FATEC RUBENS LARA

LUCAS CONCEIÇÃO MARQUES

PONTOS TURÍSTICOS DA BAIXADA SANTISTA:

Análise de Similaridade do Cosseno

Santos
2025

Sumário

1 Introdução ao Tema

O tema consiste na análise de similaridade do cosseno aplicada aos pontos turísticos da Baixada Santista. Nesse contexto, o usuário terá descrito o tipo de local que deseja visitar. Com base em arquivos `.txt` armazenados em um arquivo ZIP, contendo informações sobre diferentes pontos turísticos, o sistema realizará a análise para indicar qual local apresenta maior similaridade com a preferência do usuário.

2 Desenvolvimento do Trabalho

2.1 Importação das Bibliotecas e Ferramentas

O programa se inicia com a importação das seguintes bibliotecas: **Numpy** (para manipulação de vetores e cálculos numéricos), **NLTK** (essencial para o Processamento de Linguagem Natural e utilizada para obter as *stopwords*). O comando `nltk.download('stopwords')` baixa o conjunto de dados necessário.

Em seguida, importam-se: **OS** (para interagir com o sistema operacional, manipulando caminhos de arquivo), **Zipfile** (para manipular arquivos ZIP), **TfidfVectorizer** (do `Sklearn`, para conversão de textos em vetores TF-IDF) e **cosine_similarity** (para o cálculo da similaridade do cosseno).

```
import numpy as np
import nltk
nltk.download('stopwords')
import os # Novo: para interagir com o sistema de arquivos
import zipfile # Novo: para manipular arquivos zip
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

Figura 1: Trecho inicial do código em Python com importações e setup.

3 1. Pontos Turísticos

3.1 1.1 Upload

Descrição: Nesse ponto, o módulo `files` (do Google Colab) permite interações com o navegador. O comando `files.upload()` executa um pop-up solicitando o upload do arquivo `pontos_turisticos.zip` e armazena seu conteúdo na variável `uploaded`.

```python
# 1.1 Comando para fazer o upload do arquivo 'pontos_turisticos.zip'
# Execute esta célula e selecione seu arquivo no pop-up.

from google.colab import files
uploaded = files.upload()
```

Figura 2: Trecho de código referente ao upload do arquivo ZIP.

3.2 1.2 Extrair Conteúdo do ZIP

Descrição: A variável `zip_filename` define o nome do arquivo ZIP esperado. A estrutura condicional verifica se o arquivo foi carregado com êxito. Se sim, `with zipfile.ZipFile(zip_filename, 'r')` abre o arquivo em modo de leitura e `zip_ref.extractall(` extrai todo o conteúdo do ZIP para o diretório atual. Caso ocorra erro, uma mensagem de aviso será exibida.

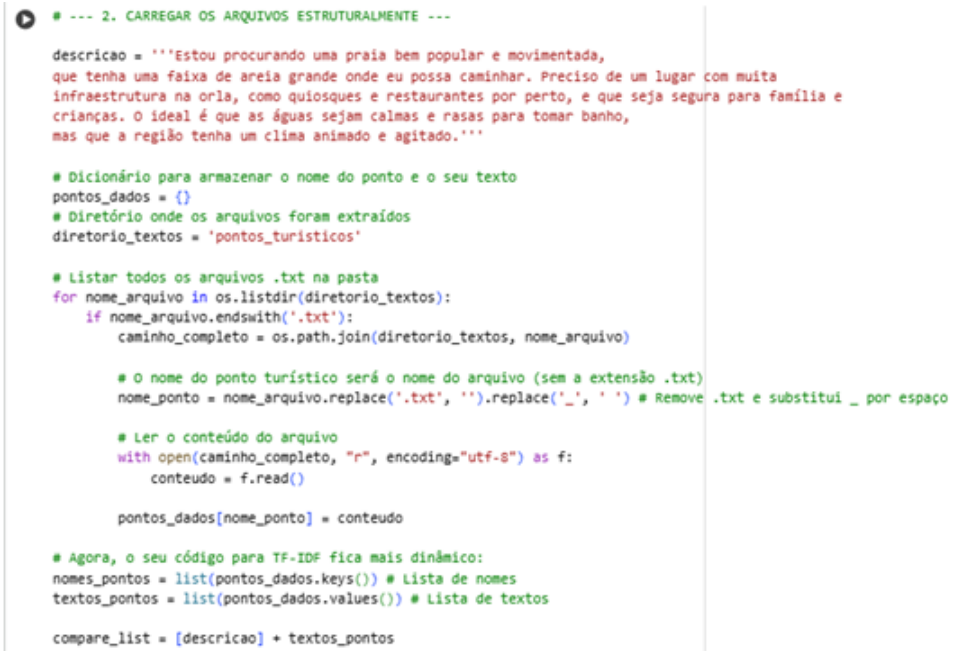
```python
# 1.2 Extrair o conteúdo do zip

zip_filename = 'pontos_turisticos.zip'
if zip_filename in uploaded:
  with zipfile.ZipFile(zip_filename, 'r') as zip_ref:
      zip_ref.extractall('.')
  print(f"Arquivos extraídos para o diretório atual")
else:
    print("ERRO: O arquivo zip não foi encontrado após o upload.")
# ------------------------------------------
```

Figura 3: Código para extração do conteúdo do arquivo ZIP.

3.3 1.3 Carregar os Arquivos

Descrição: Após a extração, define-se a variável `descrição`, que será a consulta (*query*) de busca. Cria-se um dicionário `pontos_dados` para armazenar o nome e o texto de cada ponto turístico. Um laço percorre os arquivos do diretório, filtrando apenas os com extensão `.txt`. A função `os.path.join()` constrói o caminho completo e o conteúdo é lido e armazenado. No dicionário, o nome do ponto é a chave e o texto é o valor. Listas são criadas com os nomes e textos, formando `compare_list`, que contém a descrição do usuário e os textos dos pontos turísticos.



```
# --- 2. CARREGAR OS ARQUIVOS ESTRUTURALMENTE ---

descricao = '''Estou procurando uma praia bem popular e movimentada,
que tenha uma faixa de areia grande onde eu possa caminhar. Preciso de um lugar com muita
infraestrutura na orla, como quiosques e restaurantes por perto, e que seja segura para família e
crianças. O ideal é que as águas sejam calmas e rasas para tomar banho,
mas que a região tenha um clima animado e agitado.'''

# Dicionário para armazenar o nome do ponto e o seu texto
pontos_dados = {}
# Diretório onde os arquivos foram extraídos
diretorio_textos = 'pontos_turisticos'

# Listar todos os arquivos .txt na pasta
for nome_arquivo in os.listdir(diretorio_textos):
    if nome_arquivo.endswith('.txt'):
        caminho_completo = os.path.join(diretorio_textos, nome_arquivo)

        # O nome do ponto turístico será o nome do arquivo (sem a extensão .txt)
        nome_ponto = nome_arquivo.replace('.txt', '').replace('_', ' ') # Remove .txt e substitui _ por espaço

        # Ler o conteúdo do arquivo
        with open(caminho_completo, "r", encoding="utf-8") as f:
            conteudo = f.read()

        pontos_dados[nome_ponto] = conteudo

# Agora, o seu código para TF-IDF fica mais dinâmico:
nomes_pontos = list(pontos_dados.keys()) # Lista de nomes
textos_pontos = list(pontos_dados.values()) # Lista de textos

compare_list = [descricao] + textos_pontos
```

Figura 4: Trecho do código responsável por carregar e ler os arquivos de texto.

4 Stopwords

As *stopwords* são importadas de `nltk.corpus`. Com `set()`, obtém-se a lista padrão em português. Em seguida, cria-se uma lista personalizada (`custom_stop`) com palavras genéricas (*local*, *visita*, etc.) e termos adicionais. A lista final `br_stop` é o resultado da união entre a lista padrão e a personalizada.

```
#lista com as stopwords do português

from nltk.corpus import stopwords

# 1. Pega a lista padrão de stopwords em português do NLTK
nltk_stop = set(stopwords.words('portuguese'))

custom_stop = [
    # Palavras comuns que NÃO agregam valor de busca
    'local', 'conhecer', 'visita', 'visitantes', 'arredores', 'visto', 'conta',
    'pode', 'podem', 'este', 'eu', 'ser', 'seja', 'pela', 'como', 'sendo',
    'km', '500',

    # Verbos conjugados e pronomes de busca que o NLTK pode ter omitido
    'estou', 'procurando', 'tenha', # do texto da descrição
    'onde', 'ter', 'possa', 'preciso', 'ideal', 'seja', 'para' ]

# 3. Combina as duas listas para o vetorizador
br_stop = list(nltk_stop.union(set(custom_stop)))
```

Figura 5: Código de criação da lista de *stopwords* padrão e personalizada.

5 Vetorização do Texto

Com as stopwords definidas, `TfidfVectorizer()` é usado para removê-las durante a vetorização. O método `fit_transform()` analisa e transforma os textos em vetores.

```
tfidf_vectorizer = TfidfVectorizer(stop_words=br_stop) #Remover as stop words

#vetorização dos textos
tf_idf_matrix = tfidf_vectorizer.fit_transform(compare_list)
```

Figura 6: Código responsável pela vetorização dos textos usando TF-IDF.

6 Impressão das Similaridades

A biblioteca `cosine_similarity` calcula a similaridade de cosseno entre os vetores da descrição do usuário e os dos pontos turísticos. O resultado é transformado em uma lista simples com `flatten()`. Com `numpy.arccos()` e `numpy.degrees()`, obtém-se o ângulo em graus. O programa imprime os nomes, similaridades e ângulos, indicando o ponto mais próximo do desejado.

```
#similaridade dos vetores da descrição e dos pontos
similaridades = cosine_similarity(tf_idf_matrix[0:1], tf_idf_matrix[1:]).flatten() #flatten transforma o resultado em uma lista simples

#veremos cada similaridade correspondente a cada nome dos pontos e seus angulos
for i, sim in enumerate(similaridades):
  nome = nomes_pontos[i]
  angulo = np.degrees(np.arccos(sim)) #arccos retorna angulo em radianos, degrees retorna eles em graus
  display(f"{nome}: Sim = {sim:.3f} | Ângulo = {angulo:.2f} graus")
```

Figura 7: Código para o cálculo da similaridade de cosseno entre o texto do usuário e os pontos turísticos.

7 Output

Analisando os dados obtidos, observa-se que o ponto turístico com maior taxa de similaridade e menor ângulo corresponde às praias, enquanto os mais distantes são museus e centros históricos.

```
'Monte Serrat: Sim = 0.000 | Ângulo = 90.00 graus'
'Praia enseada: Sim = 0.229 | Ângulo = 76.74 graus'
'Praia de Pitangueiras: Sim = 0.086 | Ângulo = 85.08 graus'
'Museu do Cafe: Sim = 0.000 | Ângulo = 90.00 graus'
'Aquário Municipal de Santos: Sim = 0.000 | Ângulo = 90.00 graus'
'Praia do Gonzaga: Sim = 0.129 | Ângulo = 82.57 graus'
'Basílica de Santo Antônio de Pádua: Sim = 0.000 | Ângulo = 90.00 graus'
'Museu pele: Sim = 0.027 | Ângulo = 88.46 graus'
```

Figura 8: imprime o resultado.

8 Ponto Turístico Recomendado

```
id_mais_parecido = similaridades.argmax() #argmax pega o indice que possui a maior similaridade
nome_mais_parecido = nomes_pontos[id_mais_parecido] #utiliza o id mais parecido para achar o nome
texto_mais_parecido = textos_pontos[id_mais_parecido]

print("=" * 50)

simi_max = similaridades[id_mais_parecido]
ang = np.degrees(np.arccos(simi_max))
print(f'✅{nome_mais_parecido} é o ponto turístico mais similar à sua descrição com {ang:.2f} graus ')
display(texto_mais_parecido)

print("=" * 50)
```

Figura 9: Trecho final do código que identifica o ponto turístico mais similar.

Praia da Enseada é o ponto turístico mais similar à sua descrição com 76.74 graus.

A praia mais extensa do Guarujá, com cerca de 6 km de comprimento. É famosa por suas águas calmas e rasas, o que a torna perfeita para famílias com crianças e para a prática de esportes náuticos como stand up paddle e caiaque. A orla é repleta de hotéis, restaurantes e quiosques com boa infraestrutura, sendo um dos pontos mais movimentados da cidade.

Para finalizar, a função `argmax()` localiza o ID com maior similaridade e obtém o nome e texto do ponto turístico mais semelhante — neste caso, a **Praia da Enseada**, com taxa de similaridade de 0.229 e ângulo de 76.74 graus.

9 Referências

MATEMÁTICA COM PROFA JAQUELINE SILVA. *Introdução ao LaTeX e ao Overleaf | Aula 01.* YouTube, 1 dez. 2019. Disponível em: `https://youtu.be/Y1vdXYttLSA?si=XPlbguzwkShXvtXF`. Acesso em: 25 out. 2025.

SIMPLE TECH. *Python Cosine Similarity - sklearn TfidfVectorizer and Spacy en_core_web_lg.* YouTube, 9 ago. 2022. Disponível em: `https://youtu.be/DIxxz_DvqLA?si=syqwoDYpep8Zyd-0`. Acesso em: 25 out. 2025.