.word2vec*: Word2vec embeddings. [2019c]. Disponível em: <https://radimrehurek.com/gensim/models/word2vec.html>. Acesso em: 28 abr. 2020.
- ŘEHŮŘEK, R. *Utils*: various utility functions. [2019b]. Disponível em: <https://radimrehurek.com/gensim/utils.html>. Acesso em: 28 abr. 2020.

Leitura recomendada

- SEAGATE. *Data age 2025: the digitalization of the world*. [2018]. Disponível em: <https://www.seagate.com/br/pt/our-story/data-age-2025/>. Acesso em: 28 abr. 2020.



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:



Dica do Professor

O treinamento de modelos *word2vec* pode ser custoso e demorado. Além disso, o compartilhamento e o uso podem ser limitados pelo tamanho dos arquivos. Pensando nisso, a biblioteca de Gensim permite separar os vetores criados a fim de que sejam utilizados independentemente do restante do modelo — são os denominados *KeyedVectors*.

Na Dica do Professor, você verá como a estrutura de *KeyedVectors* pode ser utilizada para compartilhar representações vetoriais de palavras, descartando as estruturas necessárias ao processo de treinamento que representam a maior parte do tamanho do arquivo. Confira.



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para accesar.

Exercícios

- 1) Parte primordial de qualquer algoritmo de aprendizado é a etapa de pré-processamento dos dados. Com o processamento de linguagem natural não é diferente. As palavras precisam se apresentar em formas similares para evitar a dupla *tokenização* de palavras iguais.

Dado o seguinte *corpus*:

`corpus = ['A primeira frase do corpus', 'A última frase do corpus']`

Assinale a opção que retornará a lista `[['primeira', 'frase', 'corpus'], ['ultima', 'frase', 'corpus']]` após o pré-processamento:

- A) Confira a alternativa a:

[Clique aqui.](#)

- B) Confira a alternativa b:

[Clique aqui.](#)

- C) Confira a alternativa c:

[Clique aqui.](#)

- D) Confira a alternativa d:

[Clique aqui.](#)

E) Confira a alternativa e:

[Clique aqui.](#)

2) O pré-processamento em algoritmos de processamento da linguagem natural envolve, principalmente, a *tokenização*, a remoção de *stopwords* e a construção de um vocabulário.

Dado o seguinte *corpus*:

`corpus = ['O algoritmo está bom'], 'No entanto poderia ser melhor']`

Assinale a opção que apresenta um algoritmo capaz de *tokenizar* e criar o vocabulário, eliminando acentos e palavras menores que 3 caracteres:

A) Confira a alternativa a:

[Clique aqui.](#)

B) Confira a alternativa b:

[Clique aqui.](#)

C) Confira a alternativa c:

[Clique aqui.](#)

D) Confira a alternativa d:

Clique aqui.

- E) Confira a alternativa e:

Clique aqui.

- 3) A biblioteca Gensim, além de ter ferramentas de manipulação geral de textos e documentos, como as ferramentas de pré-processamento, também tem modelos e diferentes algoritmos de processamento de linguagem natural, como o *word2vec*.

Sobre a biblioteca Gensim, assinale a opção em que o algoritmo cria um modelo de *word2vec* de dimensão 10, ignorando as palavras com frequência menor que 5:

- A) Confira a alternativa a:

Clique aqui.

- B) Confira a alternativa b:

Clique aqui.

- C) Confira a alternativa c:

[Clique aqui.](#)

D) Confira a alternativa d:

[Clique aqui.](#)

E) Confira a alternativa e:

[Clique aqui.](#)

4) Criar um modelo de *word embeddings* pode demandar muito tempo de processamento. No entanto, os algoritmos atualmente são capazes de aproveitar modelos pré-treinados para acumular progressivamente mais informações.

O algoritmo a seguir utiliza um *corpus* simples para treinar um modelo de *word2vec* e salvá-lo:

```
from gensim.models import Word2Vec
```

```
corpus = ['Exercícios ajudarão a fixar o conteúdo, 'A partir de 2013 o word2vec conquistou o mundo']
```

```
model = gensim.models.Word2Vec(corpus, min_count=3)
model.save('C:\\\\Sagah\\\\modelo.bin')
```

Dos algoritmos a seguir, qual aproveita o treinamento anterior desse modelo e amplia o treinamento, utilizando um *brown corpus* da biblioteca NLTK?

A) Confira a alternativa a:

[Clique aqui.](#)

B) Confira a alternativa b:

[Clique aqui.](#)

C) Confira a alternativa c:

[Clique aqui.](#)

D) Confira a alternativa d:

[Clique aqui.](#)

E) Confira a alternativa e:

[Clique aqui.](#)

5) Modelos pré-treinados permitem que boas representações vetoriais sejam aproveitadas por outros usuários que não teriam acesso a quantidades tão grandes de informação ou não teriam computador rápido o suficiente para executá-las.

A partir do modelo pré-treinado em anexo (o arquivo deve ser aberto por um algoritmo usando a função `word2vec` da biblioteca Gensim), determine qual é a palavra mais próxima da palavra *past* e assinale a opção correta:

[Clique aqui](#)

A) 1958.

B) *Future.*

C) *Last.*

D) *Dark.*

E) *Present.*

Na prática

O uso de modelos pré-treinados garante acessibilidade a modelos de alta qualidade e agilidade no desenvolvimento de aplicações. Você pode encontrar modelos envolvendo temas específicos, como é o caso do *word2vec-google-news-300*, treinado com notícias, todas no idioma inglês. É de se esperar que esse modelo seja mais eficiente quando utilizado em contextos similares. Além disso, se fosse desenvolvido do início, o tamanho do *corpus* utilizado demandaria muita memória e tempo de processamento.

A partir de um caso hipotético, veja, Na Prática, como você pode utilizar um modelo pré-treinado de *word2vec* para classificar notícias.

UTILIZANDO UM MODELO PRÉ-TREINADO DE WORD2VEC

Samuel estava desenvolvendo um aplicativo de notícias que varre a Internet em busca das últimas publicações em sites internacionais. Esse aplicativo vai exibir as notícias separadas em quatro seções: *health* (saúde), *economy* (economia), *politics* (política) e *sports* (esportes).



No entanto, ele se deparou com o seguinte problema:

COMO CLASSIFICAR ESSAS NOTÍCIAS DE MANEIRA AUTOMÁTICA?

Por sorte, Samuel é um experiente programador e já trabalhou com processamento de linguagem natural. Ele se recordou de que muitos modelos pré-treinados de representação vetorial de palavras estão disponíveis para a língua inglesa. Após uma pesquisa, descobriu um modelo cujo contexto é muito similar ao seu problema: o modelo *word2vec* do Google Notícias.



Para testar a aplicabilidade desse modelo no seu aplicativo, Samuel decidiu criar um protótipo na plataforma *Jupyter*, com o auxílio da biblioteca *Gensim*, que já dispõe do modelo *word2vec* de que ele necessita.

```
import gensim.downloader as corpus_data  
model = corpus_data.load('word2vec-google-news-300')
```

O modelo foi baixado por meio da função *gensim.downloader.load(name)*, informando o nome do conjunto que ele desejava carregar. O arquivo foi baixado e carregado para a variável da aplicação.

Samuel, então, criou um vetor contendo cada um dos rótulos que pretendia dar às notícias, utilizando esses nomes para verificar as 50 palavras mais próximas deles. O conjunto de 50 palavras de cada categoria foi utilizado para comparar com as palavras das notícias.

```
classes = ['health', 'economy', 'politic', 'sport']  
  
palavras_pontuadas = []  
for c in classes:  
    similares = model.similar_by_word(c, topn=50)  
    vetor = []  
    for s in similares:  
        vetor.append(s[0])  
    palavras_pontuadas.append(vetor)
```

Em seguida, ele utilizou a biblioteca *NLTK*. Essa biblioteca é usada para *tokenizar* as palavras de cada notícia. Cada notícia é carregada a partir de um arquivo, que é lido e tem todas as letras alteradas para minúsculas. O texto resultante é então *tokenizado*, e são removidas as palavras de alta frequência, denominadas *stopwords*.

```
import nltk  
  
nltk.download('stopwords')  
  
arq = open('C:\Sagah\lnoticia4.txt','r', errors='ignore')  
texto = arq.read()  
texto = texto.lower()  
  
tokens = nltk.word_tokenize(texto)  
  
from nltk.corpus import stopwords  
  
stops = stopwords.words('english')  
filtered_tokens = [w for w in tokens if not w in stops]
```

Assim, as palavras restantes encontradas na notícia são comparadas com cada uma das 50 palavras de cada categoria anteriormente selecionadas por meio do modelo do *word2vec* pré-treinado que fora carregado. Um vetor acumula o resultado de cada comparação, relacionando ainda as palavras comparadas e a classe a que pertencem.

```
comparativo = []  
for w in filtered_tokens:  
    for palavras, c in zip(palavras_pontuadas, classes):  
        for p in palavras:  
            try:  
                maximo = model.similarity(w, p)  
                comparativo.append([w, p, c, maximo])  
            except:  
                pass
```

Com o auxílio da biblioteca *Pandas*, um *DataFrame* foi criado para facilitar a manipulação do vetor de comparações. Esse *DataFrame* é, então, ordenado, com a coluna de *Pontuação*, e as 500 maiores pontuações servem de base para a classificação. Entre as informações retornadas pelo método *describe()* está a frequência da classe de maior ocorrência, cujo nome pode ser acessado pelo índice *'top'*.

```
import pandas as pa  
  
df = pa.DataFrame(comparativo, columns=['Notícia', 'Modelo', 'Classe', 'Pontuação'])  
df_sorted = df.sort_values('Pontuação', ascending=False)  
df_sorted['Classe'].head(500).describe()['top']
```

A partir disso, restou a Samuel testar esse algoritmo, a fim de validar a sua efetividade, e proceder com os ajustes que permitam aumentar seu desempenho.



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para accesar.

Saiba mais

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

Apoio à mediação pedagógica em um "debate de teses" utilizando técnicas de processamento de texto

Neste artigo, a biblioteca Gensim foi utilizada para auxiliar na mediação pedagógica com suas técnicas de processamento de texto. O trabalho teve por objetivo encontrar uma forma de auxiliar o professor de educação a distância no acompanhamento das atividades. Confira.



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para accesar.

Repositório de *word embeddings* do NILC

No *link* a seguir, confira o repositório de *word embeddings* da Universidade de São Paulo (USP) para a língua portuguesa.



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para accesar.

Criando sistemas de processamento de linguagem natural na prática

Nesta palestra, o engenheiro de *software* especializado em inteligência artificial aplicada ao processamento de linguagem natural William Colen faz uma abordagem geral sobre a aplicação de técnicas de processamento de linguagem natural. Aproveite.



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