

Relatório Final - Projeto de Disciplina de Processamento de Linguagem Natural com Python

Este relatório detalha as respostas às questões propostas no trabalho final da disciplina, complementando o notebook [projeto_nlp.ipynb](#) onde o código foi implementado e as análises foram realizadas.

Implementar técnicas de lematização

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

[clique aqui para ver o notebook do projeto](#)

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

O código que realiza o download dos pacotes `stopwords`, `punkt` e `rslp` do NLTK está na célula com o ID. O código adicionado foi:

```
import nltk
nltk.download("stopwords")
nltk.download("punkt")
nltk.download("punkt_tab")
nltk.download("rslp")
```

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

Código que atualiza a biblioteca Spacy e instala o modelo `pt_core_news_lg` para português:

```
!pip install -U spacy
!python -m spacy download pt_core_news_lg

import spacy
from spacy.lang.pt.stop_words import STOP_WORDS
```

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

Código responsável por baixar o dataset diretamente do Kaggle usando a API:

```
!kaggle datasets download --force -d marlesson/news-of-the-site-folhauol
```

Em qual célula está a criação do dataframe news_2016 (com examente 7943 notícias)?

Código de criação do dataframe `news_2016`, filtrando as notícias do ano de 2016 e da categoria "mercado":

```
df['date'] = pd.to_datetime(df.date)
news_2016 = df[(df["date"].dt.year == 2016) & (df["category"].str.lower() == "mercado")].copy()
```

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

Função `tokenize` que realiza a tokenização e o stemming (usando `RSLPStemmer`) com NLTK:

```
from nltk.tokenize import word_tokenize
from nltk.stem import RSLPStemmer

def tokenize(text: str) -> List:
    stemmer = RSLPStemmer()
    tokens = word_tokenize(text.lower())
    stems = [stemmer.stem(token) for token in tokens if token.isalpha()]
    return stems

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

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

A lematização usando Spacy é realizada através de duas funções auxiliares (`filter` e `lemma`). A função `lemma` aplica a lematização e utiliza a função `filter` para remover stopwords e tokens indesejados:

Na função `filter`:

```
def filter(w: spacy.lang.pt.Portuguese) -> bool:
    return w.is_alpha and w.text.lower() not in complete_stopwords and
    w.lemma_ not in ['o', 'em', 'em o', 'em a', 'ano']
```

Na função `lemma`:

```
def lemma(doc: spacy.lang.pt.Portuguese) -> List[str]:
    lemmas = [w.lemma_ for w in doc if filter(w)]
    return lemmas
```

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

A principal diferença entre stemming (radicalização) e lematização reside na abordagem para reduzir as palavras à sua forma base:

- **Stemming:** É um processo mais heurístico e rápido que remove sufixos (e às vezes prefixos) das palavras para obter um radical comum. O resultado nem sempre é uma palavra válida do dicionário. O objetivo é agrupar palavras com o mesmo significado conceitual, mesmo que a forma resultante não seja linguisticamente correta.
- **Lematização:** É um processo mais sofisticado e geralmente mais lento que utiliza análise de um dicionário para encontrar o lema de uma palavra. O resultado da lematização é sempre uma palavra válida do dicionário. Leva em consideração o contexto da palavra (classe gramatical, por exemplo) para determinar o lema correto.

A diferença é que a lematização produz palavras reais, enquanto o stemming pode produzir palavras não reais, focando apenas em truncar a palavra.

Exemplos (considerando o contexto do português e as bibliotecas usadas):

Palavra Original	Stemming (RSLPStemmer - NLTK)	Lematização (pt_core_news_lg - Spacy)
correndo	corr	correr
casas	cas	casa
falavam	fal	falar
felizmente	feliz	felizmente (ou feliz dependendo do contexto/modelo)

Nestes exemplos, vemos que o stemming (RSLP) reduz as palavras a radicais mais curtos (**corr**, **cas**, **feliz**), que não são necessariamente palavras completas. A lematização, por outro lado, retorna o infinitivo do verbo (**correr**, **falar**) ou a forma singular do substantivo (**casa**), que são formas válidas no dicionário.

Construir um modelo de reconhecimento de entidades (NER) usando Spacy

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?

O modelo **pt_core_news_lg** do Spacy é carregado na célula com ID. Nesta mesma célula, o modelo é aplicado a todos os textos da coluna **text** do dataframe **news_2016** para criar os documentos Spacy, que são armazenados na nova coluna **spacy_doc**. O código utilizado foi:

Código de carregamento do modelo **pt_core_news_lg** nesse mesmo trecho o modelo é aplicado a todos os textos da coluna **text** do dataframe **news_2016** para criar os documentos Spacy, que são armazenados na nova coluna **spacy_doc**.

```
nlp = spacy.load("pt_core_news_lg")
news_2016['spacy_doc'] = list(nlp.pipe(news_2016['text']))
```

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

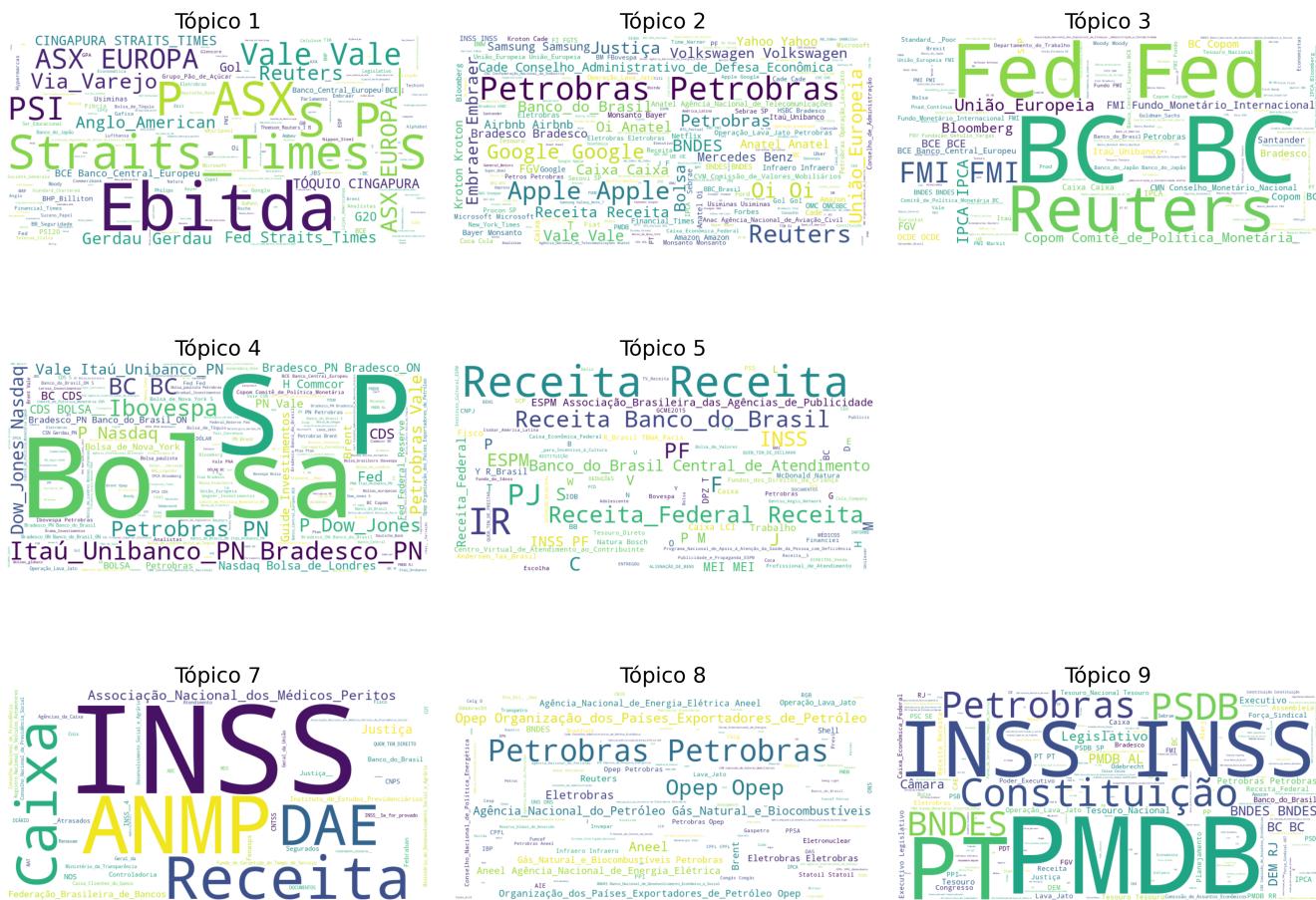
A extração das entidades do tipo "Organização" (ORG) é realizada pela função **NER**. Esta função processa cada documento Spacy (da coluna **spacy_doc**) e extrai apenas as entidades rotuladas como **ORG**. O resultado é armazenado na coluna **spacy_ner**. O código preenchido na função foi:

```
def NER(doc: spacy.lang.pt.Portuguese):
    organizations = [ent.text for ent in doc.ents if ent.label_ == 'ORG']
    return organizations
```

```
news_2016.loc[:, 'spacy_ner'] = news_2016.spacy_doc.progress_map(NER)
```

Cole a figura gerada que mostra a nuvem de entidades para cada tópico obtido (no final do notebook)

A figura com a nuvem de entidades para cada um dos 9 tópicos



Criar modelos utilizando vetorização de textos baseado em Bag of Words

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.

Foi utilizado TF-IDF em vez de One-Hot ou TF pela sua capacidade de ponderar a importância das palavras não apenas pela sua frequência dentro de um documento, mas também pela sua raridade em

toda a coleção de documentos (corpus).

- **One-Hot:** Representa cada palavra como um vetor esparsa onde apenas a posição correspondente à palavra é 1 e as outras são 0. Ignora a frequência da palavra no documento e a dimensionalidade cresce linearmente com o tamanho do vocabulário, tornando-se inviável para grandes corpus.
- **TF:** Conta a frequência de cada palavra em um documento. Dá mais peso a palavras que aparecem muitas vezes, mas pode supervalorizar palavras comuns (como artigos e preposições, mesmo após a remoção de stopwords) que aparecem frequentemente em muitos documentos, mas carregam pouca informação distinta sobre o conteúdo específico do documento.
- **TF-IDF:** Combina a frequência do termo (TF) com a frequência inversa do documento (IDF). O IDF mede quão comum ou rara uma palavra é em todo o corpus. Palavras que aparecem em muitos documentos terão um IDF baixo, enquanto palavras raras terão um IDF alto. Ao multiplicar TF por IDF, o TF-IDF atribui um peso maior a palavras que são frequentes em um documento específico, mas raras no corpus geral. Isso ajuda a destacar termos que são realmente importantes e distintivos para o conteúdo daquele documento, ao mesmo tempo que diminui o peso de palavras muito comuns e pouco informativas.

Portanto, o TF-IDF oferece uma representação vetorial mais significativa e discriminativa do conteúdo textual em comparação com One-Hot e TF puro, sendo útil para tarefas como classificação de texto, clustering e recuperação de informação.

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

A criação do vetor TF-IDF é realizada dentro da classe `Vectorizer`, no método `vectorizer`. O vetorizador `TfidfVectorizer` do Scikit-learn é inicializado e treinado (`fit`) com os tokens lematizados. A aplicação para gerar a coluna `tfidf` no dataframe usa a função `tokens2tfidf`. Foi excluído do docs tbm todas as palavras com menos de 3 dígitos. O código preenchido no método `vectorizer` foi:

```
def vectorizer(self):  
    docs = " ".join([t for t in tokens if len(t) >= 3]) for tokens in  
    self.doc_tokens]  
  
    self.tfidf = TfidfVectorizer(  
        lowercase=True,  
        stop_words=list(complete_stopwords),  
        max_features=5000,  
        min_df=10,  
        tokenizer=lambda x: x.split(),  
        preprocessor=lambda x: x,  
        ngram_range=(1, 2)  
    ).fit(docs)  
  
    return self.tfidf
```

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

Código de extração dos tópicos usando o algoritmo LDA do Scikit-learn. O modelo LDA é inicializado com 9 componentes (tópicos), 100 iterações máximas e a semente aleatória definida, e então treinado (`fit`) com a matriz TF-IDF (corpus). O código preenchido foi:

```
N_TOKENS = 9
```

```
corpus = np.stack(news_2016['tfidf'].values)
lda = LDA(n_components=N_TOKENS, max_iter=100, random_state=SEED)
lda.fit(corpus)
```

Indique a célula onde a visualização LDAVis está criada.

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

A figura com a nuvem de palavras para cada um dos 9 tópicos.



Escreva brevemente uma descrição para cada tópico extraído. Indique se você considera o tópico extraído semanticamente consistente ou não.

Tópico 1 – Indicadores de Mercado Internacional

Palavras-chave: Ebitda, ASX, Straits Times, Europa, PSI, Vale, Via Varejo

Descrição: Tópico abrange índices financeiros internacionais (como ASX e Straits Times) e empresas de capital aberto, sugerindo um foco em indicadores de desempenho corporativo global.

Consistência semântica: Alta.

Tópico 2 – Setor Automotivo e Petrobras

Palavras-chave: **Petrobras, Volkswagen, Mercedes, Oi, Apple, BNDES, Reuters**

Descrição: Mescla empresas de grande porte dos setores automotivo, telecomunicações e energia, com destaque para Petrobras e montadoras globais.

Consistência semântica: Moderada (abrange múltiplos setores).

Tópico 3 – Política Monetária Internacional

Palavras-chave: **Fed, BC, Reuters, FMI, União Europeia, Copom**

Descrição: Foco claro em instituições monetárias internacionais e decisões de política econômica de impacto global.

Consistência semântica: Alta.

Tópico 4 – Bolsa de Valores e Ações

Palavras-chave: **Bolsa, Ibovespa, SP, Nasdaq, Itaú, Petrobras PN, Bradesco PN**

Descrição: Representa discussões sobre ações e índices de bolsas nacionais e internacionais, incluindo empresas listadas como Itaú, Bradesco e Petrobras.

Consistência semântica: Alta.

Tópico 5 – Receita Federal e Tributação

Palavras-chave: **Receita, Receita Federal, IR, INSS, Banco do Brasil, PF, MEI**

Descrição: Tópico voltado para temas de fisco, tributos, pessoa física e jurídica, incluindo obrigações fiscais e declarações.

Consistência semântica: Alta.

Tópico 6 – Previdência e Justiça Social

Palavras-chave: **INSS, ANMP, DAE, Caixa, Justiça, Associação Nacional dos Médicos Peritos**

Descrição: Aponta para o universo da previdência social, benefícios, e perícias médicas, com ênfase em atores institucionais envolvidos.

Consistência semântica: Alta.

Tópico 7 – Energia e Petróleo

Palavras-chave: **Petrobras, Opep, Agência Nacional de Petróleo, Gás Natural, Biocombustíveis, Eletrobras**

Descrição: Foco no setor energético e petrolífero, com menção a empresas e órgãos reguladores como ANP e Opep.

Consistência semântica: Alta.

Tópico 8 – Previdência e Reforma Política

Palavras-chave: INSS, PMDB, PT, PSDB, Petrobras, Constituição, BNDES

Descrição: Mistura temas de previdência e política, com foco em reformas constitucionais, partidos políticos e instituições como BNDES e Petrobras.

Consistência semântica: Moderada (mistura previdência com política partidária).

Tópico 9 – Governança e Partidos Políticos

Palavras-chave: PT, PMDB, PSDB, Petrobras, Constituição, governo, BNDES

Descrição: Este tópico trata de partidos políticos e governança, incluindo instituições públicas e empresas estatais.

Consistência semântica: Alta.

Criar modelos baseados em Word Embedding

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?

Para gerar vetores de documentos usando Doc2Vec, os passos seriam os seguintes:

1. **Preparação dos Dados:** Assim como no TF-IDF, precisaríamos dos textos pré-processados (tokenizados, possivelmente lematizados ou stemizados, e com remoção de stopwords). No entanto, o Doc2Vec requer um formato específico: uma lista onde cada elemento é um objeto `TaggedDocument` da biblioteca Gensim. Cada `TaggedDocument` contém a lista de tokens do documento e uma tag única (que pode ser um índice numérico ou uma string identificadora do documento).
2. **Instanciação do Modelo:** Criar uma instância do modelo `Doc2Vec` da biblioteca Gensim. Seria necessário definir parâmetros como `vector_size` (a dimensionalidade dos vetores resultantes), `window` (o tamanho da janela de contexto), `min_count` (ignorar palavras com frequência total menor que este valor), `workers` (número de threads para treinamento), `epochs` (número de iterações sobre o corpus), e o algoritmo a ser usado (`dm=1` para PV-DM ou `dm=0` para PV-DBOW).
3. **Construção do Vocabulário:** Chamar o método `build_vocab()` no modelo Doc2Vec, passando a lista de `TaggedDocument`s. Isso constrói o vocabulário interno do modelo.
4. **Treinamento do Modelo:** Chamar o método `train()` no modelo Doc2Vec, passando novamente a lista de `TaggedDocument`s, o número total de exemplos (`total_examples=model.corpus_count`) e o número de épocas (`epochs=model.epochs`). Este passo treina os vetores das palavras e dos documentos.
5. **Extração dos Vetores:** Após o treinamento, os vetores dos documentos estão armazenados no atributo `model.dv` (ou `model.docvecs` em versões mais antigas do Gensim). Pode-se acessar o vetor de um documento específico pela sua tag. Para obter uma matriz com os vetores de todos os documentos na ordem original, seria necessário iterar pelas tags e recuperar os vetores correspondentes.

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

Escolha de Vetorização para K-Médias: TF-IDF vs. Doc2Vec

Optei por utilizar o algoritmo de **K-Médias (K-Means)** para gerar os clusters de documentos (tópicos). Para isso, é fundamental selecionar uma técnica de vetorização que se alinhe às características desse algoritmo.

Comparação entre TF-IDF e Doc2Vec:

TF-IDF (Term Frequency-Inverse Document Frequency)

- **Natureza dos vetores:** Vetores com muitos zeros e poucos valores diferentes e de alta dimensionalidade (uma dimensão por palavra ou n-grama).
- **Distância utilizada:** Funciona bem com distância euclidiana ou cosseno, que são compatíveis com a métrica usada no K-Médias.
- **Limitação:** Por não capturar o significado das palavras nem seu contexto, documentos semanticamente similares podem ter vetores diferentes se usarem vocabulário distinto.
- **Resultado:** Boa performance para clusters bem separados com vocabulário específico, mas limitada para identificar nuances semânticas.

Doc2Vec

- **Natureza dos vetores:** Vetores densos e de **dimensionalidade reduzida** (por exemplo, 100 a 300 dimensões), gerados por um modelo neural que leva em consideração o contexto das palavras.
- **Vantagem semântica:** Documentos com conteúdo semelhante, mesmo usando palavras diferentes, tendem a ter vetores próximos no espaço vetorial.
- **Adequação ao K-Médias:** Por gerar vetores mais compactos e semanticamente informativos, tende a produzir **clusters mais coerentes** quando há sobreposição de vocabulário ou temas mais subjetivos.

Conclusão

Para o uso com **K-Médias**, a vetorização com **Doc2Vec** é geralmente mais adequada do que TF-IDF. Isso se deve principalmente a dois fatores:

- **Menor dimensionalidade**, o que facilita o cálculo de centroides e reduz o custo computacional do algoritmo.
- **Capacidade de capturar relações semânticas**, permitindo que documentos com conteúdo semelhante sejam agrupados mesmo quando o vocabulário difere.

O TF-IDF pode ser útil quando os tópicos são muito distintos entre si em termos de vocabulário, mas é limitado em tarefas onde a **similaridade semântica é mais sutil**.

Leia o artigo "Introducing our Hybrid Ida2vec Algorithm"

(<https://multithreaded.stitchfix.com/blog/2016/05/27/Ida2vec/#topic=38&lambda=1&term=>). O algoritmo Ida2vec 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?

Ao ler o artigo "Introducing our Hybrid Ida2vec Algorithm", entendi que o Ida2vec foi proposto como uma forma de unir o poder semântico do word2vec com a interpretabilidade dos tópicos extraídos pelo LDA. O autor destaca que esse algoritmo é especialmente útil em contextos onde temos uma grande coleção de documentos e queremos identificar automaticamente temas relevantes, mas ao mesmo tempo manter uma

representação vetorial rica que capture relações semânticas entre palavras e documentos. Essa combinação me parece interessante principalmente porque permite representar palavras, tópicos e documentos no mesmo espaço vetorial, possibilitando análises mais profundas sem abrir mão da interpretabilidade dos resultados.

Autor

Desenvolvido por **Herbert Fenando Jarencio de Souza Martins**

 [GitHub Repository](#)