# Modelagem de tópicos com Gensim (Python)

***por Selva Prabhakaran |***

*A modelagem de tópicos é uma técnica para extrair os tópicos ocultos de grandes volumes de texto. Alocação de Dirichlet Latente (LDA) é um algoritmo popular para modelagem de tópicos com excelentes implementações no pacote Gensim do Python. O desafio, no entanto, é como extrair boa qualidade de tópicos claros, segregados e significativos . Isso depende muito da qualidade do pré-processamento de texto e da estratégia de encontrar o número ideal de tópicos. Este tutorial tenta solucionar esses dois problemas.*

1.

## Conteúdo

# 1. Introdução

Uma das principais aplicações do processamento de linguagem natural é extrair automaticamente quais tópicos as pessoas estão discutindo em grandes volumes de texto. Alguns exemplos de texto grande podem ser feeds de mídias sociais, avaliações de clientes de hotéis, filmes, etc, feedbacks de usuários, notícias, e-mails de reclamações de clientes etc.

Saber do que as pessoas estão falando e entender seus problemas e opiniões é altamente valioso para empresas, administradores e campanhas políticas. E é realmente difícil ler manualmente volumes tão grandes e compilar os tópicos.

Assim, é necessário um algoritmo automatizado que possa ler os documentos de texto e gerar automaticamente os tópicos discutidos.

Neste tutorial, pegaremos um exemplo real do conjunto de dados '20 Newsgroups 'e usaremos o LDA para extrair os tópicos discutidos naturalmente.

Utilizarei o pacote Alocação de Dirichlet Latente (LDA) do pacote Gensim junto com a implementação do Mallet (via Gensim). Mallet tem uma implementação eficiente da LDA. Sabe-se que é mais rápido e oferece uma melhor segregação de tópicos.

Também extrairemos o volume e a porcentagem de contribuição de cada tópico para ter uma idéia da importância de um tópico.

Vamos começar!

Modelagem de tópicos com Gensim em Python. Foto de Jeremy Bishop.

# 2. Pré-requisitos - Faça o download do nltk stopwords e do modelo spacy

Vamos

precisar `stopwords` do modelo NLTK e da Spacy `en` para pré-processamento de texto. Mais tarde, usaremos o modelo spacy para a lematização.

A lematização nada mais é do que converter uma palavra em sua raiz. Por exemplo: o lema da palavra 'máquinas' é 'máquina'. Da mesma forma, 'andando' -> 'andar', 'ratos' -> 'mouse' e assim por diante.

```
# Run in python console
import nltk; nltk.download('stopwords')

# Run in terminal or command prompt
python3 -m spacy download en
```

# 3. Pacotes de Importação

Os pacotes básicos usados neste tutorial são `re`, `gensim`, `spacy` e `pyLDAvis`. Além disto, também vai usar `matplotlib`, `numpy` e `pandas` para manipulação de dados e visualização. Vamos importá-los.

```
 import re
import numpy as np
import pandas as pd
from pprint import pprint

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# spacy for lemmatization
import spacy

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt
%matplotlib inline

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERROR)

import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)
```

A

# 4. O que a LDA faz?

abordagem da LDA para modelagem de tópicos é considerar cada documento como uma coleção de tópicos em uma certa proporção. E cada tópico como uma coleção de palavras-chave, novamente, em uma certa proporção.

Depois de fornecer ao algoritmo o número de tópicos, tudo o que é necessário é reorganizar a distribuição de tópicos na distribuição de documentos e palavras-chave nos tópicos para obter uma boa composição da distribuição de palavras-chave de tópico.

Quando digo tópico, o que é realmente e como é representado?

Um tópico nada mais é do que uma coleção de palavras-chave dominantes que são representantes típicos. Apenas olhando as palavras-chave, você pode identificar sobre o que é o tópico.

A seguir, apresentamos os principais fatores para obter bons tópicos de segregação:

1. A qualidade do processamento de texto.
2. A variedade de tópicos sobre os quais o texto fala.
3. A escolha do algoritmo de modelagem de tópicos.
4. O número de tópicos alimentados ao algoritmo.
5. Os parâmetros de ajuste dos algoritmos.

# 5. Prepare palavras de parada

Já baixamos as palavras de parada. Vamos importá-los e disponibilizá-lo no `stop_words` .

```
# NLTK Stop words
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])
```

# 6. Importar dados de grupos de notícias

Usaremos o conjunto de dados de 20 grupos de notícias para este exercício. Esta versão do conjunto de dados contém cerca de 11k postagens de grupos de notícias de 20 tópicos diferentes. Está disponível como newsgroups.json .

Isso é importado usando e o conjunto de dados resultante possui 3 colunas, como mostrado. pandas.read_json

```
# Import Dataset
df = pd.read_json('https://raw.githubusercontent.com/selva86/datasets/master/newsgroups.json')
print(df.target_names.unique())
df.head()
```

```
['rec.autos' 'comp.sys.mac.hardware' 'rec.motorcycles' 'misc.forsale'
 'comp.os.ms-windows.misc' 'alt.atheism' 'comp.graphics'
 'rec.sport.baseball' 'rec.sport.hockey' 'sci.electronics' 'sci.space'
 'talk.politics.misc' 'sci.med' 'talk.politics.mideast'
 'soc.religion.christian' 'comp.windows.x' 'comp.sys.ibm.pc.hardware'
 'talk.politics.guns' 'talk.religion.misc' 'sci.crypt']
```

| content | target | target_names |
|---|---|---|
| From: lerxst@wam.umd.edu (where's my thing)\nSubject: WHAT car is this!?\nNntp-Posting-Host: rac... | 7 | rec.autos |
| From: guykuo@carson.u.washington.edu (Guy Kuo)\nSubject: SI Clock Poll - Final Call\nSummary: Fi... | 4 | comp.sys.mac.hardware |
| From: irwin@cmptrc.lonestar.org (Irwin Arnstein)\nSubject: Re: Recommendation on Duc\nSummary: W... | 8 | rec.motorcycles |
| From: tchen@magnus.acs.ohio-state.edu (Tsung-Kun Chen)\nSubject: ** Software forsale (lots) **\n... | 6 | misc.forsale |
| From: dabl2@nlm.nih.gov (Don A.B. Lindbergh)\nSubject: Diamond SS24X, Win 3.1, Mouse cursor\nOrg... | 2 | comp.os.ms-windows.misc |
| From: a207706@moe.dseg.ti.com (Robert Loper)\nSubject: Re: SHO and SC\nNntp-Posting-Host: sun278... | 7 | rec.autos |
| From: kimman@magnus.acs.ohio-state.edu (Kim Richard Man)\nSubject: SyQuest 44M cartrifge FORSALE... | 6 | misc.forsale |
| From: kwilson@casbah.acns.nwu.edu (Kirtley Wilson)\nSubject: Mirosoft Office Package\nArticle-I.... | 2 | comp.os.ms-windows.misc |
| Subject: Re: Don't more innocents die without the death penalty?\nFrom: bobbe@vice.ICO.TEK.COM (... | 0 | alt.atheism |
| From: livesey@solntze.wpd.sgi.com (Jon Livesey)\nSubject: Re: Genocide is Caused by Atheism\nOrg... | 0 | alt.atheism |
| From: dls@aeg.dsto.gov.au (David Silver)\nSubject: Re: Fractal Generation of Clouds\nOrganizatio... | 1 | comp.graphics |
| Subject: Re: Mike Francesa's 1993 Predictions\nFrom: gajarsky@pilot.njin.net (Bob Gajarsky - Hob... | 9 | rec.sport.baseball |
| From: jet@netcom.Netcom.COM (J. Eric Townsend)\nSubject: Re: Insurance and lotsa points...\nIn-R... | 8 | rec.motorcycles |
| From: gld@cunixb.cc.columbia.edu (Gary L Dare)\nSubject: Re: ABC coverage\nNntp-Posting-Host: cu... | 10 | rec.sport.hockey |
| From: sehari@iastate.edu (Babak Sehari)\nSubject: Re: How to the disks copy protected.\nOriginat... | 12 | sci.electronics |

20 conjunto de dados de grupos de notícias

# 7. Remova e-mails e caracteres de nova linha

Como você pode ver, existem muitos e-mails, novas linhas e espaços extras que são bastante perturbadores. Vamos nos livrar deles usando expressões regulares .

```
 # Convert to list
data = df.content.values.tolist()

# Remove Emails
data = [re.sub('\S*@\S*\s?', '', sent) for sent in data]

# Remove new line characters
data = [re.sub('\s+', ' ', sent) for sent in data]

# Remove distracting single quotes
data = [re.sub("\'", "", sent) for sent in data]

pprint(data[:1])
```

```
['From: (wheres my thing) Subject: WHAT car is this!? Nntp-Posting-Host: '
 'rac3.wam.umd.edu Organization: University of Maryland, College Park Lines: '
 '15 I was wondering if anyone out there could enlighten me on this car I saw '
 'the other day. It was a 2-door sports car, looked to be from the late 60s/ '
 'early 70s. It was called a Bricklin. The doors were really small. In '
 'addition, the front bumper was separate from the rest of the body. This is '
 'all I know.  (..truncated..)]
```

Depois de remover os e-mails e os espaços extras, o texto ainda parece confuso. Não está pronto para o LDA consumir. Você precisa dividir cada frase em uma lista de palavras por meio de tokenização, enquanto limpa todo o texto confuso no processo.

Gensim simple_preprocess é ótimo para isso.

Vamos

# 8. Tokenize palavras e texto de limpeza

tokenizar cada frase em uma lista de palavras, removendo pontuações e caracteres desnecessários por completo.

Gensim é ótimo para isso. Além disso, configurei para remover as pontuações. simple_preprocess() deacc=True

```
def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))  # deacc=True removes punc

data_words = list(sent_to_words(data))

print(data_words[:1])
```

```
[['from', 'wheres', 'my', 'thing', 'subject', 'what', 'car', 'is', 'this', 'nntp', 'posting', 'hos
```

# 9. Criando Modelos Bigram e Trigram

Bigrams são duas palavras que ocorrem frequentemente juntas no documento. Trigramas são três palavras que ocorrem com frequência.

Alguns exemplos em nosso exemplo são: 'front_bumper', 'oil_leak', 'maryland_college_park' etc.

O Phrases modelo da Gensim pode criar e implementar bigrams, trigramas, quadrogramas e muito mais. Os dois argumentos importantes para Phrases são min_count e threshold . Quanto mais altos os valores desses parâmetros, mais difícil é a combinação das palavras com os bigrams.

```
# Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100) # higher threshold fewer phr
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)

# See trigram example
print(trigram_mod[bigram_mod[data_words[0]]])
```

```
['from', 'wheres', 'my', 'thing', 'subject', 'what', 'car', 'is', 'this', 'nntp_posting_host', 'ra
```

# 10. Remova palavras irrelevantes, faça bigrams e lematize

O modelo de bigrams está pronto. Vamos definir as funções para remover as palavras irrelevantes, criar bigrams e lematização e chamá-las sequencialmente.

```
# Define functions for stopwords, bigrams, trigrams and lemmatization
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for doc in te

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

Vamos chamar as funções em ordem.

```
# Remove Stop Words
data_words_nostops = remove_stopwords(data_words)

# Form Bigrams
data_words_bigrams = make_bigrams(data_words_nostops)

# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
# python3 -m spacy download en
nlp = spacy.load('en', disable=['parser', 'ner'])

# Do lemmatization keeping only noun, adj, vb, adv
data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

print(data_lemmatized[:1])
```

```
[['where', 's', 'thing', 'car', 'nntp_post', 'host', 'rac_wam', 'umd', 'organization', 'university
```

# 11. Crie o dicionário e o corpus necessários para a modelagem de tópicos

As duas entradas principais para o modelo de tópico LDA são o dicionário ( id2word ) e o corpus. Vamos criá-los.

```
# Create Dictionary
id2word = corpora.Dictionary(data_lemmatized)

# Create Corpus
texts = data_lemmatized

# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]

# View
print(corpus[:1])
```

```
[[(0, 1), (1, 2), (2, 1), (3, 1), (4, 1), (5, 1), (6, 5), (7, 1), (8, 1), (9, 2), (10, 1), (11, 1)
```

O Gensim cria um ID exclusivo para cada palavra no documento. O corpus produzido mostrado acima é um mapeamento de (word_id, word_frequency).

Por exemplo, (0, 1) acima implica que a palavra id 0 ocorra uma vez no primeiro documento. Da mesma forma, a palavra id 1 ocorre duas vezes e assim por diante.

Isso é usado como entrada pelo modelo LDA.

Se você deseja ver a que palavra um determinado ID corresponde, passe-o como uma chave para o dicionário.

```
id2word[0]
```

```
'addition'
```

Ou, você pode ver uma forma legível por humanos do próprio corpus.

```
# Human readable format of corpus (term-frequency)
[[(id2word[id], freq) for id, freq in cp] for cp in corpus[:1]]
```

```
[[('addition', 1),
  ('anyone', 2),
  ('body', 1),
  ('bricklin', 1),
  ('bring', 1),
  ('call', 1),
  ('car', 5),
  ('could', 1),
  ('day', 1),
  ('door', 2),
  ('early', 1),
  ('engine', 1),
  ('enlighten', 1),
  ('front_bumper', 1),
  ('maryland_college', 1),
  (..truncated..)]]
```

Tudo bem, sem se desviar ainda mais, vamos voltar à pista com o próximo passo: Construindo o modelo de tópico.

# 12. Construindo o modelo de tópico

Temos tudo o necessário para treinar o modelo LDA. Além do corpus e do dicionário, você também precisa fornecer o número de tópicos.

Além disso, alpha e eta são hiperparâmetros que afetam a escassez dos tópicos. De acordo com a documentação do Gensim, os dois padrões são 1.0 / num_topics anteriores.

chunksize é o número de documentos a serem usados em cada parte do treinamento. update_every determina com que frequência os parâmetros do modelo devem ser atualizados e passes é o número total de passes de treinamento.

```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=20,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
```

# 13. Veja os tópicos no modelo LDA

O modelo LDA acima é construído com 20 tópicos diferentes, onde cada tópico é uma combinação de palavras-chave e cada palavra-chave contribui com uma certa ponderação para o tópico.

Você pode ver as palavras-chave de cada tópico e a ponderação (importância) de cada palavra-chave, conforme mostrado a seguir. lda_model.print_topics()

```
# Print the Keyword in the 10 topics
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0,
  '0.016*"car" + 0.014*"power" + 0.010*"light" + 0.009*"drive" + 0.007*"mount" '
  '+ 0.007*"controller" + 0.007*"cool" + 0.007*"engine" + 0.007*"back" + '
  '0.006*"turn"'),
 (1,
  '0.072*"line" + 0.066*"organization" + 0.037*"write" + 0.032*"article" + '
  '0.028*"university" + 0.027*"nntp_post" + 0.026*"host" + 0.016*"reply" + '
  '0.014*"get" + 0.013*"thank"'),
 (2,
  '0.017*"patient" + 0.011*"study" + 0.010*"slave" + 0.009*"wing" + '
  '0.009*"disease" + 0.008*"food" + 0.008*"eat" + 0.008*"pain" + '
  '0.007*"treatment" + 0.007*"syndrome"'),
 (3,
  '0.013*"key" + 0.009*"use" + 0.009*"may" + 0.007*"public" + 0.007*"system" + '
  '0.007*"order" + 0.007*"government" + 0.006*"state" + 0.006*"provide" + '
  '0.006*"law"'),
 (4,
  '0.568*"ax" + 0.007*"rlk" + 0.005*"tufts_university" + 0.004*"ei" + '
  '0.004*"m" + 0.004*"vesa" + 0.004*"differential" + 0.004*"chz" + 0.004*"lk" '
  '+ 0.003*"weekly"'),
 (5,
  '0.029*"player" + 0.015*"master" + 0.015*"steven" + 0.009*"tor" + '
  '0.009*"van" + 0.008*"king" + 0.008*"scripture" + 0.007*"cal" + '
  '0.007*"helmet" + 0.007*"det"'),
 (6,
  '0.028*"system" + 0.020*"problem" + 0.019*"run" + 0.018*"use" + 0.016*"work" '
  '+ 0.015*"do" + 0.013*"window" + 0.013*"driver" + 0.013*"bit" + 0.012*"set"'),
 (7,
  '0.017*"israel" + 0.011*"israeli" + 0.010*"war" + 0.010*"armenian" + '
  '0.008*"kill" + 0.008*"soldier" + 0.008*"attack" + 0.008*"government" + '
  '0.007*"lebanese" + 0.007*"greek"'),
 (8,
  '0.018*"money" + 0.018*"year" + 0.016*"pay" + 0.012*"car" + 0.010*"drug" + '
  '0.010*"president" + 0.009*"rate" + 0.008*"face" + 0.007*"license" + '
  '0.007*"american"'),
 (9,
  '0.028*"god" + 0.020*"evidence" + 0.018*"christian" + 0.012*"believe" + '
  '0.012*"reason" + 0.011*"faith" + 0.009*"exist" + 0.008*"bible" + '
  '0.008*"religion" + 0.007*"claim"'),
 (10,
  '0.030*"physical" + 0.028*"science" + 0.012*"direct" + 0.012*"st" + '
  '0.012*"scientific" + 0.009*"waste" + 0.009*"jeff" + 0.008*"cub" + '
  '0.008*"brown" + 0.008*"msg"'),
 (11,
```

```
 '0.016*"wire" + 0.011*"keyboard" + 0.011*"md" + 0.009*"pm" + 0.008*"air" + '
 '0.008*"input" + 0.008*"fbi" + 0.007*"listen" + 0.007*"tube" + '
 '0.007*"koresh"'),
(12,
 '0.016*"motif" + 0.014*"serial_number" + 0.013*"son" + 0.013*"father" + '
 '0.011*"choose" + 0.009*"server" + 0.009*"event" + 0.009*"value" + '
 '0.007*"collin" + 0.007*"prediction"'),
(13,
 '0.098*"_" + 0.043*"max" + 0.015*"dn" + 0.011*"cx" + 0.009*"eeg" + '
 '0.008*"gateway" + 0.008*"c" + 0.005*"mu" + 0.005*"mr" + 0.005*"eg"'),
(14,
 '0.024*"book" + 0.009*"april" + 0.007*"group" + 0.007*"page" + '
 '0.007*"new_york" + 0.007*"iran" + 0.006*"united_state" + 0.006*"author" + '
 '0.006*"include" + 0.006*"club"'),
(15,
 '0.020*"would" + 0.017*"say" + 0.016*"people" + 0.016*"think" + 0.014*"make" '
 '+ 0.014*"go" + 0.013*"know" + 0.012*"see" + 0.011*"time" + 0.011*"get"'),
(16,
 '0.026*"file" + 0.017*"program" + 0.012*"window" + 0.012*"version" + '
 '0.011*"entry" + 0.011*"software" + 0.011*"image" + 0.011*"color" + '
 '0.010*"source" + 0.010*"available"'),
(17,
 '0.027*"game" + 0.027*"team" + 0.020*"year" + 0.017*"play" + 0.016*"win" + '
 '0.010*"good" + 0.009*"season" + 0.008*"fan" + 0.007*"run" + 0.007*"score"'),
(18,
 '0.036*"drive" + 0.024*"card" + 0.020*"mac" + 0.017*"sale" + 0.014*"cpu" + '
 '0.010*"price" + 0.010*"disk" + 0.010*"board" + 0.010*"pin" + 0.010*"chip"'),
(19,
 '0.030*"space" + 0.010*"sphere" + 0.010*"earth" + 0.009*"item" + '
 '0.008*"launch" + 0.007*"moon" + 0.007*"mission" + 0.007*"nasa" + '
 '0.007*"orbit" + 0.006*"research"')]
```

Como interpretar isso?

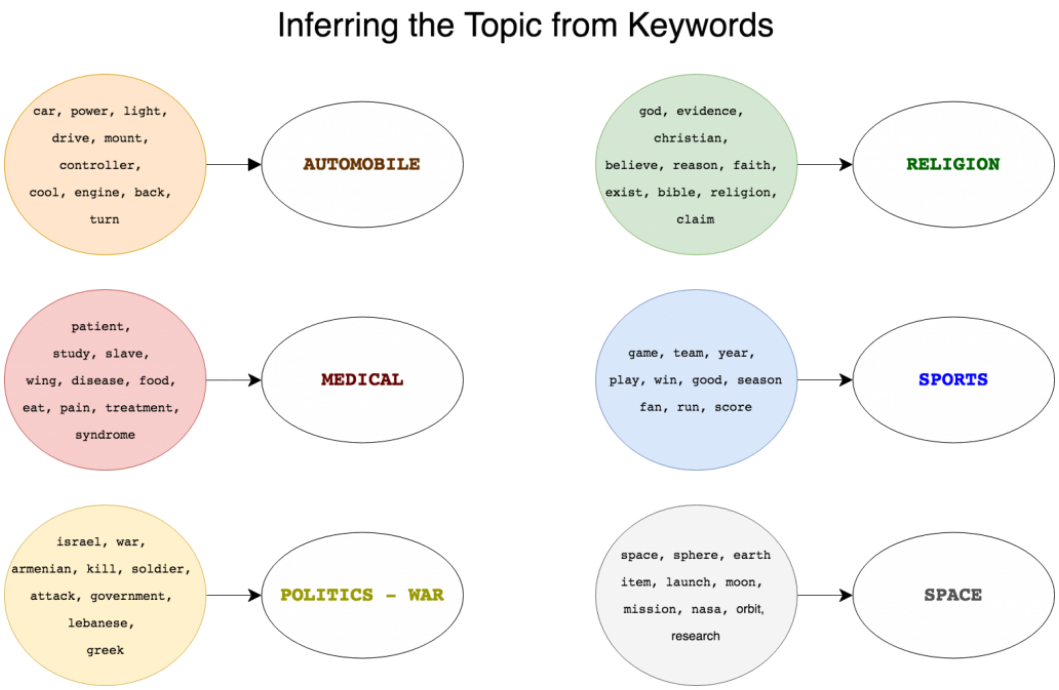
O tópico 0 é representado como _0.016 *"carro"* + *0,014* "potência" + 0,010 *"luz"* + *0,009* "movimentação" + 0,007 *"montagem"* + *0,007* " *montagem"* + *0,007*"controlador" + 0,007 *"legal"* + *0,007* "motor" + 0,007 *" voltar "*+ *'0,006* " virar ".

Significa que as 10 principais palavras-chave que contribuem para este tópico são: 'car', 'power', 'light' etc. e assim por diante, e o peso de 'car' no tópico 0 é 0,016.

Os pesos refletem a importância de uma palavra-chave para esse tópico.

Observando essas palavras-chave, você consegue adivinhar qual seria esse tópico? Você pode resumir que são "carros" ou "automóveis".

Da mesma forma, você pode examinar as palavras-chave restantes do tópico e julgar qual é o tópico?




Inferindo tópico de palavras-chave

A

# 14. Perplexidade do Modelo de Computação e Pontuação de Coerência

perplexidade do modelo e a coerência do tópico fornecem uma medida conveniente para julgar quão bom é um determinado modelo de tópico. Na minha experiência, a pontuação de coerência de tópicos, em particular, tem sido mais útil.

```
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the model is. lo

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized, dictionary=id2word, co
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

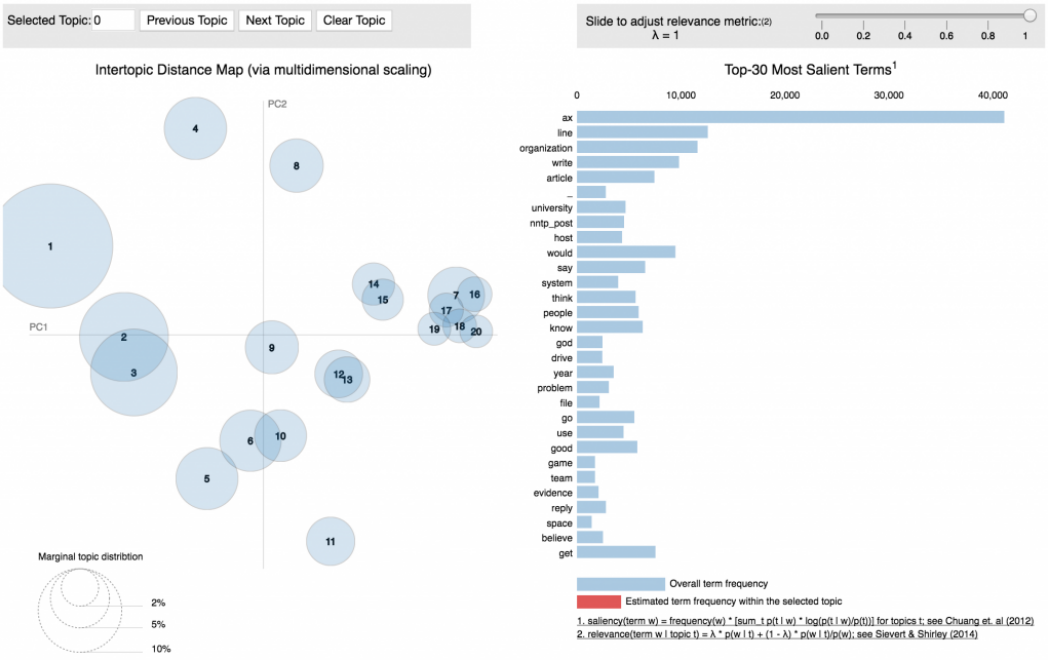
```
Perplexity:  -8.86067503009

Coherence Score:  0.532947587081
```

Lá você tem uma pontuação de coerência de 0,53.

# 15. Visualize os tópicos-palavras-chave

Agora que o modelo de LDA foi criado, a próxima etapa é examinar os tópicos produzidos e as palavras-chave associadas. Não existe ferramenta melhor do que o gráfico interativo do pacote pyLDAvis e foi projetado para funcionar bem com os notebooks jupyter.

```
# Visualize the topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```




Saída pyLDAvis

Então, como inferir a saída do pyLDAvis?

Cada bolha no gráfico do lado esquerdo representa um tópico. Quanto maior a bolha, mais prevalente é esse tópico.

Um bom modelo de tópico terá bolhas bastante grandes e não sobrepostas espalhadas pelo gráfico, em vez de serem agrupadas em um quadrante.

Um modelo com muitos tópicos geralmente possui muitas sobreposições, pequenas bolhas agrupadas em uma região do gráfico.

Tudo bem, se você mover o cursor sobre uma das bolhas, as palavras e as barras do lado direito serão atualizadas. Essas palavras são as principais palavras-chave que formam o tópico selecionado.

Criamos com sucesso um modelo de tópico atraente.

Dado o nosso conhecimento prévio do número de tópicos naturais no documento, encontrar o melhor modelo era bastante simples.

Em seguida, aprimoraremos esse modelo usando a versão do algoritmo LDA de Mallet e, em seguida, focaremos em como chegar ao número ideal de tópicos, dado um grande corpus de texto.

# 16. Construindo o modelo de malho LDA

Até agora, você viu a versão incorporada de Gensim do algoritmo LDA. A versão de Mallet, no entanto, geralmente oferece uma melhor qualidade de tópicos.

O Gensim fornece um invólucro para implementar o LDA do Mallet a partir do próprio Gensim. Você só precisa baixar o arquivo zip, descompactá-lo e fornecer o caminho para o malho no diretório descompactado . Veja como eu fiz isso abaixo. gensim.models.wrappers.LdaMallet

```
# Download File: http://mallet.cs.umass.edu/dist/mallet-2.0.8.zip
mallet_path = 'path/to/mallet-2.0.8/bin/mallet' # update this path
ldamallet = gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=20, id2word=id2
```

```
# Show Topics
pprint(ldamallet.show_topics(formatted=False))

# Compute Coherence Score
coherence_model_ldamallet = CoherenceModel(model=ldamallet, texts=data_lemmatized, dictionary=id2wo
coherence_ldamallet = coherence_model_ldamallet.get_coherence()
print('\nCoherence Score: ', coherence_ldamallet)
```

```
[(13,
  [('god', 0.022175351915726671),
   ('christian', 0.017560827817656381),
   ('people', 0.0088794630371958616),
   ('bible', 0.008215251235200895),
   ('word', 0.0077491376899412696),
   ('church', 0.0074112053696280414),
   ('religion', 0.0071198844038407759),
   ('man', 0.0067936049221590383),
   ('faith', 0.0067469935676330757),
   ('love', 0.0064556726018458093)]),
 (1,
  [('organization', 0.10977647987951586),
   ('line', 0.10182379194445974),
   ('write', 0.097397469098389255),
   ('article', 0.082483883409554246),
   ('nntp_post', 0.079894209047330425),
   ('host', 0.069737542931658306),
   ('university', 0.066303010266865026),
   ('reply', 0.02255404338163719),
   ('distribution_world', 0.014362591143681011),
   ('usa', 0.010928058478887726)]),
 (8,
  [('file', 0.02816690014008405),
   ('line', 0.021396171035954908),
   ('problem', 0.013508104862917751),
   ('program', 0.013157894736842105),
   ('read', 0.012607564538723234),
   ('follow', 0.01110666399839904),
   ('number', 0.011056633980388232),
   ('set', 0.010522980454939631),
   ('error', 0.010172770328863986),
   ('write', 0.010039356947501835)]),
 (7,
  [('include', 0.0091670556506405262),
   ('information', 0.0088169700741662776),
   ('national', 0.0085576474249260924),
   ('year', 0.0077667133447435295),
   ('report', 0.0070406099268710129),
   ('university', 0.0070406099268710129),
   ('book', 0.0068979824697889113),
   ('program', 0.0065219646283906432),
   ('group', 0.0058866241377521916),
   ('service', 0.0057180644157460714)]),
```

```
 (..truncated..)]

Coherence Score:  0.632431683088
```

Apenas alterando o algoritmo LDA, aumentamos a pontuação de coerência de 0,53 para 0,63. Não é ruim!

Minha

# 17. Como encontrar o número ideal de tópicos para a LDA?

abordagem para encontrar o número ideal de tópicos é construir muitos modelos de LDA com diferentes valores do número de tópicos (k) e escolher o que fornece o maior valor de coerência.

Escolher um 'k' que marca o fim de um rápido crescimento da coerência de tópicos geralmente oferece tópicos significativos e interpretáveis. Escolher um valor ainda mais alto às vezes pode fornecer subtópicos mais detalhados.

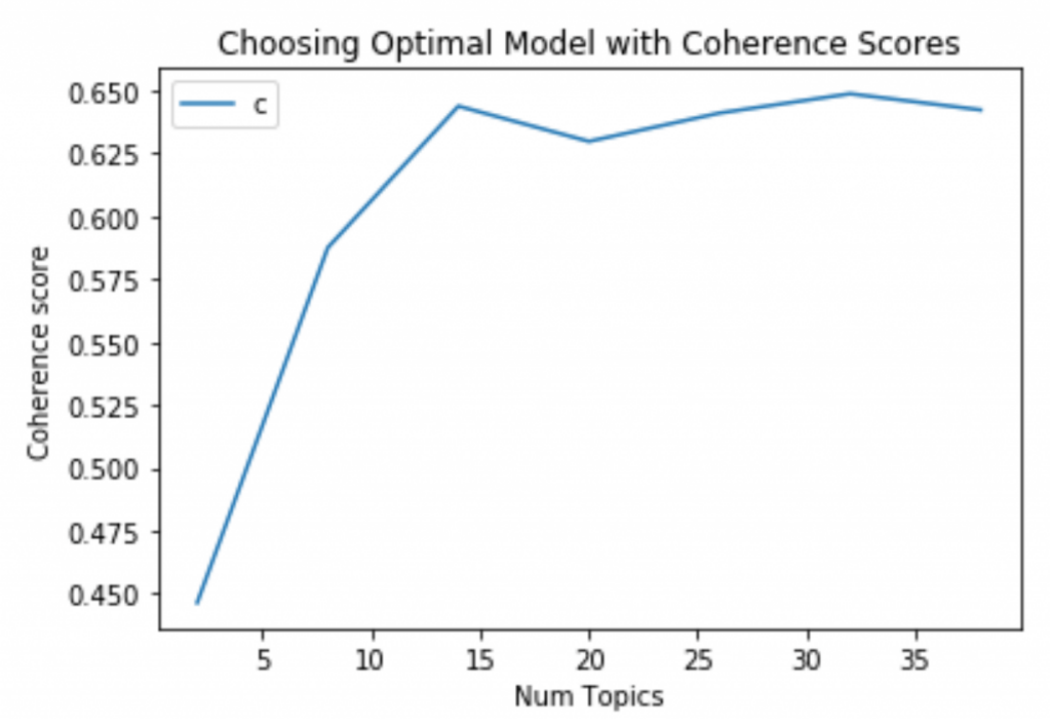
Se você vir as mesmas palavras-chave repetidas em vários tópicos, provavelmente é um sinal de que o 'k' é muito grande.

O (veja abaixo) treina vários modelos de LDA e fornece os modelos e suas correspondentes pontuações de coerência. compute_coherence_values()

```
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    """
    Compute c_v coherence for various number of topics

    Parameters:
    ----------
    dictionary : Gensim dictionary
    corpus : Gensim corpus
    texts : List of input texts
    limit : Max num of topics

    Returns:
    -------
    model_list : List of LDA topic models
    coherence_values : Coherence values corresponding to the LDA model with respective number of to
    """
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=num_topics,
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence=
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values
```

```
# Can take a long time to run.
model_list, coherence_values = compute_coherence_values(dictionary=id2word, corpus=corpus, texts=da
```

```
# Show graph
limit=40; start=2; step=6;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
```




Escolhendo o número ideal de tópicos da LDA

```
# Print the coherence scores
for m, cv in zip(x, coherence_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4))
```

```
Num Topics = 2  has Coherence Value of 0.4451
Num Topics = 8  has Coherence Value of 0.5943
Num Topics = 14  has Coherence Value of 0.6208
Num Topics = 20  has Coherence Value of 0.6438
Num Topics = 26  has Coherence Value of 0.643
Num Topics = 32  has Coherence Value of 0.6478
Num Topics = 38  has Coherence Value of 0.6525
```

Se o escore de coerência parecer continuar aumentando, pode fazer mais sentido escolher o modelo que deu o CV mais alto antes de achatar. Este é exatamente o caso aqui.

Portanto, para outras etapas, escolherei o modelo com 20 tópicos.

```
# Select the model and print the topics
optimal_model = model_list[3]
model_topics = optimal_model.show_topics(formatted=False)
pprint(optimal_model.print_topics(num_words=10))
```

```
[(0,
  '0.025*"game" + 0.018*"team" + 0.016*"year" + 0.014*"play" + 0.013*"good" + '
  '0.012*"player" + 0.011*"win" + 0.007*"season" + 0.007*"hockey" + '
  '0.007*"fan"'),
 (1,
  '0.021*"window" + 0.015*"file" + 0.012*"image" + 0.010*"program" + '
  '0.010*"version" + 0.009*"display" + 0.009*"server" + 0.009*"software" + '
  '0.008*"graphic" + 0.008*"application"'),
 (2,
  '0.021*"gun" + 0.019*"state" + 0.016*"law" + 0.010*"people" + 0.008*"case" + '
  '0.008*"crime" + 0.007*"government" + 0.007*"weapon" + 0.007*"police" + '
  '0.006*"firearm"'),
 (3,
  '0.855*"ax" + 0.062*"max" + 0.002*"tm" + 0.002*"qax" + 0.001*"mf" + '
  '0.001*"giz" + 0.001*"_" + 0.001*"ml" + 0.001*"fp" + 0.001*"mr"'),
 (4,
  '0.020*"file" + 0.020*"line" + 0.013*"read" + 0.013*"set" + 0.012*"program" '
  '+ 0.012*"number" + 0.010*"follow" + 0.010*"error" + 0.010*"change" + '
  '0.009*"entry"'),
 (5,
  '0.021*"god" + 0.016*"christian" + 0.008*"religion" + 0.008*"bible" + '
  '0.007*"life" + 0.007*"people" + 0.007*"church" + 0.007*"word" + 0.007*"man" '
  '+ 0.006*"faith"'),
 (..truncated..)]
```

Esses foram os tópicos para o modelo LDA escolhido.

# 18. Encontrar o tópico dominante em cada frase

Uma das aplicações práticas da modelagem de tópicos é determinar sobre o tópico de um determinado documento.

Para descobrir isso, encontramos o número do tópico que possui a maior porcentagem de contribuição nesse documento.

A função abaixo agrada agradavelmente essas informações em uma tabela apresentável. format_topics_sentences()

```
def format_topics_sentences(ldamodel=lda_model, corpus=corpus, texts=data):
    # Init output
    sent_topics_df = pd.DataFrame()

    # Get main topic in each document
    for i, row in enumerate(ldamodel[corpus]):
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Get the Dominant topic, Perc Contribution and Keywords for each document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => dominant topic
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                sent_topics_df = sent_topics_df.append(pd.Series([int(topic_num), round(prop_topic,
            else:
                break
    sent_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywords']

    # Add original text to the end of the output
    contents = pd.Series(texts)
    sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)
    return(sent_topics_df)


df_topic_sents_keywords = format_topics_sentences(ldamodel=optimal_model, corpus=corpus, texts=data

# Format
df_dominant_topic = df_topic_sents_keywords.reset_index()
df_dominant_topic.columns = ['Document_No', 'Dominant_Topic', 'Topic_Perc_Contrib', 'Keywords', 'Te

# Show
df_dominant_topic.head(10)
```

| Document_No | Dominant_Topic | Topic_Perc_Contrib | Keywords | Text |
|---|---|---|---|---|
| 0 | 16.0 | 0.1678 | car, bike, good, drive, buy, power, ground, light, road, dod | From: (wheres my thing) Subject: WHAT car is this!? Nntp-Posting-Host: rac3.... |
| 1 | 15.0 | 0.1916 | drive, card, system, problem, driver, scsi, mac, work, window, bit | From: (Guy Kuo) Subject: SI Clock Poll - Final Call Summary: Final call for ... |
| 2 | 16.0 | 0.2406 | car, bike, good, drive, buy, power, ground, light, road, dod | From: (Irwin Arnstein) Subject: Re: Recommendation on Duc Summary: Whats it ... |
| 3 | 5.0 | 0.3484 | window, file, program, image, version, display, server, application, run, gr... | From: (Tsung-Kun Chen) Subject: ** Software forsale (lots) ** Nntp-Posting-H... |
| 4 | 15.0 | 0.2332 | drive, card, system, problem, driver, scsi, mac, work, window, bit | From: (Don A.B. Lindbergh) Subject: Diamond SS24X, Win 3.1, Mouse cursor Org... |
| 5 | 16.0 | 0.4051 | car, bike, good, drive, buy, power, ground, light, road, dod | From: (Robert Loper) Subject: Re: SHO and SC Nntp-Posting-Host: sun278.dseg.... |
| 6 | 3.0 | 0.1533 | organization, line, nntp_post, host, university, write, reply, article, dist... | From: (Kim Richard Man) Subject: SyQuest 44M cartrifge FORSALE Article-I.D.:... |
| 7 | 18.0 | 0.2676 | line, organization, mail, sale, price, sell, interested, computer, fax, phone | From: (Kirtley Wilson) Subject: Mirosoft Office Package Article-I.D.: news.1... |
| 8 | 12.0 | 0.1779 | gun, state, law, people, government, crime, article, fire, weapon, firearm | Subject: Re: Dont more innocents die without the death penalty? From: (Rober... |
| 9 | 6.0 | 0.3625 | exist, question, evidence, write, claim, true, argument, people, science, re... | From: (Jon Livesey) Subject: Re: Genocide is Caused by Atheism Organization:... |

Tópico dominante para cada documento

# 19. Encontre o documento mais representativo para cada tópico

Às vezes, apenas as palavras-chave do tópico podem não ser suficientes para entender o significado de um tópico. Portanto, para ajudar a entender o tópico, você pode encontrar os documentos que um determinado tópico mais contribuiu e inferir o tópico lendo esse documento. Ufa !!

```
 # Group top 5 sentences under each topic
sent_topics_sorteddf_mallet = pd.DataFrame()

sent_topics_outdf_grpd = df_topic_sents_keywords.groupby('Dominant_Topic')

for i, grp in sent_topics_outdf_grpd:
    sent_topics_sorteddf_mallet = pd.concat([sent_topics_sorteddf_mallet,
                                             grp.sort_values(['Perc_Contribution'], ascending=[0]).
                                            axis=0)

# Reset Index
sent_topics_sorteddf_mallet.reset_index(drop=True, inplace=True)

# Format
sent_topics_sorteddf_mallet.columns = ['Topic_Num', "Topic_Perc_Contrib", "Keywords", "Text"]

# Show
sent_topics_sorteddf_mallet.head()
```