FACULDADE DE TECNOLOGIA BAIXADA SANTISTA

CIÊNCIA DE DADOS

Nícolas de Almeida Lopes

RELATÓRIO – SIMILARIDADE DE EVENTOS HISTÓRICOS

ÁLGEBRA LINEAR

SANTOS – SP

2025

1 INTRODUÇÃO

Para melhorar a experiência dos seus usuários, diversas empresas utilizam sistemas de recomendação. Esses sistemas são baseados em algoritmos de aprendizado de máquina a partir de cálculos matemáticos que analisam o comportamento do usuário e para sugerir produtos, serviços ou conteúdo personalizados.

Esses sistemas ajudam a reduzir a sobrecarga de informações, tornando a escolha de produtos ou conteúdos mais rápida e intuitiva. Com o uso desse sistema, é possível aumentar a retenção de usuários.

Para medir a similaridade, um dos métodos mais utilizados é a similaridade por cosseno, uma medida que quantifica a semelhança entre dois vetores, avaliando o cosseno do ângulo entre eles. Um valor próximo de 1 indica alta similaridade, enquanto valores próximos de -1 indicam alta dissimilaridade.

Por exemplo, na Netflix, um usuário pode ser representado por um vetor cujas dimensões correspondem a diferentes filmes ou séries e cujos valores refletem suas preferências.

2 OBJETIVO

A partir desse método, o trabalho tem como objetivo desenvolver algoritmos na linguagem de programação Python para implementar um sistema de similaridade sobre eventos históricos, onde o usuário seleciona um evento de um ano específico e, com base nessa escolha, o sistema retorna diversos eventos históricos semelhantes ao que o usuário escolheu.

3 EXTRAÇÃO DOS DADOS

Para a obtenção dos dados, foi desenvolvido um algoritmo de *web scraping* para extrair dados de eventos históricos que foram obtidos a partir do site 'On This Day', no qual contém eventos do ano 1 até o ano de 2024.

3.1 WEB SCRAPING

Web scraping é uma técnica utilizada para extrair dados de sites de forma automatizada, permitindo coletar dados de páginas da web e armazená-las para análise ou utilização posterior.

Para desenvolver a técnica em Python, utilizou-se duas bibliotecas: o BeautifulSoup, uma biblioteca que permite navegar e extrair informações estruturadas a partir do HTML de uma página e a biblioteca 'subprocess', utilizada para executar comandos no *Command Prompt*.

Figura 1 - Algoritmo de *Web scraping* em Python para extração de dados

```
def extrair_eventos(ano_inicio, ano_fim):
  data = []
  for ano in range(ano_inicio, ano_fim+1):
    url = "https://www.onthisday.com/date" if ano < 1492 else "https://www.onthisday.com/events/date"
    for pagina in range(1, 6):
      command = f"{url}/{ano}?p={pagina}" if pagina >= 2 else f"{url}/{ano}"
      result = subprocess.run(["curl", command], capture_output=True, text=True, encoding='utf-8')
      soup = BeautifulSoup(result.stdout, 'html.parser')
      try:
        year = soup.find('span', class_='year').get_text()
        for event_el in (soup.find_all('li', class_=['event', 'person']) +
                         soup.find_all('div', class_='section--highlight')):
          if event_el.b:
            date = event_el.b.get_text()
          for cls in ['date', 'deathDate', 'birthDate']:
            date_el = event_el.find('a', class_=cls)
            if date_el:
              date = date_el.get_text()
          event = event_el.get_text().replace(date, '', 1).strip()
          data.append({"Year": year, "Date": date, "Event": event})
      except:
        break
  return data
```

Fonte: Elaborado pelo autor

Conforme é mostrado na Figura 1, após ser fornecido o intervalo de ano desejado para extração, o algoritmo executa *prompt* de comando 'curl' que retorna todo o HTML da URL atribuída e armazena em uma variável. Em seguida, com a utilização do BeautifulSoup, obtém-se os dados de um evento a partir de suas classes HTML.

O website conta com duas seções de eventos: uma dedicada a eventos históricos comuns e outra a eventos em destaque (*highlights*): aqueles eventos que ficaram marcados na história. O algoritmo combina as duas categorias, somando os eventos em destaque com os eventos comuns.

A Figura 2 mostra que, de acordo com o nome da classe de um elemento, é possível extrair seus dados. No elemento onde é mostrado a data, como por exemplo 'Feb 17', o nome da classe pode variar de acordo com o contexto do evento.

Figura 2 – Classes dos elementos do website 'On This Day'

Fonte: Elaborado pelo autor a partir do site 'On This Day'

4 TRATAMENTO DOS DADOS

A partir dos dados dos eventos extraídos, foi criado um *DataFrame*, uma tabela gerada pela biblioteca Pandas que contém todos os dados dos eventos históricos entre o ano 1 até 2024. No total, foram extraídos quase 80 mil eventos históricos, a conforme mostra a Figura 3.

Figura 3 – *Preview* da tabela dos dados extraídos a partir do *DataFrame*

```
df = pd.DataFrame(extrair_eventos(1, 2024))
df
```

[4]

	Year	Date	Event
0	1	Jan 1	Origin of the Christian Era
1	1	Mar 25	Origin of Dionysian Incarnation of the Word
2	1	Dec 25	The first Christmas according to calendar-make...
3	2	Aug 20	Venus and Jupiter in conjunction - possible as...
4	3	Aug 12	Venus-Jupiter in conjunction-Star of Bethlehem
...	...	...	...
77992	2024	Dec 24	NASA's Parker Solar probe makes a record-break...
77993	2024	Dec 24	Pope Francis opens the Holy Door of St Peter's...
77994	2024	Dec 25	Azerbaijani airliner crashes near Kazakhstani ...
77995	2024	Dec 29	South Korean Jeju Air plane crashes and explod...
77996	2024	Dec 31	Trinidad and Tobago declare a state of emergen...

77997 rows × 3 columns

Fonte: Elaborado pelo autor

4.1 TRATAMENTO DE TEXTO

Além das três colunas que serão exibidas futuramente para o usuário, foi criada mais uma coluna que será utilizada para o algoritmo da similaridade. A nova coluna conterá o texto tratado do evento histórico, o tratamento também será feito em Python através das bibliotecas 'string' e 'nltk', como é mostrado na Figura 4.

Figura 4 – Algoritmo para tratamento de texto.

```
def tratamento_texto(texto):
    texto = texto.lower()
    texto = texto.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(texto)
    stop_words = set(stopwords.words("english"))
    tokens = [palavra for palavra in tokens if palavra not in stop_words]
    return ' '.join(tokens)
```

Fonte: Elaborado pelo autor

Primeiramente, o algoritmo converte todas as letras para minúsculas do texto fornecido, em seguida, remove todas as pontuações. Posteriormente, as palavras são guardadas em uma lista através do comando *word_tokenize*.

Para remover as *stopwords*, isto é, remover palavras muito comuns em um idioma (como *'the'*, *'of'*, *'a'*, *'for'*, *'to'* em inglês) que não carregam muito significado por si só, o algoritmo verifica, para cada palavra *tokenizada*, se a palavra não é uma *stopword*.

Por último, o programa retorna o texto após o tratamento, separado normalmente por espaço. Esse texto tratado, será adicionada à nova coluna que foi criada, conforme mostra a Figura 5.

Figura 5 – *Preview* da tabela com a nova coluna 'texto_tratado'

```
df["texto_tratado"] = df["Event"].apply(tratamento_texto)
df
```

[8]

	Year	Date	Event	texto_tratado
0	1	Jan 1	Origin of the Christian Era	origin christian era
1	1	Mar 25	Origin of Dionysian Incarnation of the Word	origin dionysian incarnation word
2	1	Dec 25	The first Christmas according to calendar-make...	first christmas according calendarmaker easter...
3	2	Aug 20	Venus and Jupiter in conjunction - possible as...	venus jupiter conjunction possible astrologica...
4	3	Aug 12	Venus-Jupiter in conjunction-Star of Bethlehem	venusjupiter conjunctionstar bethlehem
...	...	...	...	...
77992	2024	Dec 24	NASA's Parker Solar probe makes a record-break...	nasas parker solar probe makes recordbreaking ...
77993	2024	Dec 24	Pope Francis opens the Holy Door of St Peter's...	pope francis opens holy door st peters basilic...
77994	2024	Dec 25	Azerbaijani airliner crashes near Kazakhstani ...	azerbaijani airliner crashes near kazakhstani ...
77995	2024	Dec 29	South Korean Jeju Air plane crashes and explod...	south korean jeju air plane crashes explodes t...
77996	2024	Dec 31	Trinidad and Tobago declare a state of emergen...	trinidad tobago declare state emergency gang v...

77997 rows × 4 columns

4.2 DATASET

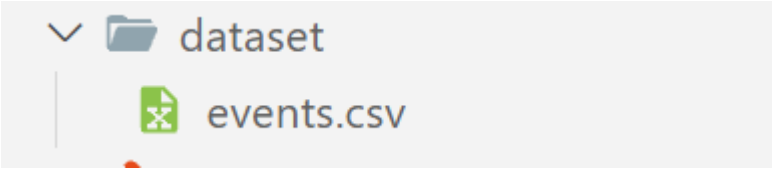
Após os dados serem carregados em um *DataFrame* e o texto dos eventos tratados adequadamente, foi possível converter para um arquivo CSV, que conterá o conjunto de dados extraídos e tratados, como é mostrado na Figura 6 e 7. O *dataset* será utilizado no sistema final para encontrar as similaridades.

Figura 6 – Conversão do *DataFrame* para um arquivo CSV.

```
df.to_csv("dataset/events.csv", sep=";", index=False)
```

Fonte: Elaborado pelo autor

Figura 7 – O arquivo CSV 'events.csv' na pasta 'dataset'.



Fonte: Elaborado pelo autor

5 O SISTEMA

Com o *dataset* dos eventos históricos adquirido, foi possível desenvolver o sistema de similaridade a partir de algoritmos e bibliotecas em Python, com uma interface gráfica. A principal biblioteca utilizada para realizar a medida da similaridade foi o Scikit-learn, uma poderosa biblioteca especializada em aprendizado de máquina. Já em questão da parte gráfica, uma versão customizada do Tkinter foi usada, o CustomTkinter, essa biblioteca trás um estilo mais moderno em comparação à biblioteca original.

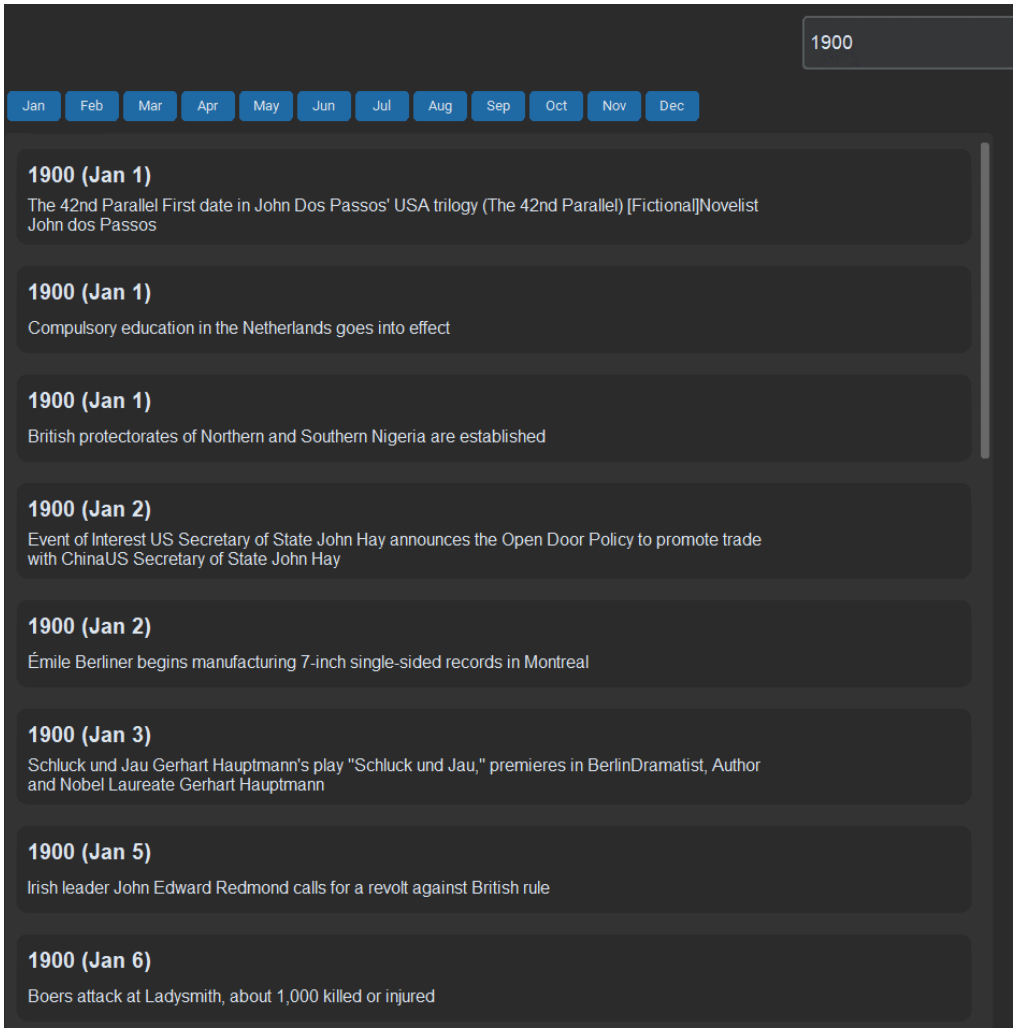
5.1 LISTAR OS EVENTOS HISTÓRICOS DE UM ANO

Antes de obter a similaridade de um evento, o usuário entrará com um ano específico para listar todos os eventos históricos que aconteceram no ano fornecido. Com o ano fornecido, é possível listar todos os eventos do ano ao clicar no botão *'List events'*, conforme é mostrado na Figura 8 e 9.

Figura 8 – Caixa de texto da interface do sistema

Fonte: Elaborado pelo autor

Figura 9 – Eventos históricos de 1900 listados



Fonte: Elaborado pelo autor

5.2 SIMILARIDADE POR COSSENO

Como já dito anteriormente, para retornar ao usuário os eventos históricos semelhantes ao que foi selecionado, o sistema irá utilizar a similaridade por cosseno, que avalia o cosseno do ângulo entre dois vetores para quantificar o grau de semelhança entre eles.

A fórmula é dada por:

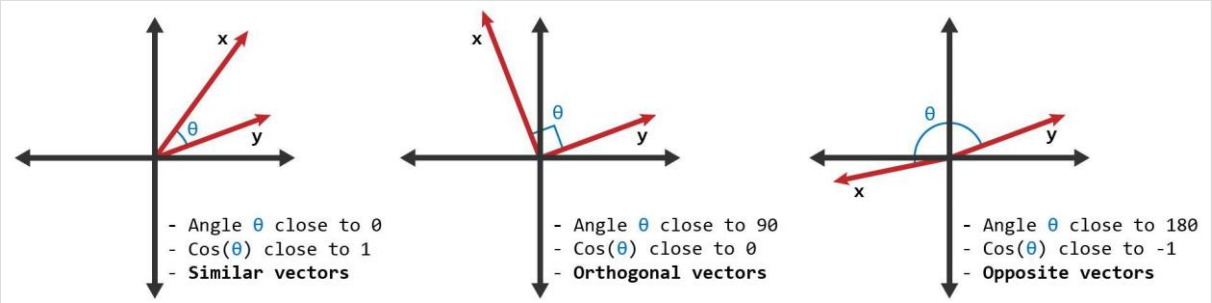
$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}||\mathbf{B}|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Onde:

A e B são vetores de características
A · B representa o produto escalar entre os vetores
|A| e |B| são as magnitudes dos vetores

Quanto mais próximo de 1 for o valor do cosseno do ângulo, os vetores ficam mais pertos um do outro, ou seja, maior a similaridade; valores próximos de -1 indicam grande dissimilaridade, isto é, os vetores estão mais distantes do outro, como é indicado a seguir na Figura 10.

Figura 10 – Similaridade por cosseno representado graficamente



Fonte: LearnDataSci

Para aplicar o método ao sistema, foi desenvolvido dois algoritmos de similaridade, um feito manualmente, sem a utilização da biblioteca Scikit-learn, o que demanda muito mais tempo e processamento. E outro foi feita com a biblioteca citada, resultando em um processo extremamente mais otimizado, de acordo com as seguintes comparações.

5.2.3 ALGORITMO MANUAL

Para desenvolver o algoritmo manualmente, foi usada apenas as bibliotecas 'numpy' para realizar os cálculos e 'nltk' para "tokenizar" as palavras, ou seja, separar cada palavras de um texto e guardar em uma lista, para posteriormente, realizar a vetorização.

Figura 11 – Algoritmo da similaridade feito manualmente

```
def obter_similaridade_manual(query, dataFrame, top):
    tokens_consulta = word_tokenize(tratamento_texto(query))
    lista_tokens = [word_tokenize(descricao) for descricao in dataFrame["texto_tratado"]]

    bag_of_words_set = set()
    for tokens in lista_tokens:
        bag_of_words_set.update(tokens)
    bag_of_words = list(bag_of_words_set)
    vetor_consulta = [tokens_consulta.count(palavra) for palavra in bag_of_words]

    lista_vetores = []
    for tokens in lista_tokens:
        vetor = [tokens.count(palavra) for palavra in bag_of_words]
        lista_vetores.append(vetor)

    def valor_cosseno(v1, v2):
        u = np.array(v1)
        v = np.array(v2)
        cos = np.dot(u,v)/((np.linalg.norm(u))*(np.linalg.norm(v)))
        return np.degrees(np.arccos(cos))

    recomendacao = dataFrame
    recomendacao["Similaridade"] = [valor_cosseno(vetor_consulta, v) for v in lista_vetores]
    recomendacao = recomendacao.sort_values("Similaridade")
    return recomendacao.head(top)
```

Fonte: Elaborado pelo autor

Conforme a Figura 11,

```
def obter_similaridade(query, dataFrame, top):
    consulta_tratada = tratamento_texto(query)
    tfidf = TfidfVectorizer()
    descricoes = dataFrame["texto_tratado"].tolist() + [consulta_tratada]
    tfidf_matriz = tfidf.fit_transform(descricoes)
    cosine_sim = cosine_similarity(tfidf_matriz[-1], tfidf_matriz[:-1])

    recomendacao = dataFrame
    recomendacao["Similaridade"] = cosine_sim[0].tolist()
    recomendacao = recomendacao.sort_values("Similaridade", ascending=False)
    return recomendacao.head(top)
```