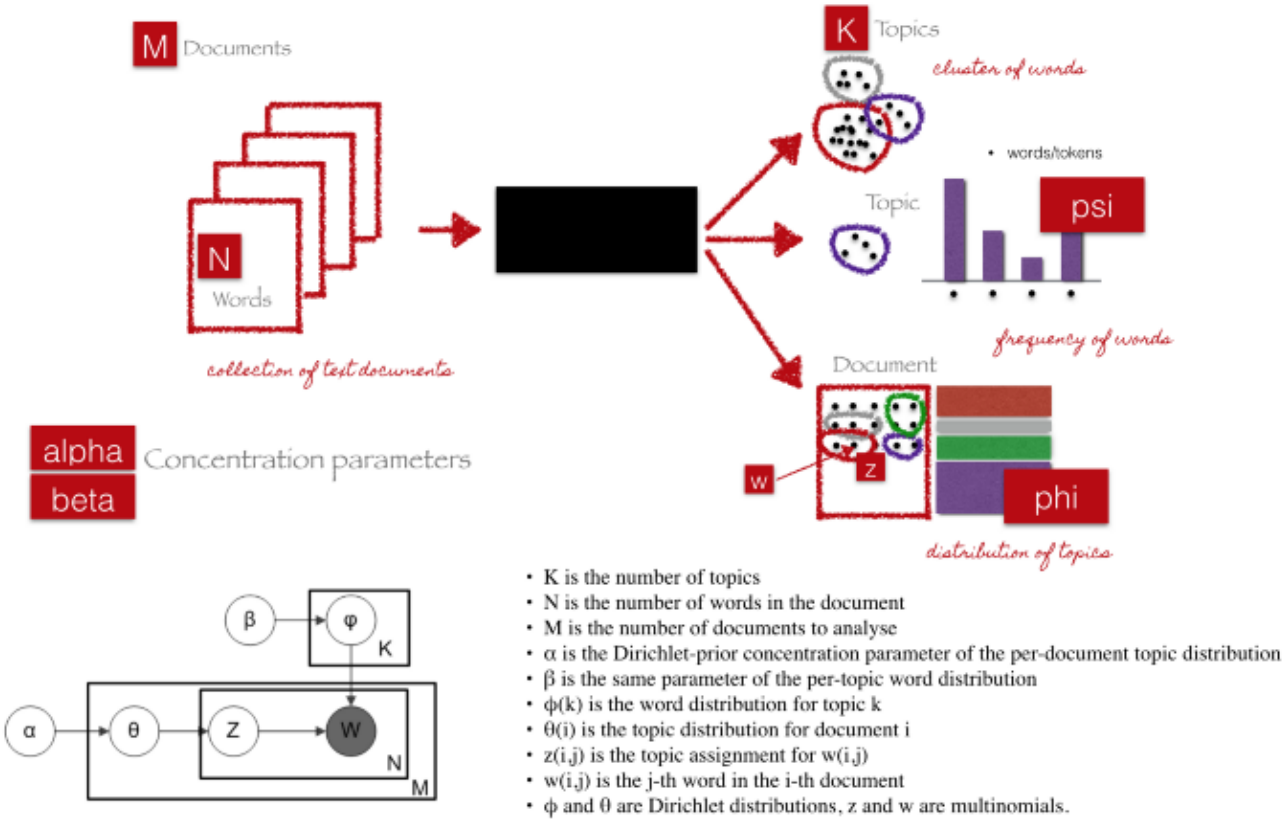
Como modelar tópicos através de Latent Dirichlet Allocation (LDA) através da biblioteca Gensim

Beatriz Yumi Simoes de Castro
Apr 12, 2020 · 11 min read

Estou me formando no curso de Data Science & Machine Learning da Tera e o meu projeto de conclusão de curso envolve diversas etapas, uma das quais, que ficou comigo, envolve modelagem de tópico de diversos dias de extrações de tweets de 19:00 até 00:00 com assuntos relacionados ao COVID-19 e à política. Resolvi fazer a modelagem através de Latent Dirichlet Allocation — LDA, pois tive experiência com este modelo em um hackathon do qual participei antes. Só que eu não entendia muito bem como que o modelo funcionava nem as suas métricas e é por isso que decidi explicar melhor sobre o modelo aqui. Eu também explico o porque que optei pela biblioteca Gensim e não a Scikit-Learn.

Agora, antes de explicar como fiz esse modelo, acredito que é interessante revisarmos os conceitos gerais de modelagem de tópico. A modelagem de tópicos é a aplicação de um modelo estatístico de linguagem que visa entender a estrutura que pode existir por detrás de um texto. Em resumo, o fluxo que acontece nesse tipo de modelagem é uma redução dos textos analisados; a aplicação de um aprendizado de máquina não supervisionado que gera os tópicos (pode ser comparado aos modelos de aprendizado de máquina não supervisionado que geram clusters, por exemplo), aonde cada tópico encorpora uma quantidade de palavras; e finalmente classificar os documentos analisados de acordo com os tópicos levantados, indicando qual o tópico que melhor se encaixa. Existem muitos modelos para realizar este tipo de análise, além do LDA, dentre os quais podemos listar Latent Semantic Analysis (LSA/LSI), Probabilistic Latent Semantic Analysis (pLSA) e a Non Negative Matrix Factorization (NNMF — que foi o outro modelo utilizado para modelar tópicos no projeto que meu grupo entregou para a conclusão do curso).

O LDA é um modelo ***altamente estatístico***, ele se baseia em acreditar que cada tópico é uma mistura de um conjunto de palavras e que cada documento é uma mistura de um conjunto de tópicos. Na figura a seguir conseguimos entender um pouco melhor como funciona o fluxo de informações dentro deste modelo.



http://chdoig.github.io/pytexas2015-topic-modeling/#/3/4

Para o LDA, entramos com uma quantidade ***M*** de documentos, que em si geram ***N*** palavras, que são tratadas no modelo gerando ***K*** tópicos, que podem ser entendidos como clusters de palavras. Demais outputs que se extraem do modelo, além dos tópicos em si, são a frequência de palavras por tópicos, representado na figura pelo ***psi***. Outro é a distribuição dos tópicos para cada documento ***phi***, ou seja, quanto percentualmente aquele tópico é relevante para documento. Já os principais parâmetros para o modelo em si, $\boldsymbol{\alpha}$ que indica de quantos tópicos os documentos são compostos, que permite ou não uma distribuição de tópicos por documento mais específica — quanto maior, maior a quantidade de tópicos, mais específica a distribuição; e o $\boldsymbol{\beta}$ que indica de quantas palavras os tópicos são compostos, que permite ou não uma distribuição de palavras por tópico mais específica — quanto maior, maior a quantidade de palavras, mais específica a distribuição.

Dentro do Python, é possível realizar o cálculo do modelo LDA através da biblioteca scikit-learn, cuja documentação você encontra aqui, também permite que o modelo seja aplicado e dá ao cientista a liberdade de realizar todo o processo, inclusive as transformações necessárias provenientes do modelo, algo que é resumido nas funções fit(), transform() e fit_transform() do scikit-learn. Independente do modelo escolhido, a primeira etapa que mencionei, de redução de textos deve ser seguida para que se obtenha tokens, ou palavras que serão analisadas pelos diferentes modelos.

O processo de redução de texto que foi realizado seguiu o seguinte fluxo:

- Tokenização dos tweets
- Redução dos tokens para uma seleção de 10 mil tokens mais significativos de acordo com uma amostra de 5 dias consolidados, significância calculada manualmente
- Remoção de caracteres especiais e de acentos através de regex e unicode
- Remoção de emojis (usando a biblioteca emoji)
- Remoção de stop words
- Aplicação de modelo Bigram
- Aplicação de modelo Trigram

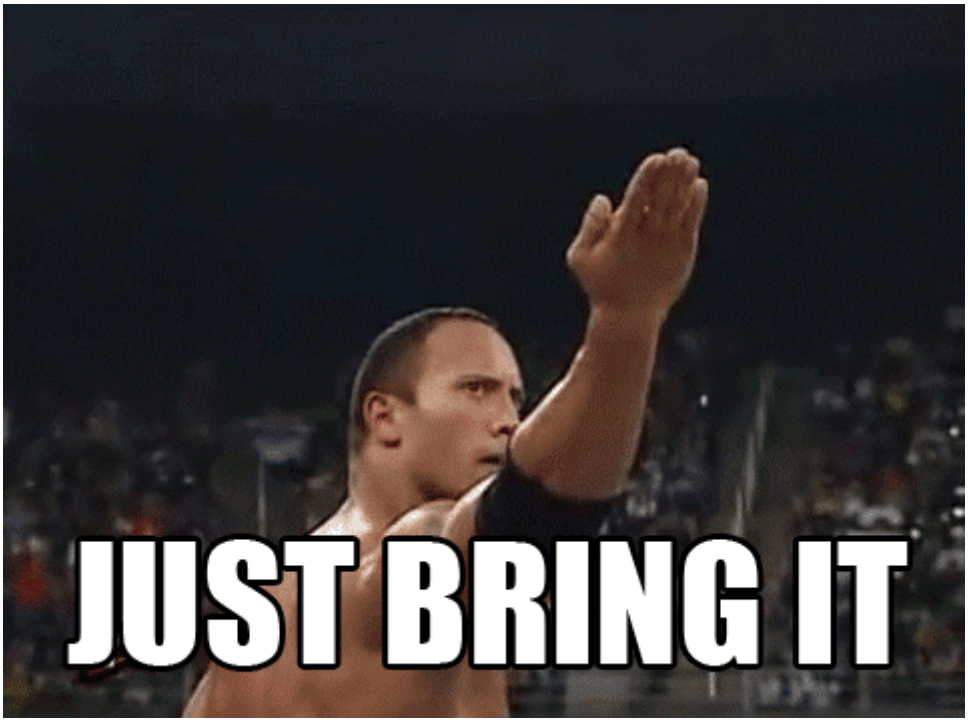
- Remoção novamente de stop words

Resolvi não aplicar nem o processo de stemming nem de lemmatização. O processo de lemmatização, conforme apresentado neste artigo, não apresenta bons resultados em português, independente da biblioteca utilizada, resultando em algo similar a um processo de stemming. Já o stemming não resultaria em bons tokens, já que um dos objetivos finais da minha análise é criar word clouds para cada um dos dias que existia dataset. Outro ponto é que ao longo do processamento fui aumentando a lista de stopwords conforme fui processando cada dia.

Alguns pontos que merecem ser mencionados sobre a redução do texto em português.

> *Não existe biblioteca com inputs suficientes para ter uma lista de* stop words *completa (lista de palavras com significado vazio que são removidas do processamento para não poluírem os resultados da modelagem), ou uma biblioteca que permitisse realizar um processo de stemm ou lemma com precisão. Os resultados não foram agradáveis com as soluções disponíveis, como nltk, inclusive com as soluções desenvolvidas específicas para a língua, como a SnowBall. No final, para realizar a análise de que me refiro neste artigo, criei uma lista de* stop words *manualmente pelo spacy . Trago isso para que você talvez se interesse em se envolver no desenvolvimento de soluções desse gênero, nós certamente precisamos.*

Vamos agora para a parte do aprendizado de máquina não supervisionado



Depois que tinha uma base preprocessada, consegui seguir para a modelagem em si

Primeiro, você precisa ter as bibliotecas necessárias pra conseguir realizar as análises. Aqui nós vamos trabalhar com a Gensim.

```
# gerais
import sys
import os
from pathlib import Path
import re
import numpy as np
import pandas as pd
from pprint import pprint
from tqdm import tqdm
import pt_core_news_sm

# Gensim
import gensim, spacy, logging, warnings
import gensim.corpora as corpora
from gensim.utils import lemmatize, simple_preprocess
from gensim.models import CoherenceModel
from gensim.models.ldamulticore import LdaMulticore

#plot
import pyLDAvis
import pyLDAvis.gensim
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
%matplotlib inline
from wordcloud import WordCloud, STOPWORDS
```

Depois, com a sua base já pronta, toda tokenizada em formato de lista, você precisa criar um dicionário e um corpus. O corpus irá retornar a frequência de cada palavra dentro de cada tweet, ele é uma lista de tuple, onde o primeiro elemento é o id da palavra e o segundo elemento é a frequência dela no tweet. No meu caso, a minha lista se chama ‘texto’.

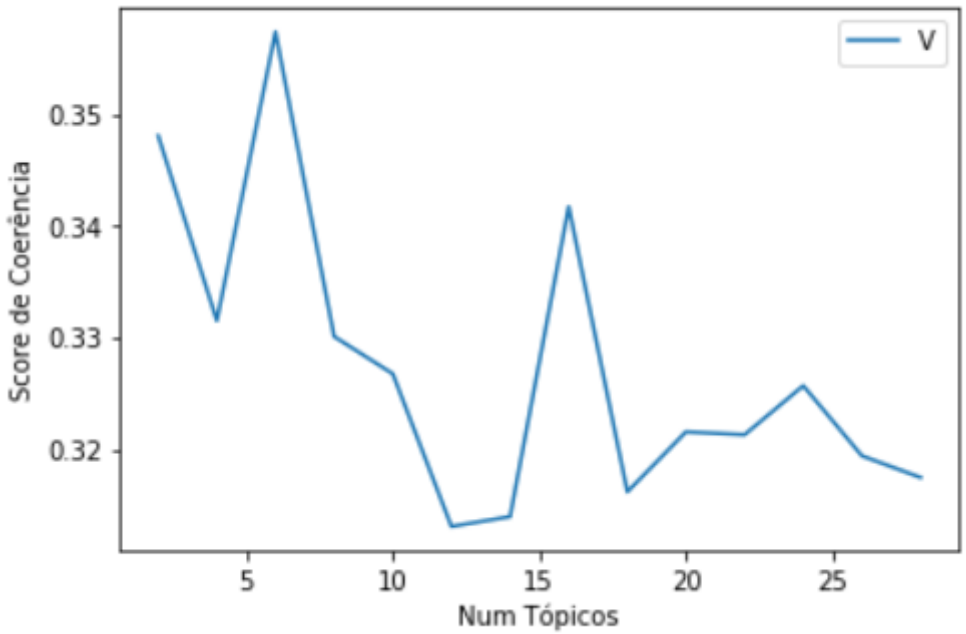
```
# Criar dicionário

id2word = corpora.Dictionary(texto)

# Criar o Corpus: A Frequência das palavras nos documentos

corpus = [id2word.doc2bow(text) for text in texto]
```

Antes de continuar, vou explicar o que é um valor de coerência. O valor de coerência mede, dentro de um único tópico, a coerência semântica das palavras dentro dele, utilizando a métrica de cossenos para sua similaridade, que vai de 0 a 1. Este é um dos modelos de coerência (o c_v), existem outros, que não cabem a discussão neste post. Existem outros critérios para avaliar também modelagem de tópicos, como a perplexidade.

O passo que eu costumo dar a seguir na modelagem é totalmente optativo. Vou explicar o porquê. Eu calculo o valor de coerência para cada quantidade de tópicos, para tentar identificar uma quantidade de tópicos "ideal" para se trabalhar. Aqui entram vários pontos que precisam ser discutidos. Primeiro, não dá para garantir que mesmo escolhendo o melhor valor de coerência vou ter tópicos coerentes depois, afinal de contas é um modelo não supervisionado. Isso inclusive aconteceu em cerca de quatro dias analisados, aonde esta análise me apontou para quatro tópicos como a quantidade ideal e vendo visualmente a distribuição e as palavras chaves era visível que existiam apenas três. Segundo, não se leva em consideração nenhum ajuste de hiperparametro aqui e isso pode por si só já melhorar muito o valor de coerência para a mesma quantidade de tópicos. Terceiro, valor de coerência, assim como outras métricas de aprendizado de máquina não supervisionado não são precisos. No final, é necessário se analisar manualmente os resultados e se ajustar o modelo novamente.

```
# Função para determinar a melhor quantidade de tópicos para a
modelagem

def compute_coherence_values(dictionary, corpus, texts, limit,
start=2, step=5):
    coherence_values = []
    model_list = []
    for num_topics in tqdm(range(start, limit, step)):
        model = LdaMulticore(corpus, id2word=id2word, num_topics=10)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts,
dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

return model_list, coherence_values

# Colocando parametros na função

model_list, coherence_values =
compute_coherence_values(dictionary=id2word, corpus=corpus,
texts=texto, start=2, limit=30, step=2)

# Mostrando visualmente a quantidade de tópicos
```

```
limit=30; start=2; step=2;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Tópicos")
plt.ylabel("Score de Coerência")
plt.legend(("Valores de Coerência"), loc='best')
plt.show()

# Lista dos valores de coerência, para melhor identificar o ponto de
inflexão do gráfico

for m, cv in zip(x, coherence_values):
    print("A quantidade de tópicos =", m, " tem um valor de
coerência de ", round(cv, 4))
```



Exemplo de gráfico de valor de coerência x número de tópicos

```
A quantidade de tópicos = 2  tem um valor de coerência de  0.3481
A quantidade de tópicos = 4  tem um valor de coerência de  0.3315
A quantidade de tópicos = 6  tem um valor de coerência de  0.3573
A quantidade de tópicos = 8  tem um valor de coerência de  0.3301
A quantidade de tópicos = 10  tem um valor de coerência de  0.3268
A quantidade de tópicos = 12  tem um valor de coerência de  0.3132
A quantidade de tópicos = 14  tem um valor de coerência de  0.314
A quantidade de tópicos = 16  tem um valor de coerência de  0.3417
A quantidade de tópicos = 18  tem um valor de coerência de  0.3163
A quantidade de tópicos = 20  tem um valor de coerência de  0.3216
A quantidade de tópicos = 22  tem um valor de coerência de  0.3214
A quantidade de tópicos = 24  tem um valor de coerência de  0.3257
A quantidade de tópicos = 26  tem um valor de coerência de  0.3195
A quantidade de tópicos = 28  tem um valor de coerência de  0.3175
```

Exemplo de tabela de apoio para decidir quantidade de tópicos

Quero ressaltar os valores de coerência, todos foram baixos, considerando que pode-se ir até 1. Realmente, no final, quando olho meus tópicos, consigo entender e ter uma ideia geral de cada um, mas não é possível dizer que existe uma forte coesão entre as palavras presentes em cada um.

Em seguida, realizo de fato a modelagem de tópicos, eu já selecionei os valores que me deram melhor resultado para alfa e beta. Fiz isso testando manualmente mesmo. Como a base é muito grande, estava me dando mais de 365 horas para voltar um teste que desse as melhores opções de alfa e beta (fica 4 for dentro de for), que com certeza iria extrapolar o prazo de entrega do meu projeto. Recomendo utilizar o teste que existe neste artigo do Towards Data Science. Basicamente você seta seu alfa e seu beta em 0.01 e verifica o maior valor de coerência e depois roda a função que tem lá, verificando qual combinação de alfa e beta traz o melhor valor de coerência para a quantidade de tópicos que você verificou previamente.

```
# Vamos agora construir de fato o modelo LDA

lda_model = LdaMulticore(corpus=corpus,
                         id2word=id2word,
                         num_topics=3,
                         random_state=42,
                         chunksize=100,
                         passes=10,
                         per_word_topics=True,
                         alpha = 0.9,
                         eta = 0.3)
```

Apesar de ter dito antes que não acredito que as métricas que avaliam aprendizados de máquina não supervisionados de modelagem de tópico de forma eficiente, realizei a análise da perplexidade e também da coerência.

```
# Computar a perplexidade do modelo (quanto menor, melhor)

print('\nPerplexidade: ', lda_model.log_perplexity(corpus))
```

```
Perplexidade:  -6.487251262934781
```

Exemplo de resultado de perplexidade

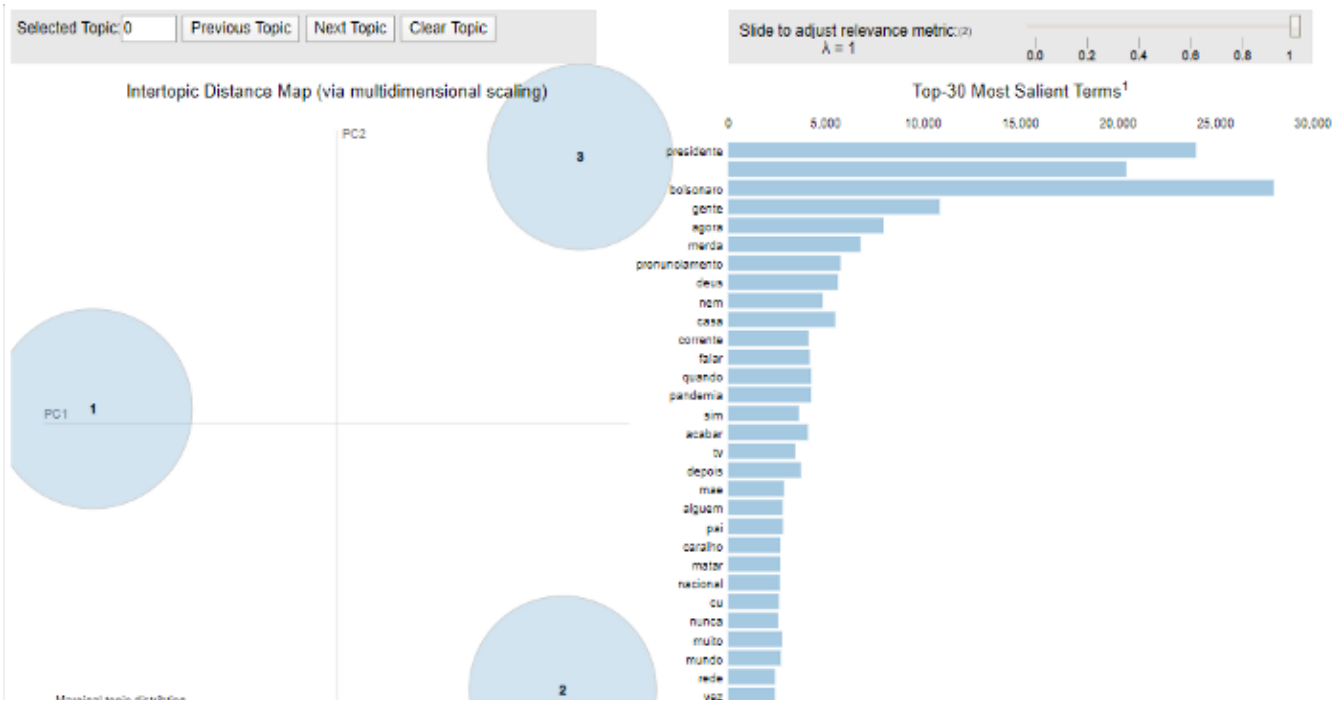
```
# Computar o Score de Coerência

coherence_model_lda = CoherenceModel(model=lda_model, texts=texto,
dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nScore de Coerência: ', coherence_lda)
```

```
Score de Coerência:  0.39213988208955675
```

Exemplo de resultado de coerência, como mencionei, o valor é muito baixo. O ideal seria algo mais próximo de 1.

Em seguida, eu crio um gráfico com os tópicos gerados por esse modelo. Dependendo de como os resultados se apresentam, modifico a quantidade de tópicos. A modificação depende da distribuição espacial dos tópicos e das palavras chave que se apresentam em cada um.

```
# Visualização dos tópicos em clusters

pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```



Exemplo de gráfico gerado, ao trocar de tópico, nos botões next topic, é possível verificar as principais palavras chave de cada tópico. A distribuição é feita de acordo com a frequência de cada palavra em cada tópico.

Depois disso, eu gero novamente funções de suporte para criar um novo modelo para extrair as palavras chave de acordo com a quantidade de tópico que determinei antes. É a mesma função de antes, devemos apenas ajustar a quantidade de tópicos.

```
# Função para determinar a melhor quantidade de tópicos para a
modelagem

def compute_coherence_values(dictionary, corpus, texts, limit,
start=2, step=5):
    coherence_values = []
    model_list = []
    for num_topics in tqdm(range(start, limit, step)):
        model = LdaMulticore(corpus, id2word=id2word, num_topics=3)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts,
dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

return model_list, coherence_values

# Colocando parametros na função

model_list, coherence_values =
compute_coherence_values(dictionary=id2word, corpus=corpus,
texts=texto, start=2, limit=4, step=2)
```

Apresentamos então os tópicos, as palavras chaves e seu percentual de contribuição.

```
# Escolhe o model

optimal_model = model_list[0]
optimal_model.print_topics(num_topics = 3, num_words = 20)
```

```
[(0,
  '0.055*"bolsonaro" + 0.045*"" + 0.019*"casa" + 0.018*"alguem" + 0.012*"corrente" + 0.011*"passar
ois" + 0.008*"presidente" + 0.007*"nunca" + 0.006*"vida" + 0.006*"amor" + 0.006*"pandemia" + 0.006
 (1,
  '0.160*"bolsonaro" + 0.038*"pronunciamento" + 0.035*"presidente" + 0.029*"merda" + 0.026*"gente"
```

```
v" + 0.011*"rede" + 0.010*"mundo" + 0.010*"depois" + 0.009*"atleta" + 0.009*"falar" + 0.008*"vez"
 (2,
  '0.083*"presidente" + 0.018*"quando" + 0.015*"brasil" + 0.015*"acabar" + 0.013*"muito" + 0.012*"
ha" + 0.008*"sim" + 0.008*"deus" + 0.008*"coisa" + 0.007*"merda" + 0.007*"casa" + 0.007*"nada" + 0
```

Exemplo da distribuição apresentada

Em seguida realizo diversas funções para poder encontrar os tópicos, seus percentuais de contribuição para os tweets e suas palavras chaves. Isso é muito importante para a análise que quero realizar, que é verificar quais palavras mais foram utilizadas em cada tópico por dia.

```
# Encontrar qual o principal tópico em cada tweet

def format_topics_sentences(ldamodel=lda_model, corpus=corpus,
texts=texto):
    # Output
    sent_topics_df = pd.DataFrame()

# Seleciona o principal tópico de cada tweet
    for i, row in enumerate(ldamodel[corpus]):
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Calcula o principal tópico, o percentual de contribuição e
as palavras chaves de cada tweet
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => mostra o principal tópico
                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,4), topic_keywords]), ignore_index=True)
            else:
                break
    sent_topics_df.columns = ['Principal_Topico',
'Perc_Contributicao', 'Palavras_Chave']

# Inclui o texto original no final do DataFrame
    contents = pd.Series(texts)
    sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)
    return(sent_topics_df)

# roda a função de calcular os principais tópicos de cada tweet

df_topic_sents_keywords =
format_topics_sentences(ldamodel=optimal_model, corpus=corpus,
texts=texto)

# Formata o DataFrame

df_dominant_topic = df_topic_sents_keywords.reset_index()
df_dominant_topic.columns = ['Index_DF_Original',
```

```
'Principal_Topico', 'Perc_Contrib_Topico', 'Palavras_Chave',
'Tweets']

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

sent_topics_outdf_grpd =
df_topic_sents_keywords.groupby('Principal_Topico')

for i, grp in sent_topics_outdf_grpd:
    sent_topics_sorteddf_mallet =
pd.concat([sent_topics_sorteddf_mallet,

grp.sort_values(['Perc_Contributicao'], ascending=[0]).head(1)],
                                            axis=0)

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

# Format
sent_topics_sorteddf_mallet.columns = ['Topico',
"Topico_Perc_Contrib", "Palavras_Chave", "Tweet"]

# Show
sent_topics_sorteddf_mallet
```

	Topico	Topico_Perc_Contrib	Palavras_Chave	Tweet
0	0.0	0.9542	bolsonaro, , casa, alguem, corrente, passar, g...	[pijama, paz, contatinhos, violento, passo, ge...
1	1.0	0.9600	bolsonaro, pronunciamento, presidente, merda, ...	[rede, nacional, crise, pandemia, perdido, cad...
2	2.0	0.9576	presidente, quando, brasil, acabar, muito, nem...	[falar, usar, casa, ir, lavar, maos, mal, volt...

Exemplo do resultado apresentado

Calculamos a quantidade de tweets por tópico.

```
# Numero de tweets por topico

topic_counts =
df_topic_sents_keywords['Principal_Topico'].value_counts()

# Porcentagem de tweets por tópico
topic_contribution = round(topic_counts/topic_counts.sum(), 4)

# Número do tópico e suas palavras chave
topic_num_keywords = df_topic_sents_keywords[['Principal_Topico',
'Palavras_Chave']]

# Concatena as colunas
df_dominant_topics = pd.concat([topic_num_keywords, topic_counts,
topic_contribution], axis=1)
```

```
# Renomeia as colunas
df_dominant_topics.columns = ['Principal_Topico', 'Palavras_Chave',
'Qtd_Tweets', 'Perc_Tweets']

# Mostra o DataFrame
df_dominant_topics
```

	Principal_Topico	Palavras_Chave	Qtd_Tweets	Perc_Tweets
0.0	2.0	presidente, quando, brasil, acabar, muito, nem...	32769.0	0.2448
1.0	1.0	bolsonaro, pronunciamento, presidente, merda, ...	57765.0	0.4315
2.0	0.0	bolsonaro, , casa, alguem, corrente, passar, g...	43332.0	0.3237

Exemplo do resultado de cálulo de tweets por tópico

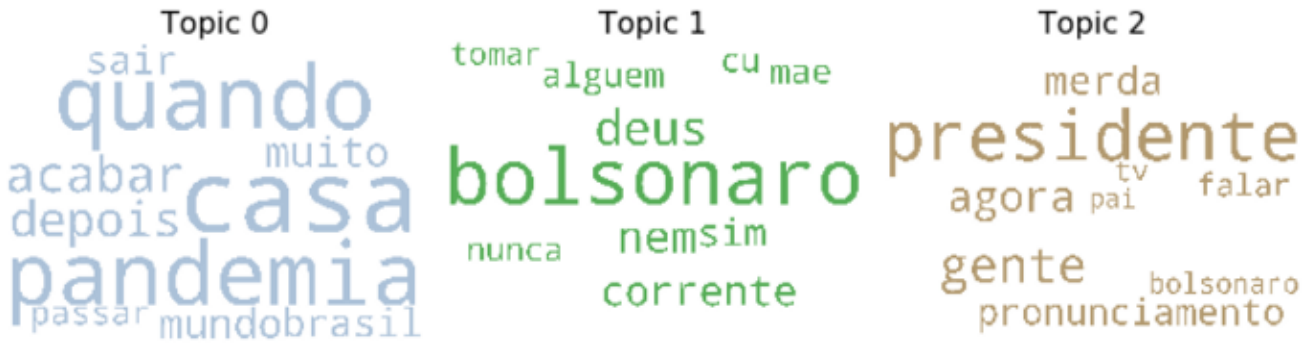
Por último, ploto as word clouds. Para isso preciso primeiro declarar as stop words que utilizei.

```
# Criando stopwords em português utilizando a biblioteca spacy
import pt_core_news_sm
nlp = pt_core_news_sm.load()
pt_stopwords = sorted([token.text for token in nlp.vocab if
token.is_stop])
list_exclude = ['obrigado', 'bom', 'mal', 'nenhuma', 'maior',
             'bem', 'não', 'máximo', 'boa', 'mais',
               'bastante', 'certamente', 'certeza', 'contra',
                'quarentena', 'coronavírus', 'presidente',
'impeachment', 'demitido', 'demitida']
for word in list_exclude:
    nlp.vocab[word].is_stop = False
list_include = set(['o', 'a', 'tá', 'ta', 'ser', 'pro', 'to', 'tô',
'vc', 'você', 'voce', 'pra',
                    'pq', 'é', 'vou', 'que','tão', 'gt', 'de', 'da',
'do', 'em', 'uma', 'lá',
                    'já', 'no', 'para', 'na', 'com', 'um', 'minha',
'se', 'isso', 'por', 'vou',
                    'os', 'isso', 'como', 'mesmo', 'tenho', 'aqui',
'ele', 'ela', 'quem', 'fazer',
                    'eu', 'só', 'ai', 'mais', 'só', 'querer',
'https', 'ter', 'estar', 'ficar',
                    'dos', 'das', 'vcs', 'tem', 'as', 'mas','ao'
                    'tava', 'nao', 'sao', 'ja', 'so', 'nossa',
                    'nosso', 'estao', 'tco', 'me', 'dia', 'te',
'ver', 'sera', 'porra', 'fez', 'ne',
                    'kkk','kkkkkk', 'puta', 'kkkkkkkk', 'hj',
'afff', 'gbr', 'meu', 'cara', 'guri', 'cmg',
                    'ctg', 'agr', 'pqp', 'vdd', 'eh', 'va', 'obg',

'corona','virus','coronavirus','covid','covid19','19'
                   'nem', 'numa', 'num', 'nuns', 'ces', 'voces',
'oce', 'oces', 'kkkk', 'vao', 'via',
```

```
                    'hj', 'hoje', 'tudo', 'todo', 'toda',
                    'vir', 'bem','ao','sem','ou','vai', 'dizer',
'entao', 'dizer', 'entao',
                    'tao', 'tu', 'mim', 'mano', 'oq', 'pos', 'dm',
'dps',
                    'coronavirusoutbreak', 'coronavirusPandemic',
'dar', 'vairus',
                    'ainda', 'assim']
                  )
for w in list_include:
    nlp.vocab[w].is_stop = True
stop_words = sorted([token.text for token in nlp.vocab if
token.is_stop])
```

Como vocês podem ver, a ordem está bem confusa das stop words. Isso aconteceu porque a cada modelagem fomos incluindo e retirando itens diferentes. Isso não só no meu modelo, mas no modelo de NNMF que meu grupo utilizou também.

```
# Cria wordclouds

cols = [color for name, color in mcolors.XKCD_COLORS.items()]

cloud = WordCloud(stopwords=stop_words,
                  background_color='white',
                  width=2500,
                  height=1800,
                  max_words=20,
                  colormap='tab10',
                  color_func=lambda *args, **kwargs: cols[i],
                  prefer_horizontal=1.0)

topics = lda_model.show_topics(formatted=False)

fig, axes = plt.subplots(1, 3, figsize=(10,10), sharex=True,
sharey=True)

for i, ax in enumerate(axes.flatten()):
    fig.add_subplot(ax)
    topic_words = dict(topics[i][1])
    cloud.generate_from_frequencies(topic_words, max_font_size=600)
    plt.gca().imshow(cloud)
    plt.gca().set_title('Topic ' + str(i), fontdict=dict(size=16))
    plt.gca().axis('off')

plt.subplots_adjust(wspace=0, hspace=0)
plt.axis('off')
plt.margins(x=0, y=0)
plt.tight_layout()
plt.show()
```



Exemplo de word cloud extraída

É isso, espero que fique claro pra vocês também como que o modelo LDA e a biblioteca Gensim funcionam. Também é possível plotá-lo utilizando o t-SNE, mas vou explicar isso melhor em outro post.

Data Science Machine Learning Lda Topic Modeling Pbl