

 Instituto Infnet	Avaliação	Nota:
		Visto do Professor:
MIT em Inteligência Artificial, Machine Learning e Deep Learning		
Nome	Mateus Teixeira Ramos da Silva	
Link do repositório	<a href="https://github.com/GitMateusTeixeira/ml_clustering/tree/main/05_infnet_nlp_pd">https://github.com/GitMateusTeixeira/ml_clustering/tree/main/05_infnet_nlp_pd</a>	
Matéria	Processamento de Linguagem Natural com Python	
Prazo	02.06.2025	

## Parte 1. Implementar técnicas de lematização

1. Qual o endereço do seu notebook (colab) executado? Use o botão de compartilhamento do colab para obter uma url?

R: Foi utilizado um notebook no VSCode, cujo arquivo se encontra no repositório do GitHub:

[https://github.com/GitMateusTeixeira/ml\\_clustering/tree/main/05\\_infnet\\_nlp\\_pd](https://github.com/GitMateusTeixeira/ml_clustering/tree/main/05_infnet_nlp_pd).

2. Em qual célula está o código que realiza o download dos pacotes necessários para tokenização e stemming usando nltk?

R: Os pacotes necessários foram instalados através do 'pip tools', com os comandos 'pip-compile requirements.in' e 'pip-sync requirements.txt'. A instalação do 'pip tools' se encontra na 'Parte 4' do código (célula 17):

## Parte 4. Instalar o pip tools e importar todas as dependencias necessarias

[Voltar ao início](#)

```
[17] !pip install pip-tools

... Collecting pip-tools
      Using cached pip_tools-7.4.1-py3-none-any.whl.metadata (26 kB)
Collecting build>=1.0.0 (from pip-tools)
      Using cached build-1.2.2.post1-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: click>=8 in c:\programdata\anaconda3\envs\env_25e2_2\lib\site-packages (from pip-tools) (8.1.8)
Requirement already satisfied: pip>=22.2 in c:\programdata\anaconda3\envs\env_25e2_2\lib\site-packages (from pip-tools) (25.1)
Collecting pyproject-hooks (from pip-tools)
      Using cached pyproject_hooks-1.2.0-py3-none-any.whl.metadata (1.3 kB)
Requirement already satisfied: setuptools in c:\programdata\anaconda3\envs\env_25e2_2\lib\site-packages (from pip-tools) (78.1.1)
Requirement already satisfied: wheel in c:\programdata\anaconda3\envs\env_25e2_2\lib\site-packages (from pip-tools) (0.45.1)
Requirement already satisfied: packaging>=19.1 in c:\programdata\anaconda3\envs\env_25e2_2\lib\site-packages (from build>=1.0.0->pip-tools) (25.0)
Requirement already satisfied: colorama in c:\programdata\anaconda3\envs\env_25e2_2\lib\site-packages (from build>=1.0.0->pip-tools) (0.4.6)
Using cached pip_tools-7.4.1-py3-none-any.whl (61 kB)
Using cached build-1.2.2.post1-py3-none-any.whl (22 kB)
Using cached pyproject_hooks-1.2.0-py3-none-any.whl (10 kB)
Installing collected packages: pyproject-hooks, build, pip-tools

----- 2/3 [pip-tools]
----- 3/3 [pip-tools]
```

3. Em qual célula está o código que atualiza o spacy e instala o pacote `pt_core_news_lg`?

R: O spaCy se trata de uma biblioteca de Processamento de Linguagem Natural (PLN), e o modelo 'pt\_core\_news\_lg' é treinado especificamente para textos em português, permitindo tokenização, lematização e reconhecimento de entidades nomeadas. O código que realiza essa atualização e instalação pode ser encontrado na 'Parte 8' do notebook (células 70, 71, 4 e 5):

## Parte 8. Atualizar o SPACY, instalar e importar o modelo `pt_core_news_lg`

[Voltar ao início](#)

```
!pip install --upgrade spacy
```

```
[70] ✓ 9.7s Python  
Requirement already satisfied: spacy in c:\programdata\anaconda3\envs\env_25e2_2\lib\site-packages (3.8.4)  
Collecting spacy  
  Downloading spacy-3.8.7-cp311-cp311-win_amd64.whl.metadata (28 kB)  
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in c:\programdata\anaconda3\envs\env_25e2_2\lib\site-packages (from spacy) (3.0.12)
```

```
!python -m spacy download pt_core_news_lg
```

```
[71] ✓ 17.3s Python  
----- 356.5/568.2 MB 57.6 MB/s eta 0:00:04  
----- 367.0/568.2 MB 57.6 MB/s eta 0:00:04  
----- 379.8/568.2 MB 57.0 MB/s eta 0:00:04  
----- 393.5/568.2 MB 57.0 MB/s eta 0:00:04  
You can now load the package via spacy.load('pt_core_news_lg')
```

```
os.system("python -m spacy download pt_core_news_lg")
```

```
[4] ✓ 13.3s Python
```

```
nlp = spacy.load("pt_core_news_lg")
```

```
[5] ✓ 3.2s Python
```

#### 4. Em qual célula está o download dos dados diretamente do kaggle?

R: O código baixa os dados do kaggle, via API, para a pasta '../data/01-raw/' na “Parte 6” do notebook (célula 81):

## Parte 6. Baixar os dados

[Voltar ao início](#)

Instale o gerenciador kaggle no ambiente do Colab e faça o upload do arquivo kaggle.json

```
[81] ✓ 14.1s Python
# Criar o diretório se não existir
data_path = '../data/01-raw'
os.makedirs(data_path, exist_ok=True)

# Baixar os dados do Kaggle
kaggle.api.dataset_download_files('marlesson/news-of-the-site-folhauol', path=data_path, unzip=True)

... Dataset URL: https://www.kaggle.com/datasets/marlesson/news-of-the-site-folhauol
```

#### 5. Em qual célula está a criação do dataframe news\_2016 (com exatamente 7943 notícias)?

R: O dataframe 'news\_2016' foi criado através do filtro do dataframe original na 'Parte 10' do notebook (células 7 e 95):

## Parte 10. Filtrar os dados

[Voltar ao início](#)

Filtre os dados do DataFrame df e crie um DataFrame news\_2016 que contenha apenas notícias de **2016** e da categoria **mercado**.

```
[7] ✓ 0.0s Python
df['date'] = pd.to_datetime(df.date)

# Create a dataframe named news_2016
news_2016 = df[
    (df['date'].dt.year == 2016) &
    (df['category'] == 'mercado')
]

[95] Python
news_2016.shape

... (7943, 6)
```

6. Em qual célula está a função que tokeniza e realiza o stemming dos textos usando funções do nltk?

R: A tokenização divide o texto em palavras individuais, enquanto o stemming reduz cada palavra ao seu radical, removendo sufixos e tornando a análise mais eficiente. A função responsável por essas operações está na 'Parte 11' do notebook (células 8 e 9):

## Parte 11. Criar as colunas NLTK Tokenizer e Stemmer

[Voltar ao início](#)

Crie uma coluna no dataframe `news_2016` contendo os tokens para cada um dos textos. Os tokens devem estar representados pelo radical das palavras (stem).

Para tal, complete o conteúdo da função `tokenize`.

```
def tokenize(text: str) -> List[str]:
    """
    Function for tokenizing using `nltk.tokenize.word_tokenize`

    Returns:
    - A list of stemmed tokens (`nltk.stem.RSLPStemmer`)
    IMPORTANT: Only tokens with alphabetic
    | | | | | characters will be returned.
    """
    stemmer = RSLPStemmer()

    # Tokenizar o texto
    text = str(text).lower()
    tokens = word_tokenize(text)

    # Aplicar stemming apenas em tokens alfabéticos
    stemmed_tokens = [stemmer.stem(token) for token in tokens if token.isalpha()]

    return stemmed_tokens
```

[8] ✓ 0.0s Python

```
news_2016.loc[:, 'nltk_tokens'] = news_2016['text'].progress_map(tokenize)
```

[9] ✓ 1m 26.6s Python

... 100% | 7943/7943 [01:26<00:00, 91.72it/s]

## 7. Em qual célula está a função que realiza a lematização usando o spacy?

R: Essa etapa do notebook garante que as palavras sejam reduzidas à sua forma base, o que melhora a análise de texto e facilita tarefas como categorização e mineração de informações. A função responsável por essa operação está na 'Parte 13' do notebook (célula 61):

## Parte 13. Realizar a Lemmatização com o SPACY

[Voltar ao início](#)

O modelo NLP do spacy oferece a possibilidade de lematizar textos em português (o que não acontece com a biblioteca NLTK).

Iremos criar uma lista de tokens lematizados para cada texto do nosso dataset. Para tal, iremos retirar as stopwords, usando uma função que junta stopwords provenientes do NLTK e do Spacy. Essa lista completa, é retornada pela função stopwords (e você não precisa mexer).

Já a função filter retorna True caso o token seja composto por caracteres alfabéticos, não estiver dentro da lista de stopwords e o lemma resultante não estiver contido na lista "o", "em", "em o", "em a" e "ano".

Crie uma coluna chamada `spacy_lemma` para armazenar o resultado desse pré-processamento.

```
def stopwords() -> Set:
    """
    Return complete list of stopwords
    """
    return set(list(nltk.corpus.stopwords.words("portuguese")) + list(STOP_WORDS))

complete_stopwords = stopwords()

def filter(w: spacy.lang.pt.Portuguese) -> bool:
    """
    Filter stopwords and undesired tokens
    """
    undesired_lemmas = {
        "o", "em", "em o", "em a", "ano", "g", "s", "p", "afirmar", "dizer", "hoje", "ser", "dia",
        "chegar", "haver", "ficar", "apesar", "criar", "ligar", "voltar", "fazer", "esperar", "voltar"
    }
    return w.is_alpha and w.text.lower() not in complete_stopwords and w.lemma_ not in undesired_lemmas

def lemma(doc: spacy.lang.pt.Portuguese) -> List[str]:
    """
    Apply spacy lemmatization on the tokens of a text
    """
    return [token.lemma_ for token in doc if filter(token)]

news_2016.loc[:, 'spacy_lemma'] = news_2016.spacy_doc.progress_map(lemma)
```

[61]

✓ 4.7s

Python

... 0%| | 0/7943 [00:00<?, ?it/s]

100%| 7943/7943 [00:04<00:00, 1672.64it/s]

8. Baseado nos resultados qual a diferença entre stemming e lematização, qual a diferença entre os dois procedimentos? Escolha quatro palavras para exemplificar.

R: A técnica de stemming busca reduzir palavras ao seu radical, removendo seus sufixo, porém sem considerar regras linguísticas. Isso pode gerar formas truncadas que não correspondem exatamente à palavra original. Por outro lado, a lematização utiliza uma biblioteca bem mais robusta, capaz de transformar palavras na sua forma base correta, levando em conta o contexto e regras gramaticais.

Isso se mostra de forma clara quando comparamos as colunas 'spacy\_doc', 'nltk\_tokens' e 'spacy\_lemma' e fica ainda mais evidentes em substantivos e nomes próprios (onde a lematização não reduz nada) ou verbos (onde a lematização leva o verbo para o seu infinitivo) como:

Original	Stemming	Lematização
'oposição'	'opos'	'oposição'
'Calheiros'	'calh'	'Calheiros'
'senso'	'sens'	'senso'
'informou'	'inform'	'informar'

<pre>news_2016[['spacy_doc', 'nltk_tokens', 'spacy_lemma']].sample(5)</pre>				Python	
...					
	spacy_doc	nltk_tokens	spacy_lemma		
85719	(O, grupo, de, alimentação, francês, Danone, p...	[o, grup, de, aliment, franc, danon, prev, nov...	[alimentação, francês, Danone, previu, melhora...		
86951	(O, presidente, do, Senado, ,, Renan, Calheiro...	[o, presid, do, sen, renan, calh, confirm, nes...	[presidente, senado, Renan, Calheiros, confirm...		
62418	(O, senso, comum, faz, pensar, que, no, inver...	[o, sens, comum, faz, pens, que, no, inver, a...	[senso, comum, pensar, inverno, pequeno, médio...		
63110	(Com, uma, oposição, cada, vez, mais, fraca, ,...	[com, uma, opos, cad, vez, mais, frac, o, gove...	[oposição, fraco, governo, Mauricio, Macri, co...		
48523	(A, Petrobras, informou, nesta, sexta-feira, (...)	[a, petrobr, inform, nest, que, seu, conselh, ...	[Petrobras, informar, administração, aprovar, ...		

## Parte 2. Construir um modelo de reconhecimento de entidades (NER) usando Spacy

9. Em qual célula o modelo `pt_core_news_lg` está sendo carregado? Todos os textos do dataframe precisam ser analisados usando os modelos carregados. Em qual célula isso foi feito?

R: O '`pt_core_news_lg`' é um modelo do 'spaCy' treinado para análise de texto em português. Ele permite tokenização, lematização e reconhecimento de entidades nomeadas.

Nessa etapa, todos os textos do dataframe são analisados utilizando o modelo carregado, de modo a estruturar e preparar as informações para as análises futuras.

O carregamento do modelo '`pt_core_news_lg`' foi feito no final da 'Parte 8' do notebook (células 4 e 5):

## Parte 8. Atualizar o `SPACY`, instalar e importar o modelo `pt_core_news_lg`

[Voltar ao início](#)

```
[4] os.system("python -m spacy download pt_core_news_lg")
✓ 13.3s Python

... 0

[5] nlp = spacy.load("pt_core_news_lg")
✓ 3.2s Python
```

A aplicação do modelo ‘pt\_core\_news\_lg’ sobre os textos foi feito na ‘Parte 12’ do notebook (célula 10):

## Parte 12. Criar um documento `SPACY` dentro de cada texto do dataset

[Voltar ao início](#)

Crie uma coluna `spacy_doc` que contenha os objetos spacy para cada texto do dataset de interesse. Para tal, carregue os modelos `pt_core_news_lg` e aplique em todos os textos (pode demorar alguns minutos...)

```
# Aplicar o processamento aos textos
news_2016.loc[:, 'spacy_doc'] = news_2016['text'].astype(str).progress_apply(nlp)

[10] ✓ 8m 45.8s Python

... 100% | 7943/7943 [08:45<00:00, 15.11it/s]
```

**10. Indique a célula onde as entidades dos textos foram extraídas. Estamos interessados apenas nas organizações.**

R: Nessa etapa, o modelo ‘pt\_core\_news\_lg’ identifica e classifica entidades dentro do texto, permitindo que apenas aquelas com a etiqueta "ORG" sejam extraídas.

A função ‘NER()’ percorre os textos já processados e coleta nomes de empresas, instituições e organizações mencionadas nos documentos, armazenando os resultados na coluna ‘spacy\_ner’ do dataframe. Essa operação está localizada na ‘Parte 14’ do notebook (célula 62):







### Parte 3. Criar modelos utilizando vetorização de textos baseado em Bag of Words

---

**12. Quando adotamos uma estratégia frequentista para converter textos em vetores, podemos fazê-lo de diferentes maneiras. Mostramos em aula as codificações One-Hot, TF e TF-IDF. Explique a principal motivação em adotar TF-IDF frente as duas outras opções.**

R: O TF-IDF (Term Frequency-Inverse Document Frequency) foi adotado dentre às codificações One-Hot (One-Hot Encoding) e TF (Term Frequency) pois, além de considerar a frequência dos termos em um documento, ele ajusta seu peso com base na presença nos demais documentos do conjunto.

Isso evita que palavras muito comuns dominem a análise, destacando termos mais relevantes e específicos.

Enquanto One-Hot apenas indica a presença de uma palavra e o TF conta sua frequência sem distinção de importância, o TF-IDF equilibra esses fatores, tornando-se uma abordagem mais eficaz para classificação de textos, buscas e mineração de dados em Processamento de Linguagem Natural.

Um exemplo clássico desse fenômeno é a palavra "oportunidade".

Se um documento menciona "oportunidade" muitas vezes, o TF atribuirá um alto peso a essa palavra.

No entanto, se "oportunidade" também aparecer frequentemente em todos os documentos do conjunto, seu impacto como indicador específico diminui.

O TF-IDF ajusta esse peso, reduzindo sua relevância, pois é uma palavra comum no corpus. Assim, palavras mais específicas, como "investimento" ou "parceria estratégica", podem ter um peso maior por aparecerem menos frequentemente em outros documentos, tornando a análise mais precisa.

**13. Indique a célula onde está a função que cria o vetor de TF-IDF para cada texto.**

R: Nessa etapa, os textos são convertidos em vetores de TF-IDF. Essa técnica se presta a representar documentos numericamente em Processamento de Linguagem Natural. O TF-IDF atribui pesos às palavras, destacando aquelas mais importantes para cada documento ao reduzir a influência de termos muito frequentes em todo o 'corpus'.

A classe 'Vectorizer' processa os textos tokenizados e aplica o 'TfidfVectorizer' do sklearn, limitando o número de características ('max\_features=5000') e filtrando termos com baixa frequência ('min\_df=10').

A função 'tokens2tfidf()' garante que cada conjunto de tokens seja corretamente transformado em um vetor numérico. Essa operação está localizada na 'Parte 15' do notebook (célula 63):

## Parte 15. Bag of Words

[Voltar ao início](#)

Crie uma coluna `tfidf` no dataframe `news_2016`. Use a coluna `spacy_lemma` como base para cálculo do TFIDF. O número máximo de features que iremos considerar é 5000. E o token, tem que ter aparecido pelo menos 10 vezes (`min_df`) nos documentos.

```
class Vectorizer:
    def __init__(self, doc_tokens: List[List[str]]):
        """
        Inicializa o vetor TF-IDF com os documentos tokenizados.
        """
        # Transformando listas de tokens em strings
        self.doc_tokens = [" ".join(tokens) for tokens in doc_tokens]
        self.tfidf = None

    def vectorizer(self) -> TfidfVectorizer:
        """
        Converte uma lista de tokens em vetores TF-IDF e retorna o objeto `TfidfVectorizer`.
        """
        vectorizer = TfidfVectorizer(max_features=5000, min_df=10)
        self.tfidf = vectorizer.fit(self.doc_tokens)
        return self.tfidf # Retorna o objeto treinado

    def __call__(self) -> TfidfVectorizer:
        if self.tfidf is None:
            return self.vectorizer()
        return self.tfidf

# Criar vetor TF-IDF com base na coluna `spacy_lemma`
doc_tokens = news_2016.spacy_lemma.values.tolist()
vectorizer = Vectorizer(doc_tokens)

# Função para converter tokens em vetor TF-IDF
def tokens2tfidf(tokens: List[str]):
    tokens_str = " ".join(tokens) # Garantindo que seja uma string
    array = vectorizer().transform([tokens_str]).toarray()[0]
    return array

news_2016.loc[:, 'tfidf'] = news_2016.spacy_lemma.progress_map(tokens2tfidf)
```

[63] ✓ 6.8s

Python

... 100%| 7943/7943 [00:05<00:00, 1343.97it/s]

14. Indique a célula onde estão sendo extraídos os tópicos usando o algoritmo de LDA.

R: O LDA (Latent Dirichlet Allocation) é uma técnica estatística que identifica padrões ocultos nos textos, agrupando palavras que aparecem frequentemente juntas em tópicos distintos.

O código converte os textos em vetores TF-IDF (corpus) e aplica o modelo LDA, configurado para identificar 9 (N\_TOKENS=9) tópicos. Com max\_iter=100, o algoritmo realiza múltiplas iterações para refinar as atribuições de tópicos, garantindo uma melhor separação entre os temas identificados.

Esse processo foi realizado na Parte 16 do notebook (célula 64):

## Parte 16. Extrair os tópicos

[Voltar ao início](#)

Realize a extração de 9 tópicos usando a implementação do sklearn do algoritmo Latent Dirichlet Allocation. Como parâmetros, você irá usar o número máximo de iterações igual à 100 (pode demorar) e o `random_seed` igual a `SEED` que foi setado no início do notebook

```
corpus = np.array(news_2016.tfidf.tolist())

lda = LDA(n_components=N_TOKENS, max_iter=100, random_state=SEED)
lda.fit(corpus)
```

[64]

✓ 1m 48.8s

Python

**15. Indique a célula onde a visualização LDAVis está criada.**

R: Ponto ignorado conforme convencionado pelo Professor em aula.

**16. Cole a figura com a nuvem de palavras para cada um dos 9 tópicos criados.**

R: As nuvens de palavras de cada tópico foram plotadas na 'Parte 18' do notebook (célula 67). Segue a figura das nuvens:

A word cloud visualization of terms related to social security and labor in Brazil. The words are arranged in a circular pattern, with the most prominent terms in the center and smaller words towards the edges. The colors are primarily shades of blue and green. The most prominent words include "segurado", "benefício", "trabalhador", "aposentado", "INSS", "contribuição", "direito", "reforma", "integral", "fator", "salário", "trabalho", "pequeno", "idade", "médica", "salarial", "media", "trabalho", "contribuição", "integral", "fator", "salário", "trabalho", "pequeno", "idade", "médica", "salarial", "media". Other visible words include "whatsapp", "previdenciário", "aposentadoria", "mulher", "acordo", "data", "completo", "julho", "receber", "poder", "dados", "reforma", "integral", "fator", "salário", "trabalho", "pequeno", "idade", "médica", "salarial", "media", "trabalho", "contribuição", "integral", "fator", "salário", "trabalho", "pequeno", "idade", "médica", "salarial", "media".

[illegible][illegible]

A word cloud visualization of the text "Receita Federal - Imposto de Renda Pessoa Física". The words are arranged in a dense, overlapping manner, with the most prominent words being "Receita Federal", "Imposto de Renda", and "Pessoa Física". Other visible words include "Receita", "Federal", "Imposto", "Renda", "Pessoa", "Física", "Receita Federal", "Imposto de Renda", "Pessoa Física", "Receita", "Federal", "Imposto", "Renda", "Pessoa", "Física". The colors are primarily shades of blue, green, and yellow, with some red and orange accents. The background is a light, textured surface.

[illegible]

A word cloud visualization showing various terms associated with the debate series. The most prominent words are "Diálogos", "Centro Brasileiro de Análise e Planejamento", "Folha", "conjuntura econômica", "debate", "Brasil", "análise", "planejamento", "realizar", "mercado", "folha", "conjuntura", "diálogo", "auditoria", "ciclo", "considerar", "influência", "tema", "conjuntura", "Mariana Paulo", "mediador", "Alvaro Fagundes", "Planejamento Folha", "Brasil", "conjuntura econômica", "conjuntura", "debate", "entrada gratuito". Other visible words include "debate série", "deputado Mateus", "econômico Brasil", "política pública", "uso coletivo", "mercado debate", "Vila Mariana", "objetivo discutir", "Milton Lima", "tema", "Alvaro Fagundes", "Planejamento Folha", "Brasil", "conjuntura econômica", "conjuntura", "debate", "entrada gratuito".

[illegible]

O quinto tópico apresenta palavras como ‘imposto renda’, ‘lote restituição’ e ‘malha fina’, indicando que se trata da declaração do imposto de renda e do direito à restituição.

O sexto tópico destaca termos como ‘China’, ‘Xangai’, ‘Shenzhen’, ‘Nikkei’, ‘mercado’, ‘econômico’ e ‘avançar’, sugerindo o tema do avanço econômico da China. O sétimo tópico evidencia palavras como ‘Rodolfo Hoffman’, ‘Folha realizar’, ‘Morgado Mateus’, ‘Vila Mariana’ e ‘conjectura econômico’, indicando um debate promovido pelo político Rodolfo Hoffman, que ocorrerá em São Paulo sobre conjectura econômica.

O oitavo tópico traz termos como ‘Brasil’, ‘China’, ‘Estados Unidos’, ‘mercado’ e ‘acordo’, evidenciando tratados econômicos entre países e seus reflexos no consumidor e na população.

No geral, as palavras possuem sentido semântico suficiente para reconhecer o conteúdo, ainda que haja algumas palavras truncadas (como ‘paí-’ - provavelmente referente a ‘país’ ou ‘países’), mas que podem ser compreendidas pelo contexto.

#### **Parte 4. Criar modelos baseados em Word Embedding**

---

**18. Neste projeto, usamos TF-IDF para gerar os vetores que servem de entrada para o algoritmo de LDA. Quais seriam os passos para gerar vetores baseados na técnica de Doc2Vec?**

R: Para gerar vetores utilizando a técnica de Doc2Vec, primeiro é necessário pré-processar os textos, removendo caracteres indesejados, stopwords e aplicando tokenização.

Em seguida, cada documento precisa ser convertido em uma lista de palavras. Com os textos preparados, inicializamos o modelo Doc2Vec do ‘gensim’, definindo parâmetros como o tamanho dos vetores e a janela de contexto.

O próximo passo é treinar o modelo usando os documentos representados como ‘TaggedDocument’, onde cada texto recebe um identificador único. Após o treinamento, podemos gerar vetores para novos documentos transformando-os em tokens e utilizando o método ‘infer\_vector()’ do modelo treinado.

Esses vetores podem então ser usados como entrada para algoritmos de aprendizado de máquina ou análise semântica, permitindo capturar relações mais profundas entre os textos.

**19. Em uma versão alternativa desse projeto, optamos por utilizar o algoritmo de K-Médias para gerar os clusters (tópicos). Qual das abordagens (TF-IDF ou Doc2Vec) seria mais adequada como processo de vetorização? Justifique com comentários sobre dimensionalidade e relação semântica entre documentos.**

R: A escolha entre TF-IDF e Doc2Vec como processo de vetorização depende do objetivo da clusterização com K-Médias.

O TF-IDF gera vetores de alta dimensionalidade, onde cada dimensão representa uma palavra do vocabulário. Assim, a separação dos clusters irá se basear na frequência dos termos.

Essa abordagem é eficiente para identificar diferenças entre documentos por meio da ocorrência de palavras-chave, mas pode perder contexto semântico e ser impactada pela alta dispersão dos dados.

Já o Doc2Vec cria vetores densos de dimensões reduzidas, capturando a relação semântica entre textos, permitindo que documentos semelhantes sejam agrupados mesmo sem compartilharem palavras exatas. Como os vetores gerados pelo modelo refletem o significado e a estrutura dos textos, ele permite uma análise mais profunda e conexões mais naturais entre os dados.

Como o K-Médias depende de uma boa métrica de similaridade, o Doc2Vec tende a ser mais adequado, pois representa cada documento como um vetor contínuo que considera contexto e significado. Isso melhora a qualidade dos clusters ao reconhecer padrões além da frequência das palavras, tornando a segmentação mais precisa e interpretável.

**20. Leia o artigo "Introducing our Hybrid lda2vec Algorithm" (<https://multithreaded.stitchfix.com/blog/2016/05/27/lda2vec/#topic=38&lambda=1&term=>).**

**O algoritmo lda2vec pretende combinar o poder do word2vec com a interpretabilidade do algoritmo LDA. Em qual cenário o autor sugere que há benefícios para utilização deste novo algoritmo? Indique um caso de uso para essa solução projetada.**

R: A combinação proposta pelo algoritmo 'lda2vec' permite que os tópicos gerados sejam mais coerentes e semanticamente ricos. O autor sugere que essa abordagem é especialmente útil em cenários onde é necessário interpretar tópicos de maneira mais intuitiva, sem perder a capacidade de capturar relações contextuais entre palavras.

Um caso de uso relevante para essa solução é a análise de feedback de clientes em grandes empresas. Com 'lda2vec', é possível identificar tópicos recorrentes nas avaliações dos consumidores, garantindo que os temas extraídos sejam interpretáveis e, ao mesmo tempo, preservem nuances semânticas. Isso permite que empresas ajustem seus produtos e serviços com base em insights mais precisos sobre as opiniões dos clientes.