LDA em Python - Como pesquisar na grade os melhores modelos de tópicos?

por Selva Prabhakaran |

O Scikit Learn do Python fornece uma interface conveniente para modelagem de tópicos usando algoritmos como alocação de diretórios latentes (LDA), LSI e fatoração de matriz não negativa. Neste tutorial, você aprenderá como criar o melhor modelo de tópico LDA possível e explorará como exibir as saídas como resultados significativos.

1.

Conteúdo

Como criar modelos de tópicos com o python sklearn. Foto de Sebastien Gabriel.

1. Introdução

No último tutorial, você viu como criar modelos de tópicos com o LDA usando o gensim . Neste tutorial, no entanto, vou usar a biblioteca de aprendizado de máquina mais popular do python - o scikit learn .

Com o scikit learn, você tem uma interface totalmente diferente e, com a pesquisa em grade e os vetorizadores, você tem muitas opções para explorar, a fim de encontrar o modelo ideal e apresentar os resultados.

Neste tutorial, você aprenderá:

1. Como limpar e processar dados de texto?
2. Como preparar os documentos de texto para criar modelos de tópicos com o scikit learn?
3. Como criar um modelo de tópico básico usando o LDA e entender os parâmetros?
4. Como extrair as palavras-chave do tópico?
5. Como pesquisar em grade e ajustar o modelo ideal?
6. Como obter os tópicos dominantes em cada documento?
7. Revise e visualize a distribuição de palavras-chave do tópico
8. Como prever os tópicos para um novo pedaço de texto?
9. Agrupe os documentos com base na distribuição de tópicos
10. Como obter a maioria dos documentos semelhantes com base nos tópicos discutidos?

Muitas coisas emocionantes pela frente. Vamos rolar!

2. Carregue os pacotes

O pacote principal usado neste tutorial é o scikit-learn (`sklearn`).

As expressões regulares `re` , `gensim` e `spacy` são usados para textos de processo. `pyLDAvis` e `matplotlib` para visualização e `numpy` e `pandas` para a manipulação e a visualização de dados em formato tabular.

Vamos importá-los.

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

import numpy as np
import pandas as pd
import re, nltk, spacy, gensim

# Sklearn
from sklearn.decomposition import LatentDirichletAllocation, TruncatedSVD
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import GridSearchCV
from pprint import pprint

# Plotting tools
import pyLDAvis
import pyLDAvis.sklearn
import matplotlib.pyplot as plt
%matplotlib inline
```

3. Importar dados de texto de grupos de notícias

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

Como ele está em um formato json com uma estrutura consistente, estou 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())
```

```
['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']
```

```
df.head(15)
```

	content	target	target_names
0	From: lerxst@wam.umd.edu (where's my thing)\nSubject: WHAT car is this!?\nNn...	7	rec.autos
1	From: guykuo@carson.u.washington.edu (Guy Kuo)\nSubject: SI Clock Poll - Fin...	4	comp.sys.mac.hardware
10	From: irwin@cmptrc.lonestar.org (Irwin Arnstein)\nSubject: Re: Recommendatio...	8	rec.motorcycles
100	From: tchen@magnus.acs.ohio-state.edu (Tsung-Kun Chen)\nSubject: ** Software...	6	misc.forsale
1000	From: dabl2@nlm.nih.gov (Don A.B. Lindbergh)\nSubject: Diamond SS24X, Win 3....	2	comp.os.ms-windows.misc
10000	From: a207706@moe.dseg.ti.com (Robert Loper)\nSubject: Re: SHO and SC\nNntp-...	7	rec.autos
10001	From: kimman@magnus.acs.ohio-state.edu (Kim Richard Man)\nSubject: SyQuest 4...	6	misc.forsale
10002	From: kwilson@casbah.acns.nwu.edu (Kirtley Wilson)\nSubject: Mirosoft Office...	2	comp.os.ms-windows.misc
10003	Subject: Re: Don't more innocents die without the death penalty?\nFrom: bobb...	0	alt.atheism
10004	From: livesey@solntze.wpd.sgi.com (Jon Livesey)\nSubject: Re: Genocide is Ca...	0	alt.atheism
10005	From: dls@aeg.dsto.gov.au (David Silver)\nSubject: Re: Fractal Generation of...	1	comp.graphics
10006	Subject: Re: Mike Francesa's 1993 Predictions\nFrom: gajarsky@pilot.njin.net...	9	rec.sport.baseball
10007	From: jet@netcom.Netcom.COM (J. Eric Townsend)\nSubject: Re: Insurance and l...	8	rec.motorcycles
10008	From: gld@cunixb.cc.columbia.edu (Gary L Dare)\nSubject: Re: ABC coverage\nN...	10	rec.sport.hockey
10009	From: sehari@iastate.edu (Babak Sehari)\nSubject: Re: How to the disks copy ...	12	sci.electronics

Entrada - 20 Newsgroups

4. Remova e-mails e caracteres de nova linha

Você pode ver muitos emails, caracteres de nova linha e espaços extras no texto e isso é bastante perturbador. 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. If anyone can tellme a model name, engine specs, years of '
 'production, where this car is made, history, or whatever info you have on '
 'this funky looking car, please e-mail. Thanks, - IL ---- brought to you by '
 'your neighborhood Lerxst ---- ']
```

5. Tokenizar e limpar usando simple_preprocess () de gensim

As frases parecem melhores agora, mas você deseja colocar cada frase em uma lista de palavras, removendo pontuações e caracteres desnecessários.

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 pun

data_words = list(sent_to_words(data))

print(data_words[:1])
```

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

A

6. Lematização

lematização é um processo em que convertemos palavras em sua raiz.

Por exemplo: 'Estudar' torna-se 'Estudo', 'Reunião torna-se' Conheça ',' Melhor 'e' Melhor 'torna-se' Bom '.

A vantagem disso é que conseguimos reduzir o número total de palavras exclusivas no dicionário. Como resultado, o número de colunas na matriz documento-palavra (criada pelo CountVectorizer na próxima etapa) será mais densa com colunas menores.

Você pode esperar que tópicos melhores sejam gerados no final.

```
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(" ".join([token.lemma_ if token.lemma_ not in ['-PRON-'] else '' for toke
    return texts_out

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

# Do lemmatization keeping only Noun, Adj, Verb, Adverb
data_lemmatized = lemmatization(data_words, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

print(data_lemmatized[:2])
```

```
['where s  thing subject what car be nntp post host rac wam umd edu organization university maryla
```

O

7. Crie a matriz Documento-Palavra

algoritmo de modelo de tópico LDA requer uma matriz de palavras do documento como entrada principal.

Você pode criar um usando CountVectorizer . No código abaixo, eu configurei o comando CountVectorizer para considerar palavras que ocorreram pelo menos 10 vezes (min_df), remover palavras de parada em inglês incorporadas, converter todas as palavras em letras minúsculas e uma palavra pode conter números e alfabetos com pelo menos o comprimento 3 em ordem ser qualificado como uma palavra.

Portanto, para criar a matriz doc-word, você precisa primeiro inicializar a CountVectorizer classe com a configuração necessária e depois aplicar fit_transform para realmente criar a matriz.

Como a maioria das células contém zeros, o resultado será na forma de uma matriz esparsa para economizar memória.

Se você deseja materializá-lo em um formato de matriz 2D, chame o método da matriz esparsa como foi feito na próxima etapa. todense()

```
vectorizer = CountVectorizer(analyzer='word',
                             min_df=10,                        # minimum reqd occurences of a word
                             stop_words='english',             # remove stop words
                             lowercase=True,                   # convert all words to lowercase
                             token_pattern='[a-zA-Z0-9]{3,}',  # num chars > 3
                             # max_features=50000,             # max number of uniq words
                            )

data_vectorized = vectorizer.fit_transform(data_lemmatized)
```

8. Verifique a Sparsicity

A escarificação não é senão a porcentagem de pontos de dados diferentes de zero na matriz documento-palavra, ou seja data_vectorized .

Como a maioria das células nessa matriz será zero, estou interessado em saber qual porcentagem de células contém valores diferentes de zero.

```
# Materialize the sparse data
data_dense = data_vectorized.todense()

# Compute Sparsicity = Percentage of Non-Zero cells
print("Sparsicity: ", ((data_dense > 0).sum()/data_dense.size)*100, "%")
```

```
Sparsicity:  0.775887569365 %
```

9. Crie o modelo LDA com o sklearn

Tudo está pronto

para criar um modelo de Alocação Dirichlet Latente (LDA). Vamos inicializar um e chamar para construir o modelo LDA. fit_transform()

Para este exemplo, eu defini os n_topics como 20 com base no conhecimento prévio sobre o conjunto de dados. Mais tarde, encontraremos o número ideal usando a pesquisa em grade.

```
# Build LDA Model
lda_model = LatentDirichletAllocation(n_topics=20,               # Number of topics
                                      max_iter=10,               # Max learning iterations
                                      learning_method='online',
                                      random_state=100,          # Random state
                                      batch_size=128,            # n docs in each learning iter
                                      evaluate_every = -1,       # compute perplexity every n iter
                                      n_jobs = -1,               # Use all available CPUs
                                     )
lda_output = lda_model.fit_transform(data_vectorized)

print(lda_model)  # Model attributes
```

```
LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
             evaluate_every=-1, learning_decay=0.7,
             learning_method='online', learning_offset=10.0,
             max_doc_update_iter=100, max_iter=10, mean_change_tol=0.001,
             n_components=10, n_jobs=-1, n_topics=20, perp_tol=0.1,
             random_state=100, topic_word_prior=None,
             total_samples=1000000.0, verbose=0)
```

10. Diagnostique o desempenho do modelo com perplexidade e probabilidade de log

Um modelo com maior probabilidade logarítmica e menor perplexidade (exp (-1. * Probabilidade logarítmica por palavra)) é considerado bom. Vamos verificar o nosso modelo.

```
# Log Likelyhood: Higher the better
print("Log Likelihood: ", lda_model.score(data_vectorized))

# Perplexity: Lower the better. Perplexity = exp(-1. * log-likelihood per word)
print("Perplexity: ", lda_model.perplexity(data_vectorized))

# See model parameters
pprint(lda_model.get_params())
```

```
Log Likelihood:  -9965645.21463
Perplexity:  2061.88393838
{'batch_size': 128,
 'doc_topic_prior': None,
 'evaluate_every': -1,
 'learning_decay': 0.7,
 'learning_method': 'online',
 'learning_offset': 10.0,
 'max_doc_update_iter': 100,
 'max_iter': 10,
 'mean_change_tol': 0.001,
 'n_components': 10,
 'n_jobs': -1,
 'n_topics': 20,
 'perp_tol': 0.1,
 'random_state': 100,
 'topic_word_prior': None,
 'total_samples': 1000000.0,
 'verbose': 0}
```

Em uma observação diferente, a perplexidade pode não ser a melhor medida para avaliar modelos de tópicos, porque não considera o contexto e as associações semânticas entre as palavras. Isso pode ser capturado usando a medida de coerência de tópicos, um exemplo disso é descrito no tutorial do gensim que mencionei anteriormente.

11. Como GridSearch o melhor modelo de LDA?

O parâmetro de ajuste mais importante para os modelos LDA é `n_components` (número de tópicos). Além disso, vou procurar `learning_decay` (que controla a taxa de aprendizado) também.

Além desses, outros possíveis parâmetros de pesquisa podem ser `learning_offset` (menos que as iterações iniciais. Devem ser> 1) e `max_iter` . Vale a pena experimentar se você tiver recursos de computação suficientes.

Esteja avisado, a pesquisa na grade constrói vários modelos LDA para todas as combinações possíveis de valores de parâmetros no parâmetro param_grid. Portanto, esse processo pode consumir muito tempo e recursos.

```
# Define Search Param
search_params = {'n_components': [10, 15, 20, 25, 30], 'learning_decay': [.5, .7, .9]}

# Init the Model
lda = LatentDirichletAllocation()

# Init Grid Search Class
model = GridSearchCV(lda, param_grid=search_params)

# Do the Grid Search
model.fit(data_vectorized)
```

```
GridSearchCV(cv=None, error_score='raise',
       estimator=LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
             evaluate_every=-1, learning_decay=0.7, learning_method=None,
             learning_offset=10.0, max_doc_update_iter=100, max_iter=10,
             mean_change_tol=0.001, n_components=10, n_jobs=1,
             n_topics=None, perp_tol=0.1, random_state=None,
             topic_word_prior=None, total_samples=1000000.0, verbose=0),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_topics': [10, 15, 20, 25, 30], 'learning_decay': [0.5, 0.7, 0.9]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

12. Como ver o melhor modelo de tópico e seus parâmetros?

```
# Best
best_lo

# Model
print('

# Log L
print('

# Perpl
print('
```

```
Best Model's Params:  {'learning_decay': 0.9, 'n_topics': 10}
Best Log Likelyhood Score:  -3417650.82946
Model Perplexity:  2028.79038336
```

13. Compare as pontuações de desempenho do modelo LDA

A plotagem das pontuações de probabilidade de log com os num_topics mostra claramente o número de tópicos = 10 tem pontuações melhores. E learning_decay de 0,7 supera 0,5 e 0,9.

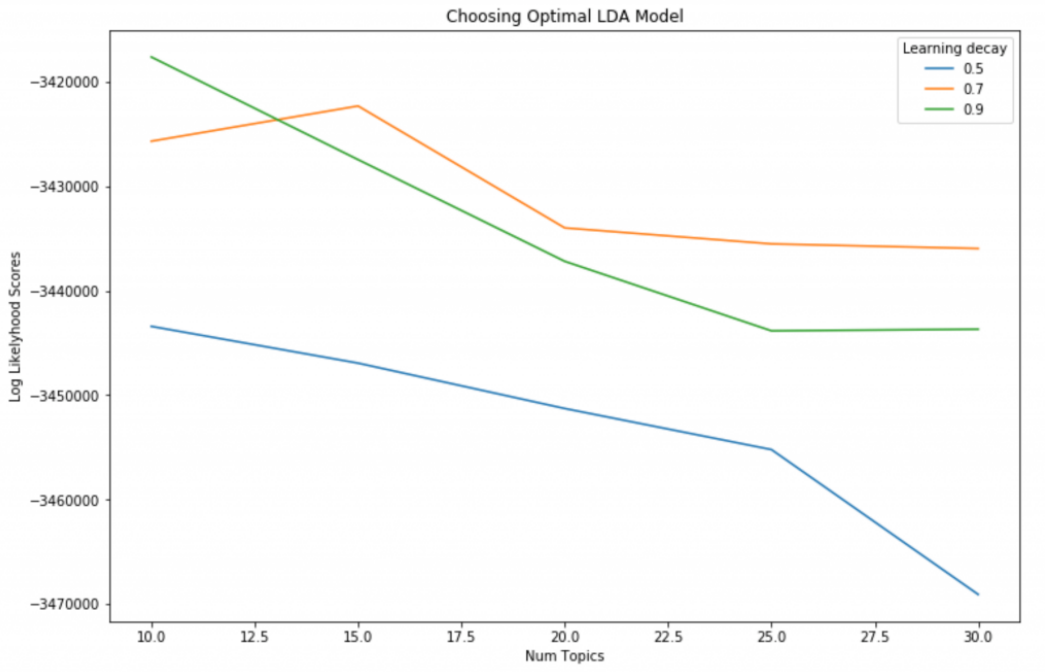
Isso me faz pensar que, embora saibamos que o conjunto de dados tem 20 tópicos distintos para começar, alguns tópicos podem compartilhar palavras-chave comuns. Por exemplo, 'alt.atheism' e 'soc.religion.christian' podem ter muitas palavras comuns. Mesmo com 'rec.motorcycles' e 'rec.autos', 'comp.sys.ibm.pc.hardware' e 'comp.sys.mac.hardware', você entendeu.

Para ajustar ainda mais, você pode fazer uma pesquisa mais precisa na grade para um número de tópicos entre 10 e 15. Mas eu vou pular isso por enquanto.

Portanto, o ponto principal é que um número ideal menor de tópicos distintos (até 10 tópicos) pode ser razoável para esse conjunto de dados. Ainda não sei disso. Mas a LDA diz isso. Vamos ver.

```
# Get Log Likelyhoods from Grid Search Output
n_topics = [10, 15, 20, 25, 30]
log_likelyhoods_5 = [round(gscore.mean_validation_score) for gscore in model.grid_scores_ if gscor
log_likelyhoods_7 = [round(gscore.mean_validation_score) for gscore in model.grid_scores_ if gscor
log_likelyhoods_9 = [round(gscore.mean_validation_score) for gscore in model.grid_scores_ if gscor

# Show graph
plt.figure(figsize=(12, 8))
plt.plot(n_topics, log_likelyhoods_5, label='0.5')
plt.plot(n_topics, log_likelyhoods_7, label='0.7')
plt.plot(n_topics, log_likelyhoods_9, label='0.9')
plt.title("Choosing Optimal LDA Model")
plt.xlabel("Num Topics")
plt.ylabel("Log Likelyhood Scores")
plt.legend(title='Learning decay', loc='best')
plt.show()
```



Modelos de tópicos de pesquisa de grade

Para

14. Como ver o tópico dominante em cada documento?

classificar um documento como pertencente a um tópico específico, uma abordagem lógica é ver qual tópico tem a maior contribuição para esse documento e atribuí-lo.

Na tabela abaixo, esverdeiei todos os principais tópicos de um documento e designei o tópico mais dominante em sua própria coluna.

```
 # Create Document - Topic Matrix
lda_output = best_lda_model.transform(data_vectorized)

# column names
topicnames = ["Topic" + str(i) for i in range(best_lda_model.n_topics)]

# index names
docnames = ["Doc" + str(i) for i in range(len(data))]

# Make the pandas dataframe
df_document_topic = pd.DataFrame(np.round(lda_output, 2), columns=topicnames, index=docnames)

# Get dominant topic for each document
dominant_topic = np.argmax(df_document_topic.values, axis=1)
df_document_topic['dominant_topic'] = dominant_topic

# Styling
def color_green(val):
    color = 'green' if val > .1 else 'black'
    return 'color: {col}'.format(col=color)

def make_bold(val):
    weight = 700 if val > .1 else 400
    return 'font-weight: {weight}'.format(weight=weight)

# Apply Style
df_document_topics = df_document_topic.head(15).style.applymap(color_green).applymap(make_bold)
df_document_topics
```

	Topic0	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	dominant_topic
Doc0	0	0	0	0	0	0.14	0	0	0	0.84	9
Doc1	0	0.05	0	0	0.05	0.24	0	0.65	0	0	7
Doc2	0	0	0	0	0	0.08	0.2	0	0	0.71	9
Doc3	0	0.55	0	0	0	0.44	0	0	0	0	1
Doc4	0.16	0.29	0	0	0	0.53	0	0	0	0	5
Doc5	0	0	0.05	0	0	0	0	0.12	0	0.83	9
Doc6	0	0	0	0	0	0.88	0.1	0	0	0	5
Doc7	0	0	0	0	0	0.99	0	0	0	0	5
Doc8	0	0	0.08	0.67	0	0	0	0	0.24	0	3
Doc9	0	0	0.74	0	0	0.14	0	0	0.11	0	2
Doc10	0	0	0	0	0.41	0.16	0	0.06	0	0.36	4
Doc11	0	0	0	0	0	0	0	0.97	0	0	7
Doc12	0	0	0	0.44	0	0.04	0	0.27	0	0.24	3
Doc13	0.14	0	0	0	0	0.07	0.57	0.08	0	0.13	6
Doc14	0	0	0	0	0.78	0.22	0	0	0	0	4

Pesos de tópicos do documento: df_document_topics

15. Revise a distribuição de tópicos entre documentos

```
df_top
df_topi
df_topi
```

	Topic Num	Num Documents
0	5	2979
1	2	1516
2	4	1286
3	9	1252
4	1	1024
5	7	976
6	3	930
7	8	607
8	0	428
9	6	316

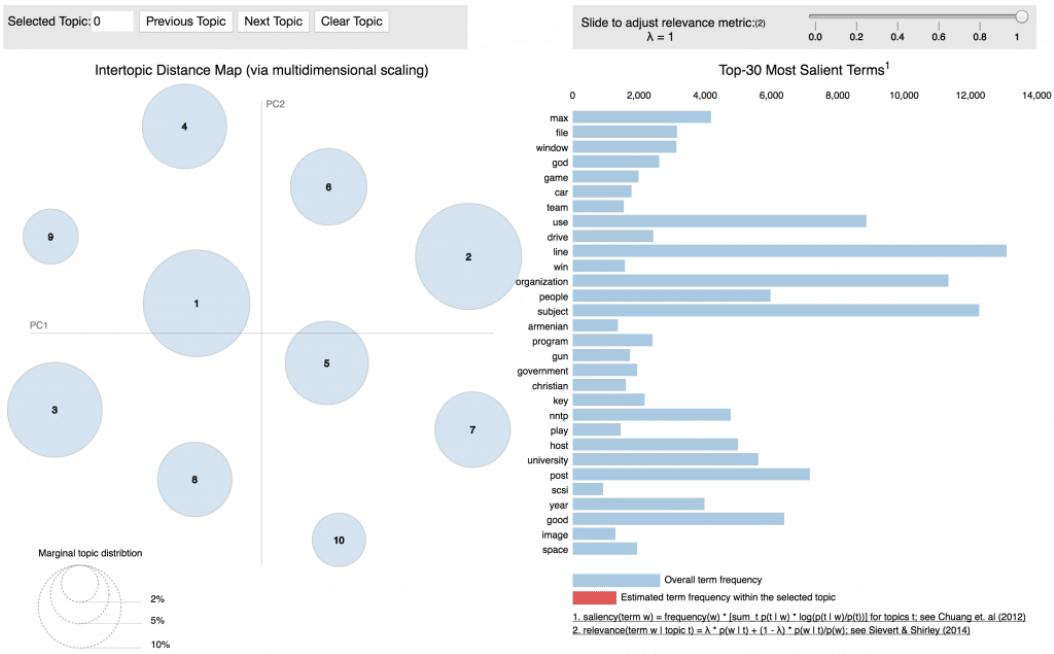
Distribuição de documentos do tópico: df_topic_distribution

16. Como visualizar o modelo de LDA com pyLDAvis?

O pyLDAvis oferece a melhor visualização para visualizar a distribuição de tópicos-palavras-chave.

Um bom modelo de tópico terá blobs de tamanho relativamente grande e não sobrepostos para cada tópico. Este parece ser o caso aqui. Então, nós somos bons.

```
pyLDAvis.enable_notebook()
panel = pyLDAvis.sklearn.prepare(best_lda_model, data_vectorized, vectorizer, mds='tsne')
panel
```



Visualize a distribuição de tópicos usando pyLDAvis

17. Como ver as palavras-chave do tópico?

Os pesos de cada palavra-chave em cada tópico estão contidos em uma matriz 2D. Os nomes das próprias palavras-chave podem ser obtidos no objeto usando . lda_model.components_ vectorizer get_feature_names()

Vamos usar essas informações para construir uma matriz de peso para todas as palavras-chave em cada tópico.

```
# Topic-Keyword Matrix
df_topic_keywords = pd.DataFrame(best_lda_model.components_)

# Assign Column and Index
df_topic_keywords.columns = vectorizer.get_feature_names()
df_topic_keywords.index = topicnames

# View
df_topic_keywords.head()
```

	aaa	aaron	abandon	abbreviation	abc	abide	ability	able	abolish	abomination	...	zion	zionism	zionist	zip	zisfein	zone	zoology
Topic0	9.575406	0.359981	0.100651	0.101689	0.117518	0.103411	3.453262	0.191085	0.107853	0.101680	...	0.100883	0.100633	0.100840	0.104984	0.100637	0.138300	0.100731
Topic1	0.100890	0.105746	0.131275	1.755703	0.101024	0.100685	19.461132	120.830022	0.104326	0.100753	...	0.100750	0.100667	0.100638	153.218332	0.100576	0.101657	0.100688
Topic2	0.109275	18.638578	31.992522	0.109649	0.110307	0.109873	85.985221	163.572888	23.501138	2.393444	...	0.101008	0.100948	0.102842	0.299180	0.100691	0.303542	0.100890
Topic3	0.103078	0.179178	0.101424	0.106328	0.101892	0.134666	5.270624	65.015866	0.100779	0.102793	...	1.246157	0.100726	0.102040	0.103102	29.849950	0.249888	82.317755
Topic4	0.100805	0.101627	1.919950	0.409059	6.007337	75.657222	87.171712	198.055427	27.465725	0.210156	...	0.256475	0.121024	0.143113	16.549160	0.115601	2.433356	0.101339

Pesos das Palavras do Tópico: df_topic_keywords

18. Obtenha as 15 principais palavras-chave de cada tópico

A partir da saída acima, desejo ver as 15 principais palavras-chave que são representativas do tópico.

O definido abaixo cria isso. show_topics()

```
# Show top n keywords for each topic
def show_topics(vectorizer=vectorizer, lda_model=lda_model, n_words=20):
    keywords = np.array(vectorizer.get_feature_names())
    topic_keywords = []
    for topic_weights in lda_model.components_:
        top_keyword_locs = (-topic_weights).argsort()[:n_words]
        topic_keywords.append(keywords.take(top_keyword_locs))
    return topic_keywords

topic_keywords = show_topics(vectorizer=vectorizer, lda_model=best_lda_model, n_words=15)

# Topic - Keywords Dataframe
df_topic_keywords = pd.DataFrame(topic_keywords)
df_topic_keywords.columns = ['Word '+str(i) for i in range(df_topic_keywords.shape[1])]
df_topic_keywords.index = ['Topic '+str(i) for i in range(df_topic_keywords.shape[0])]
df_topic_keywords
```

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9	Word 10	Word 11	Word 12	Word 13	Word 14
Topic 0	game	team	win	play	year	line	organization	subject	league	season	fan	new	san	baseball	red
Topic 1	file	window	use	program	image	run	version	line	available	server	ftp	set	user	software	display
Topic 2	say	god	people	write	think	know	believe	christian	make	subject	line	good	just	organization	thing
Topic 3	people	gun	say	article	write	just	know	time	make	think	organization	line	subject	child	year
Topic 4	key	use	space	government	make	law	line	organization	write	subject	public	people	encryption	year	know
Topic 5	line	subject	organization	post	university	host	nntp	edu	thank	write	computer	article	know	use	distribution
Topic 6	max	use	bit	line	subject	wire	chip	bhj	organization	giz	power	signal	high	circuit	cable
Topic 7	drive	good	write	line	think	organization	subject	article	scsi	year	time	make	game	play	just
Topic 8	armenian	right	people	state	government	turkish	war	write	israeli	say	israel	article	arab	muslim	subject
Topic 9	car	write	line	article	subject	organization	good	just	post	bike	nntp	host	look	think	dod

As 15 principais palavras-chave do tópico

19. Como prever os tópicos para um novo pedaço de texto?

Supondo que você já tenha construído o modelo de tópico, é necessário conduzir o texto pela mesma rotina de transformações e antes de prever o tópico.

Para o nosso caso, a ordem das transformações é:

sent_to_words() -> -> -> lemmatization() vectorizer.transform() best_lda_model.transform()

Você precisa aplicar essas transformações na mesma ordem. Então, para simplificá-lo, vamos combinar essas etapas em uma função. predict_topic()

```
 # Define function to predict topic for a given text document.
nlp = spacy.load('en', disable=['parser', 'ner'])

def predict_topic(text, nlp=nlp):
    global sent_to_words
    global lemmatization

    # Step 1: Clean with simple_preprocess
    mytext_2 = list(sent_to_words(text))

    # Step 2: Lemmatize
    mytext_3 = lemmatization(mytext_2, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

    # Step 3: Vectorize transform
    mytext_4 = vectorizer.transform(mytext_3)

    # Step 4: LDA Transform
    topic_probability_scores = best_lda_model.transform(mytext_4)
    topic = df_topic_keywords.iloc[np.argmax(topic_probability_scores), :].values.tolist()
    return topic, topic_probability_scores

# Predict the topic
mytext = ["Some text about christianity and bible"]
topic, prob_scores = predict_topic(text = mytext)
print(topic)
```

```
['say', 'god', 'people', 'write', 'think', 'know', 'believe', 'christian', 'make', 'subject', 'lir
```

mytext foi alocado para o tópico que possui palavras-chave relacionadas à religião e ao cristianismo, o que é bastante significativo e faz sentido.

20. Como agrupar documentos que compartilham tópicos e plotagem semelhantes?

Você pode usar o agrupamento k-means na matriz de probabilidades do tópico do documento, que nada mais é do que um lda_output objeto. Desde o melhor modelo tem 15 conjuntos,