FACULDADE DE TECNOLOGIA

BAIXADA SANTISTA

CIÊNCIA DE DADOS

RELATÓRIO – SIMILARIDADE DE EVENTOS HISTÓRICOS

Nícolas de Almeida Lopes

ÁLGEBRA LINEAR

SANTOS – SP

2025

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 ou consultas do usuário 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.

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.

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.

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 'subprocess' para executar comandos no Command Prompt. As Figuras a seguir mostra o desenvolvimento do algoritmo e como os elementos são apresentados no website.

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

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

Fonte: Elaborado pelo autor

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

	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

TRATAMENTO DE TEXTO

Para uma maior precisão na similaridade, o algoritmo irá converter todas as letras para minúsculas do texto do evento, em seguida, remove todas as pontuações, como é mostrado na Figura 4. Além disso, o programa removerá 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.

Figura 4: Algoritmo para tratamento de texto

```python
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

Por fim, o texto tratado do evento histórico é guardado em uma nova coluna do dataset, pois o sistema ainda irá utilizar a coluna original do texto do evento para apresentar ao usuário, conforme é apresentado na Figura 5.

Figura 5: Preview da tabela com a nova coluna 'texto_tratado'

```python
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

Fonte: Elaborado pelo autor

DATASET

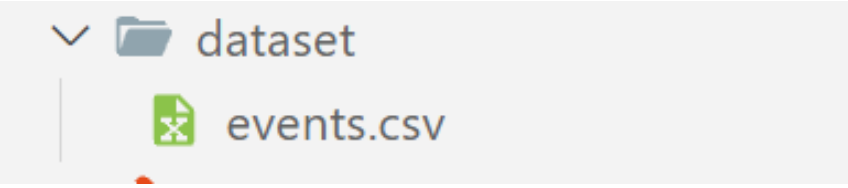
Após os dados serem carregados em um DataFrame e o texto dos eventos tratados adequadamente, foi possível converter para um arquivo CSV, que conterá o conjunto de dados extraídos e tratados, como é mostrado na Figura 6 e 7. O dataset será utilizado no sistema final para encontrar as similaridades.

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

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 traz um estilo mais moderno em comparação à biblioteca original.

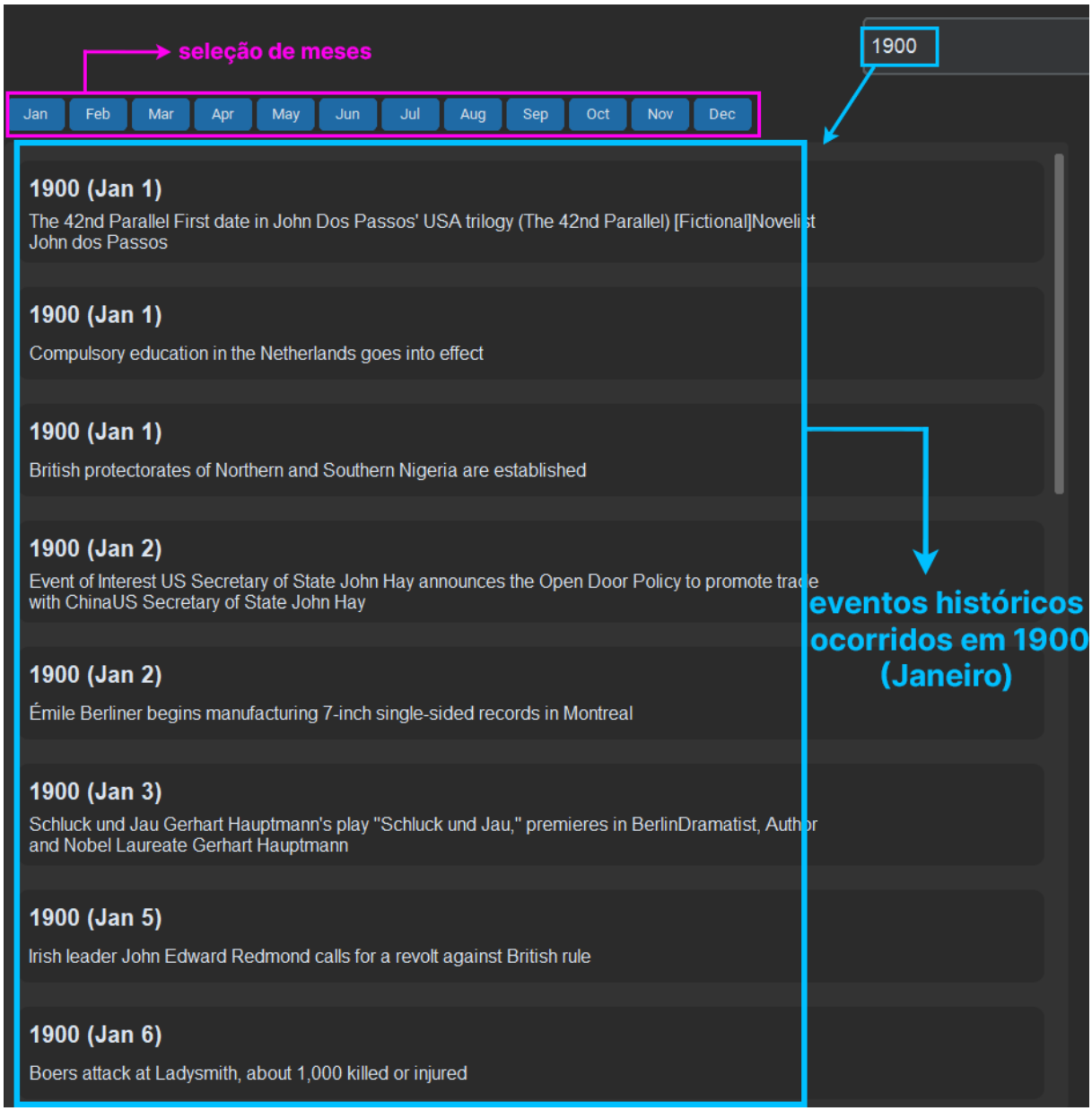
Listar os eventos históricos de um ano

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.

Figura 8: Caixa de texto da interface do sistema

Fonte: Elaborado pelo autor

Figura 9: Eventos históricos de 1900 listados



Fonte: Elaborado pelo autor

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:

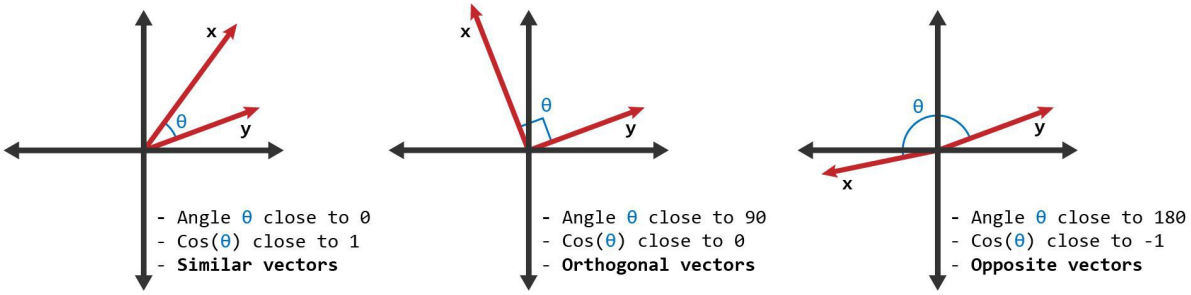
A e B são vetores de características

$A \cdot B$ representa o produto escalar entre os vetores

$\|A\|, \|B\|$ são as magnitudes

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: Elaborado pelo autor

Para aplicar o método ao sistema, foi desenvolvido dois algoritmos de similaridade, um feito com a biblioteca 'Scikit-learn', o que realiza a vetorização e o cálculo da similaridade de maneira fácil e rápida. Já o outro, foi desenvolvido manualmente, ou seja, sem a utilização da biblioteca citada, o que demanda muito mais tempo e processamento, por este motivo, não foi utilizado no sistema final.

ALGORITMO

O algoritmo da similaridade isso foi desenvolvido com o uso da biblioteca 'Scikit-learn', no qual é possível utilizar a classe 'TfidfVectorizer', que transforma todos os textos, inclusive a consulta, em representações numéricas. Esse processo é baseado no TF-IDF (Term Frequency-Inverse Document Frequency), que valoriza palavras importantes dentro de um texto e diminui o peso das mais comuns.

Com esses textos transformados em números, a função `cosine_similarity` calcula o valor do cosseno do ângulo. Por fim, os textos do DataFrame são organizados em ordem decrescente, e os mais semelhantes à consulta são retornados como similaridade. A Figura 11 mostra como o algoritmo foi desenvolvido.

Figura 11: Algoritmo da similaridade utilizado no sistema

```
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)
```

Fonte: Elaborado pelo autor

Algoritmo manual (não utilizado)

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. Além disso, foi criada uma bag of words, isto é, todas as palavras de todos os textos obtidos, sem repetições.

Por fim, é realizado o cálculo da similaridade, com o primeiro vetor sendo sempre da consulta, e o segundo sendo todos os vetores para cada evento que irá comparar com o vetor da consulta. A Figura 12 mostra todo o desenvolvimento do algoritmo feito manualmente.

Figura 12: Algoritmo da similaridade feito manualmente

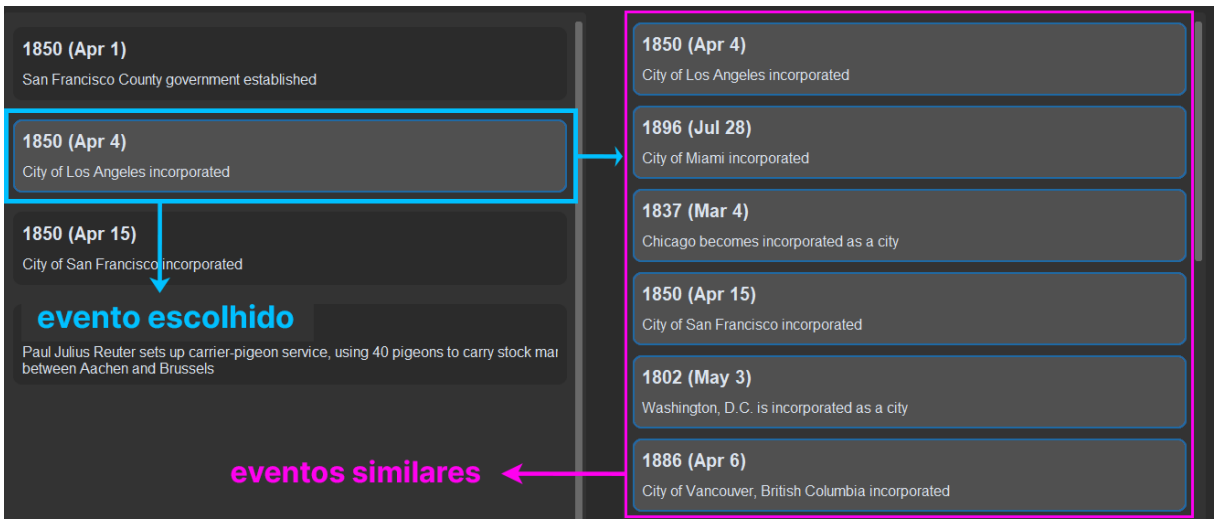
```python
def obter_similaridade_manual(query, dataFrame, top):
    tokens_consulta = word_tokenize(tratamento_texto(query))
    lista_tokens = [word_tokenize(descricao) for descricao in dataFrame["texto_tratado"]]

    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

RESULTADO

Com o algoritmo da similaridade desenvolvido, foi possível finalmente aplica-lo ao sistema. A Figura a seguir mostra que, após a seleção de um evento histórico, o sistema retorna diversos eventos similares ao que o usuário escolheu.

Figura 13: Funcionamento final do sistema



Fonte: Elaborado pelo autor