UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
PROGRAMA DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Gustavo Vicente Barroso Moser

ANÁLISE DE SIMILARIDADE ENTRE TF-IDF E MODELOS CONTEXTUALIZADOS DE LINGUAGEM BASEADOS EM TOKENS

Florianópolis, Santa Catarina – Brasil
2021

Gustavo Vicente Barroso Moser

ANÁLISE DE SIMILARIDADE ENTRE TF-IDF E MODELOS CONTEXTUALIZADOS DE LINGUAGEM BASEADOS EM TOKENS

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador(a): Prof. Dr. Carina Friedrich Dorneles

Florianópolis, Santa Catarina – Brasil
2021

Gustavo Vicente Barroso Moser

ANÁLISE DE SIMILARIDADE ENTRE TF-IDF E MODELOS CONTEXTUALIZADOS DE LINGUAGEM BASEADOS EM TOKENS

Este(a) Trabalho de Conclusão de Curso foi julgado adequado(a) para obtenção do Título de Bacharel em Ciências da Computação, e foi aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação do INE – Departamento de Informática e Estatística, CTC – Centro Tecnológico da Universidade Federal de Santa Catarina.

Florianópolis, Santa Catarina – Brasil, 13 de Dezembro de 2022.

Prof. Dr. Jean Everson Martina
Coordenador(a) do Programa de
Graduação em Ciências da Computação

Banca Examinadora:

Prof. Dr. Carina Friedrich Dorneles
Orientador(a)
Universidade Federal de Santa
Catarina – UFSC

Prof. Dr. Renato Fileto
Universidade Federal de Santa Catarina –
UFSC

Prof. Dr. Ronaldo dos Santos Mello
Universidade Federal de Santa Catarina –
UFSC

Aos meus pais, os dois maiores incentivadores dos meus sonhos,
e a Ally (in memoriam), que me ensinou muito sobre o amor
sem dizer uma única palavra. Este trabalho é dedicado a eles.

AGRADECIMENTOS

Agradeço aos meus pais, Ieda e Isair, pelo cuidado, dedicação e amor incondicional enquanto guias da minha formação como ser humano. Obrigado por me apoiarem em todos os meus sonhos, e pelo colo e carinho quando algum destes não se realiza. Pelos conselhos e orientações quando tudo parecia estar perdido, nunca me senti desamparado. Vocês me ensinaram sobre a importância da educação e do conhecimento como meio de transformação social.

À minha avó Zaida e à minha madrinha Marilene, pessoas com papel fundamental para que eu me tornasse quem sou hoje. Sem o apoio incansável da minha família nada disso seria possível, essa conquista é nossa. Amo vocês, muito obrigado por tudo.

À minha querida Ally (in memoriam), meu grude, minha companheira nos últimos quatorze anos. Eu contava os dias ansiosamente para comemorar esse momento com você, espero que esteja me vendo aí de cima e que tenha orgulho de mim. Ao Snoopy e a Luna, meus amores, minha dupla dinâmica, mesmo de longe sempre presentes, fazem meus dias mais felizes e cheios de amor.

Ao escotismo e a todos os ensinamentos que por meio dele me foi proporcionado nesses 18 anos como membro integrante. Aos desafios que superei, às lições que aprendi, sou eternamente grato.

À Monique, meu xuxu, por todo o amor, apoio e companheirismo desde meu primeiro dia de graduação. Obrigado por compartilhar sua vida comigo, dividir as alegrias, as angústias, você traz leveza aos meus dia e faz tudo ficar melhor.

Aos meus amigos, pessoas que integraram à minha vida ao longo dessa jornada, do escoteiro, do ensino médio, da faculdade, da firma, que acompanharam cada passo e cada vitória. Vocês são peça importante desta conquista.

À minha orientadora, Carina Dorneles, por ter aceitado o desafio de me orientar e partilhar comigo conhecimento, descobertas, sucessos e fracassos. Pelos conselhos e orientações recebidos. Valorizo muito isso e levarei comigo para sempre.

À Universidade Federal de Santa Catarina e aos professores, que têm papel vital na formação de profissionais competentes e comprometidos com o futuro, contribuíram muito para o profissional que me tornei. Agradeço imensamente por todos os aprendizados.

A Deus, por todas as oportunidades.

“Deixe o mundo um pouco melhor do que encontrou.”
Robert Stephenson Smyth Baden-Powell

RESUMO

Com o crescimento do acesso aos meios de comunicação e a popularização das redes sociais, o termo *fake news* ganha cada vez mais forma e espaço. O termo é utilizado para denominar informações falsas, mentiras que circulam pelos meios de informação. Diferentemente de sátiras, *fake news* são publicadas com a intenção de enganar os leitores a fim de obter ganhos, sejam eles políticos ou financeiros, geralmente acompanhados de títulos sensacionalistas para chamar a atenção. O Twitter é uma rede social voltada para comunicação em tempo real utilizada por milhões de usuários, é nela que muitas *fake news* são divulgadas e conseguem tomar grandes proporções. A possibilidade de compartilhar informações com apenas um clique faz com que sua disseminação se dê muito rapidamente. Um exemplo atual diz respeito às *fake news* que têm circulado sobre temas voltados à saúde. Com o surto da COVID-19 pelo planeta, muitos viram oportunidades de gerar informações falsas a respeito do uso de máscaras, vacinas, entre outros. Portanto, nesse contexto, o objetivo do presente trabalho é desenvolver uma solução para localizar *tweets* com informações cuja veracidade possa ser checada com experimentos publicados em artigos científicos, utilizando casamento de dados e similaridade semântica, posteriormente, classificando-os *tweets* como fraudulentos ou não.

Palavras-chaves: Notícias falsas. Twitter. saúde. coronavírus. casamento de dados. TFIDF. Word embeddings. SBERT.

ABSTRACT

As the access to the media grows, and the popularity of social networks increases, the term "fake news" gets more and more form and space. This term is used to denote false information, lies that surround through the media. Unlike satires, the fake news are published intending to deceive readers in order to get gains that can be financial or political, and usually are followed by sensationalist titles to get attention. Twitter is a social network focused on real-time communication, used by millions of users. It is where a lot of news spread and can take large proportions. The possibility of sharing information with just one click makes the dissemination of it happen very quickly. A current example concerns the false news that have been circulating on health-related topics. While the outbreak of COVID-19 generates a crisis across the planet, many opportunities to create false information regarding the use of masks, vaccines, and others. Therefore, in this context, the objective of the present work is to develop a solution to locate tweets whose veracity can be checked with experiments published in scientific articles, using data matching and semantic similarity, and later classifying them as fraudulent or not.

Keywords: Fake news. Twitter. health. coronavirus. data matching. TFIDF. Word embeddings. SBERT.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BERT	*Bidirectional Encoder Representations from Transformer*
DOM	*Document Object Model*
ETL	*Extract, Transform and Load*
HTML	*HyperText Markup Language*
IDF	*Inverse Document Frequency*
JSON	*JavaScript Object Notation*
NLTK	*Natural Language Toolkit*
PLN	Processamento de Linguagem Natural
SBERT	*Sentence-BERT*
SGBD	Sistema Gerenciador de Banco de Dados
STS	*Semantic Textual Similarity*
TF	*Term Frequency*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium*
XML	*Extensible Markup Language*
XPath	*XML Path Language*

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais inserido no cotidiano das pessoas, o termo *fake news* têm exercido o papel de vilão quando se trata do combate à desinformação. Com seu uso popularizado a partir de 2016, durante as eleições americanas (ALLCOTT; GENTZKOW, 2017), o termo, que simboliza uma falácia, uma mentira, é inimigo dos meios de comunicação, sendo influência direta no crescimento de crises de informação.

Oferecendo facilidade para o compartilhamento de informações, as redes sociais têm sido vetores essenciais para a disseminação de notícias falsas. Com o crescimento do acesso à internet, cresce também o acesso a redes sociais. Um exemplo bastante popular de redes sociais é o Twitter, fundado em 2006. A plataforma oferece aos seus milhões de usuários um espaço para compartilhamento de conteúdo de maneira prática e simples, além de uma funcionalidade onde é possível ver quais são os assuntos mais falados no mundo, os *Trending Topics*. Dados publicados por (SMITH, 2020) mostram que aproximadamente 500 milhões de *tweets* são postados por dia.

A grave situação sanitária gerada pela COVID-19 se complicou ainda mais devido à rápida disseminação de informações. Por um lado, possibilita ao público o acesso a mídias confiáveis como fonte de informação, sites e artigos de entidades e órgãos de saúde nacionais e internacionais. No entanto, abre espaço para uma intensa propagação de *fake news* (BARCELOS *et al.*, 2021). Nesta situação, a disseminação de notícias falsas afeta diretamente a eficácia de programas e iniciativas como, por exemplo, campanhas pró-vacinas e pró-máscaras.

Além disso, (MONTEIRO, R. A. *et al.*, 2018) relata que há um desempenho ruim por parte dos humanos em separar informações verdadeiras de *fake news*, um fator que pode ser influente na alta disseminação. A partir dos problemas citados acima, a tarefa de classificar *tweets* como informações falsas e identificá-las se torna ainda mais difícil.

Com base nesta problemática, a proposta deste trabalho é o desenvolvimento de uma ferramenta, o *Tweet-Nature Similarity Analyzer* (TNSA), responsável por extrair dados de artigos científicos comprovados e de *tweets*, e após a extração, utilizando técnicas de *data matching* e similaridade semântica a partir dos *tweets* e artigos, realiza o casamento dos dados com técnicas que não usem de *machine learning* para a análise dos dados, como TF-IDF e modelos contextualizados de linguagem baseados em tokens. Dessa forma, com o resultado da análise é possível avaliar se o que está sendo afirmado na conclusão do artigo é o que vem sendo propagado, ou seja, se são ou não *fake news*.

1.1 OBJETIVOS

O presente trabalho resulta na ferramenta *Tweet-Nature Similarity Analyzer*, que realiza uma *match* entre *tweets* e artigos científicos buscando, desta forma, avaliar se os *tweets* estão propagando informações de acordo com o que é apresentado no artigo. Para isso, dados do Twitter e de artigos de revistas científicas são extraídos e tratados através de estruturas de coleta, para então serem analisados e o *match* seja feito.

Os objetivos específicos são os seguintes:

- Desenvolver um Scraper para coleta e extração de *tweets* utilizando as palavras-chave "covid mask";
- Desenvolver um Scraper para coleta e extração de dados de artigos científicos utilizando as palavras-chave "covid mask";
- Armazenar dados de *tweets* e artigos em um banco de dados;
- Implementar o *match* dos dados entre os dados de artigos e *tweets* utilizando TF-IDF / modelos contextualizados de linguagem baseados em tokens;
- Apresentar a avaliação do resultado da análise de dados de forma visual (por meio de gráficos);

1.2 METODOLOGIA

A metodologia adotada para a obtenção dos resultados é o ciclo PDCA (*Plan-Do-Check-Adjust*) (DEMING, 1986). Trata-se de um processo iterativo, composto por quatro passos, utilizado para o controle e melhoria contínua de projetos. Este método é usado para o desenvolvimento das atividades do presente projeto, e o passo *check* é a análise ou verificação dos resultados alcançados e dados coletados, onde são utilizados os indicadores de acompanhamento, descritos abaixo. Este passo pode ocorrer simultaneamente à realização do projeto, momento em que se verifica se o trabalho está sendo feito da forma devida, ou após a execução quando são feitas análises estatísticas dos dados e verificação dos itens de controle. Nesta fase podem ser detectados erros ou falhas.

1. **Levantamento bibliográfico** (*plan*): o levantamento bibliográfico é realizado constantemente e deve tratar da busca retrospectiva sobre o assunto investigado, como forma de verificar os trabalhos desenvolvidos recentemente, ou em desenvolvimento, por outros grupos de pesquisa. Esta etapa proporciona o amadurecimento dos temas pesquisados e seu posterior estudo analítico. Além disso, esse estudo possibilitará o estabelecimento de parâmetros de comparação

entre a pesquisa deste projeto com os demais trabalhos, e o planejamento na construção dos algoritmos, que são extremamente importantes na confecção de artigos técnicos que apresentam resultados deste projeto.

2. **Propostas de soluções** (*do*): com base no levantamento bibliográfico realizado, levando-se em conta as deficiências e oportunidades de pesquisa encontradas, são especificadas propostas de soluções para os problemas levantados.

3. **Implementação das soluções** (*check*): todas as atividades desenvolvidas são implementadas dentro do ambiente proposto através de módulos de software, ou protótipos, que possibilitem a análise das soluções através da validação de resultados.

4. **Validação e ajuste das propostas** (*adjust*): a cada finalização da implementação de um protótipo (gerado a partir dos algoritmos e propostas desenvolvidas), são realizados testes para validação das propostas e, se necessário, realização de ajustes. O objetivo principal é confirmar, através de testes e métricas objetivas, se os requisitos específicos para um determinado objetivo foram cumpridos. Caso sejam necessários ajustes, um novo planejamento deverá ser realizado, com consulta à trabalhos da literatura (voltando ao passo *plan* do método PDCA).

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentados alguns conceitos e ferramentas utilizadas neste projeto. Primeiramente, são introduzidos conceitos importantes como a definição de Notícia, a problemática das *fake news* e o ponto de diferenciação entre ambos. Também são apresentadas as definições de Redes Sociais e Twitter, assuntos abordados neste trabalho. Após essa seção, é introduzido o conceito de *Web Scraping* e são apresentadas duas ferramentas onde esta técnica é aplicada, o Scrapy e o Tweepy, ambos utilizados no desenvolvimento deste trabalho. Também são introduzidos os conceitos do processo de *data matching* e de similaridade semântica, bem como os métodos de TF-IDF e *word embeddings*, também explorados para o desenvolvimento. Por último, a ferramenta Sentence-BERT é introduzida.

2.1 NOTÍCIA

Cada vez mais nos encontramos cercados de notícias. Presente em todos os meios de comunicação, através de texto, áudio ou vídeo, elas já estão incluídos no cotidiano das pessoas. Desde um relato de um evento recente, significativo ou dramático sobre algum tema, há uma grande quantidade de formas para definir o que é uma notícia. De acordo com (TANDOC JR.; LIM; LING, 2018), as notícias costumam ser vistas como uma produção do jornalismo, onde espera-se que informações independentes, precisas, abrangentes e confiáveis sejam fornecidas.

Dessa forma, espera-se que a noticia seja a representação dos acontecimentos. É muito semelhante a uma reportagem, mas o que torna sutil a diferença entre esses dois formatos de informação é o fator tempo. Enquanto a reportagem trata sobre fatos não necessariamente novos, sendo mais importante reforçar as reações que reatualizem os fatos, a notícia é o atual, o agora, o novo. De acordo com (LAGE, 2021): “A reportagem é planejada e obedece a uma linha editorial, um enfoque. A notícia, não”.

Também salientado por (TANDOC JR.; LIM; LING, 2018), espera-se que o jornalismo relate, antes de qualquer coisa, a verdade. Porém, há um julgamento a partir do material bruto de uma noticia sobre qual informação será incluída ou excluída do resultado final. Logo, existe uma lacuna, uma vulnerabilidade sobre a produção de notícias que pode ser utilizada para fins de transmitir preferências próprias ou como área útil para influência externa.

2.1.1 *Fake News*

Se as notícias possuem essas lacunas, então utilizar delas em prol de uma ideologia ou para obter algum ganho é uma possibilidade. O termo *Fake news* consiste

no ato de fraudar informações, e ganhou proporções nas eleições americanas de 2016. No entanto, a ideia de *fake news* já é bem mais antiga. Um exemplo é *"The great moon haux"* (em português, "O grande engodo da lua") de 1835, onde o jornal *New York Suns* publicou uma série de artigos sobre a descoberta de vida na lua (ALLCOTT; GENTZKOW, 2017).

Durante as eleições americanas, (ALLCOTT; GENTZKOW, 2017) reuniram alguns fatos sobre os casos de *fake news* e sua possível influência no resultado final das eleições, como, por exemplo, o fato que a maior parte das informações fraudulentas que circulavam tendiam a favor de Donald Trump sobre Hillary Clinton. Há a sugestão de que, se não fosse pela influência das *fake news*, o presidente não teria sido eleito na época.

As *fake news* também estiveram presentes durante o surto de COVID-19 no Brasil, desde estatísticas até tratamento e prevenção ao novo coronavírus. No estudo realizado por (BARCELOS *et al.*, 2021), das *fake news* registradas até 30 de junho de 2020, foram identificadas 329 notícias fraudulentas relacionadas à pandemia, disseminadas principalmente através de redes sociais como *WhatsApp* e *Facebook*.

2.2 TWITTER

O Twitter é uma das 10 redes sociais mais utilizadas no Brasil e entre as 20 mais utilizadas no mundo, de acordo com (VOLPATO, 2021). Fundado em 2006, a rede conta com quase 400 milhões de usuários ao redor do mundo.

Organizada na forma de microblog, a rede permite que seus usuários publiquem textos de até 280 caracteres. Essas mensagens são denominadas *tweets*, e possibilitam, além de texto, o uso de mídias, como vídeos, imagens e links nas publicações, a inclusão de *hashtags*, caracteres precedidos do símbolo (#), muito utilizada na filtragem ou na promoção de conteúdos. Permite também que o usuário curta e reposte *tweets* de outros usuários, o chamado *retweet*.

Há também a página do usuário, organizada no formato de *timeline*, que exibe as postagens feitas em ordem cronológica. Pode ser personalizada com imagem de fundo e foto de perfil e pode exibir algumas informações básicas sobre o usuário. O usuário pode ter seu perfil público ou privado, o último restringe o acesso aos *tweets* apenas a seguidores daquele usuário.

2.3 *WEB SCRAPING*

Definido por (MITCHELL, 2015), *web scraping* é a prática de coletar dados de forma automatizada, ou seja, através da escrita de programas que, por sua vez, requisitam dados ao servidor da web e analisam os dados recebidos para extrair as informações necessárias. Os *web scrapers* são excelentes na coleta e no processa-

mento de grandes volumes de dados pois, diferentemente da visão que comumente se possui sobre sites, através de páginas, visualizando e acessando uma por vez, os *scrapers* possibilitam a visualização de milhares de páginas por vez.

Além disso, *scrapers* fornecem um acesso a informação que os dispositivos de busca tradicionais não possuem. Enquanto os sites de pesquisas conseguem informar apenas anúncios ou sites populares de pesquisas de voos, que são as informações que o buscador conhece, como exemplifica (MITCHELL, 2015), um *web scraper* bem desenvolvido consegue mapear o custo de voos ao longo do tempo e em uma grande variedade de sites, podendo informar a melhor hora de comprar uma passagem.

O uso de *web scrapers* também pode ser uma opção quando existem limitações com APIs. Geralmente, prefere-se o uso de uma API antes de se pensar em construir um *scraper* para a extração dos mesmos dados mas, muitas vezes, a documentação ou o formato de dados fornecidos como resposta pode não suprir a necessidade desejada, e então, opta-se pela construção do *scraper*.

Ferrara *et al.* (2014) afirma que a natureza semi-estruturada das paginas web é uma das funcionalidades mais exploradas na extração de dados da internet. As páginas podem ser representadas como árvores com raízes ordenadas e rotuladas, onde estes rótulos representam os níveis de elementos HTML que constituem uma página web, e a organização da árvore representa a hierarquia dos elementos e os diferentes níveis de aninhamento dos mesmos. A representação da página web utilizando a estrutura de árvore é geralmente referida como DOM, conceito regulamentado pela W3C.

A árvore DOM é muito bem explorada para fins de extração de dados da web, se aproveitando do uso de ferramentas como, por exemplo, o XPath. Definido também pela W3C e tipicamente utilizada em arquivos do dialeto XML, esta ferramenta fornece uma sintaxe poderosa para endereçar elementos de um documento XML e, também, das páginas HTML, podendo encontrar de maneira os elementos existentes na estrutura da página (FERRARA *et al.*, 2014). A adoção da ferramenta como aparato de endereçamento de elementos na página web é muito explorada na literatura.

2.3.1 Scrapy

Um popular framework desenvolvido em Python, o Scrapy é uma ferramenta de alto nível para *web scraping* utilizadas para extrair dados estruturados das páginas. Com uma vasta gama de finalidades, pode ser utilizado desde mineração de dados até monitoramento de testes automatizados (SCRAPY, 2021).

2.3.2 Tweepy

Oferecido como um pacote *open source* do Python, o Tweepy é uma interface para acessar a API oficial do Twitter com Python. A ferramenta inclui uma série de classes e métodos que representam os modelos do Twitter e os *endpoints* da API. Além

disso, trata de forma transparente de vários detalhes de implementação. Abstraindo o tratamento de detalhes de baixo nível, o Tweepy permite que se possa focar apenas na funcionalidade que se quer construir (GARCIA, s.d.).

2.4 *DATA MATCHING*

Desde os anos 80, a comunidade de bancos de dados vem estudando sobre a integração de dados e informações. O principal problema é a correspondência entre esquemas, o casamento de dados, onde dois ou mais esquemas de dados são utilizados para produzir um mapeamento entre atributos que correspondam semanticamente um ao outro (LIU, 2009).

Contudo, os critérios e métodos utilizados quase todos são baseados em heurísticas de domínio, onde a formulação matemática é complexa. Liu (2009) Afirma ainda que, para construir um sistema de correspondência entre esquemas, é necessário que as heurísticas de mapeamento sejam produzidas a partir do entendimento de "boa correspondência".

Data matching tem como propósito identificar e comparar conjuntos dados, buscando por uma correspondência entre eles. Contudo, encontrar essa correspondência é um grande desafio. Conceitos idênticos podem possuir diferenças estruturais ou léxicas, conceitos semelhantes podem não ser exatamente os mesmos e até palavras chave podem ser similares, mas com significados opostos.

2.5 *SIMILARIDADE SEMÂNTICA TEXTUAL*

Textos em geral são estruturas complexas quanto à sua construção e sua representação. Estimar a similaridade semântica textual (do inglês *Semantic Textual Similarity*, abreviado para STS) é um dos problemas de pesquisa desafiadores e abertos na área de PLN (sigla para Processamento de Linguagem Natural), pois a versatilidade da linguagem natural dificulta o trabalho de definir métodos baseados para determinar medidas de similaridade semântica (CHANDRASEKARAN; MAGO, 2021).

A similaridade semântica textual é definida como a medida da equivalência semântica entre dois blocos de texto. Os métodos de similaridade semântica pontuam a similaridade dos textos buscando decidir se são ou não similares. A similaridade semântica é frequentemente utilizada como sinônimo de relacionamento semântico. Porém, o relacionamento semântico analisa também as propriedades semânticas entre duas palavras. Por exemplo, "café" e "chá" são semanticamente semelhantes, diferentemente de "café" e "caneca", que apesar de estar muito relacionadas, não são consideradas semelhantes (CHANDRASEKARAN; MAGO, 2021).

2.5.1 TF-IDF

O TF-IDF, sigla para "*Term Frequency-Inverse Document Frequency*", é uma técnica para quantificar palavras em um conjunto de documentos (MANNING; RAGHAVAN; SCHÜTZE, 2008). É similar ao conceito de *Bag-of-Words*, mas nesse processo ao invés da contagem da quantidade de palavras, as palavras recebem um peso que significa a sua importância no conteúdo de um documento e em todo o conjunto de documentos, baseada no cálculo de frequência apresentado abaixo.

$$TF\_IDF = TF * IDF$$

Term Frequency (TF) é a frequência de uma palavra "*p*" em um documento "*d*", calculado pela equação abaixo. Para isso, é necessário computar a contagem de todas as palavras do vocabulário no escopo do documento. Como o fato de uma palavra aparecer 100 vezes em um documento não a torna 100 vezes mais importante, é comum que se reduza a frequência bruta utilizando o log_{10} da frequência. Além disso, soma-se 1 à contagem pois não se pode calcular o log de 0 (JURAFSKY; MARTIN, 2021).

$$TF(p, d) = log_{10}(count(p, d) + 1)$$

Já *Inverse Document Frequency*)(IDF) é a inversa da frequência de um documento, com o objetivo de balancear o TF, fazendo com que palavras muito frequentes em todos os documentos se tornem menos importantes, como por exemplo as *stop words*, ou palavras vazias, como conjunções e artigos. O cálculo do IDF é dado pela equação abaixo. Assim, para cada documento o resultado é um vetor de pesos, uma para cada palavra do documento. *D* representa o conjunto de documentos, e "*df*" representa a função que conta a frequência de um termo "*t*" em todos os documentos.

$$IDF(t) = log_{10}(\frac{count(D)}{df(t)})$$

Para melhorar a eficiência do método, é possível realizar um pré-processamento, onde o conteúdo da página é tratado, removendo *stop words*, que são palavras sem significado semântico para o cálculo tais como artigos e conjunções, e aplicando a técnica de *stemming*, que opera sobre as palavras do texto retornando sua raiz, com o objetivo de reduzir as diversas formas de derivação de uma palavra para sua base comum. O exemplo de Manning, Raghavan e Schütze (2008) apresentado abaixo demonstra a extração das raízes das palavras e a aplicação da técnica em uma sentença.

$$\text{am, are, is} \Rightarrow \text{be}$$
$$\text{car, cars, car's, cars'} \Rightarrow \text{car}$$
$$\text{the boy's car are different colors} \Rightarrow \text{the boy car be differ color}$$

2.5.2 *Word embeddings*

Word embeddings são vetores de tamanho fixo que representam palavras (ALMEIDA; XEXÉO, 2019). Traduzido para o português como incorporação de palavras, é o resultado de métodos utilizados para representar palavras de um vocabulário por meio de vetores de números reais (GRAZZIOTIM, 2022).

O objetivo é que, a partir desses números calculados sobre cada uma das palavras, palavras com semelhança semântica estejam próximas no espaço vetorial. A representação vetorial do documento como um todo é um vetor resultante da redução dos vetores de valor das palavras que fazem parte do contexto do documento. Diferentemente do TF-IDF apresentado em 2.5.1, que representam as palavras como um longo e esparso vetor com as dimensões correspondentes às palavras em um vocabulário ou a documentos em uma coleção, *embeddings* são vetores curtos e densos, com um número de dimensões bem menor que o produzido por modelos de vetor esparsos (conceito sem interpretação muito clara), que funcionam melhor em tarefas de PLN. Densos pois os valores vão ser números reais que podem ser negativos (JURAFSKY; MARTIN, 2021). Grazziotim (2022) diz que essa construção de *embeddings* se baseia na hipótese distribucional, dessa forma, a partir de palavras vizinhas no espaço vetorial é possível obter o significado de uma palavra.

Frequentemente empregados em aplicações com interesse no sentido das palavras, modelos de *word embeddings* podem ser divididos em dois tipos. *Static Word Embeddings* produzem um único vetor para uma determinada palavra, independendo do contexto a qual ela esteja inserida. O contexto é considerado apenas para o treinamento de um modelo, ou seja, no momento de converter as palavras de um documento a vetores as palavras vizinhas não são consideradas, impossibilitando a obtenção de diferentes vetores para palavras homônimas. *Contextual Word Embeddings* produzem diferentes vetores para uma mesma palavra pois consideram o contexto no qual a mesma está inserida. Dessa forma, palavras homônimas recebem pontuações diferentes a partir das palavras que as acompanham em uma sentença. Ambas atingem performance no estado da arte em tarefas de PLN, contudo, com maior gasto computacional pois necessitam observar todas as palavras da sentença (GRAZZIOTIM, 2022).

2.6 SENTENCE-BERT

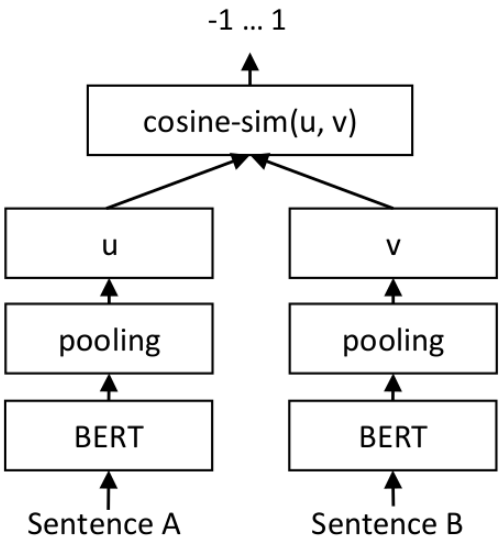
De acordo com o estudo feito por Grazziotim (2022), o BERT (*Bidirectional Encoder Representations from Transformers*) é o primeiro modelo de linguagem não supervisionado, profundamente bidirecional e pré-treinado utilizando conjuntos de textos não anotados, capaz de gerar *word embeddings* contextuais. Desenvolvido pela Google, está incorporado ao Google Search, serviço de busca da empresa.

O BERT define um novo estado da arte em performance para diversas classi-

ficações de sentenças, incluindo resposta a questões, classificação de sentenças e regressão de pares de sentenças. Contudo, a configuração do mesmo é inadequada para várias tarefas de regressão devido a uma grande quantidade de combinações possíveis. Por exemplo, buscando em uma coleção de 10.000 sentenças o par com maior grau de similaridade possível, o BERT leva cerca de 65 horas de computação até encontrar o resultado, pois cerca de 50 milhões de combinações de sentenças precisam ser computadas (REIMERS; GUREVYCH, 2019). Uma das desvantagens da estrutura de rede do BERT é que *word embeddings* de sentenças não são computados independentemente, o que dificulta o processo de derivação de *word embeddings* no BERT. O desempenho para sete tarefas STS ficou abaixo do desempenho médio do GloVe, outro algoritmo para representação vetorial de palavras.

Utilizando de uma arquitetura de redes siamesas, o Sentence-BERT (ou SBERT) é criado como uma modificação do BERT, permitindo que vetores de tamanho fixo para sentenças de entrada possam ser derivados. Nele, sentenças semanticamente similares podem ser encontradas utilizando medidas de similaridade como a função cosseno (Cosine-Similarity) ou Manhatten/Euclidian Distance (REIMERS; GUREVYCH, 2019). Essas medidas têm uma performance extremamente eficiente em hardwares modernos, dessa forma, permite com que o SBERT seja utilizado para busca de similaridade tão bem quanto para clusterização. A Figura 1 apresenta a estrutura resultante das modificações feitas no BERT.

Figura 1 – Redes siamesas e Pooling do SBERT



O SBERT acrescentou uma operação de *pooling* na saída do BERT com o objetivo de derivar *word embeddings* de tamanho fixo. Para fazer ajustes na rede neural do BERT, processo conhecido como *fine-tunning*, Reimers e Gurevych (2019) cria redes siamesas/trigêmeas para atualizar os pesos de forma que as *word embeddings* produ-

zidas sejam semanticamente significativas e possam ser comparadas por cosseno.

O estudo avalia a qualidade em vários benchmarks comuns onde poderia se alcançar uma melhoria significativa em relação aos métodos de *word embeddings* modernos. O SBERT é computacionalmente mais eficiente e pode ser usado para tarefas que não são viáveis para serem modeladas com o BERT, como o exemplo citado acima, onde o agrupamento de sentenças requer cerca de 65 horas. Com o SBERT, a computação passa a ser de cerca de 5 segundos, e pode calcular a similaridade por cosseno em aproximadamente um centésimo de segundo (REIMERS; GUREVYCH, 2019).

3 TRABALHOS RELACIONADOS

Como resultado do levantamento bibliográfico, esta seção apresenta alguns trabalhos relacionados a *fake news* e sobre a criação de ferramentas para análise e detecção das mesmas no cenário da língua portuguesa. Todos os estudos relatam a falta de bases de dados para o português brasileiro.

3.1 *CONTRIBUTIONS TO THE STUDY OF FAKE NEWS IN PORTUGUESE: NEW CORPUS AND AUTOMATIC DETECTION RESULTS*

O estudo de Rafael A. Monteiro *et al.* (2018) investiga o problema da detecção de *fake news* na língua portuguesa e, inspirado por iniciativas em outras linguagens, introduz uma coleção de notícias de referência para o português, composta por notícias verdadeiras e falsas, as quais foram analisadas com o objetivo de descobrir características linguísticas de cada uma. Após a avaliação em cima da coleção de dados, utilizando técnicas de aprendizado de máquina e aplicando ideias de estudos anteriores para outras linguagens, foram realizados testes sobre a detecção automática de *fake news*.

Buscando criar uma coleção de confiança, amostras reais escritas em português foram coletadas e rotuladas. A coleção, chamada de “Fake.br Corpus” é composta de notícias reais e fraudulentas com foco apenas no português brasileiro. Rafael A. Monteiro *et al.* (2018) afirma ainda que não há um conjunto similar para o português. Ao todo, foram coletadas 7200 notícias, com exatas 3600 verdadeiras e 3600 notícias falsas.

Num primeiro momento, foram coletadas e analisadas manualmente todas as *fake news* disponíveis durante o período de coleta. Das notícias obtidas, foram filtradas apenas as notícias completamente falsas. A partir disso, utilizando de um *web crawler*, foram coletadas noticias das maiores agências de notícias do Brasil (G1, Folha de São Paulo e Estadão). O *crawler* procurou nas páginas da web dessas agências por palavras-chave das notícias falsas, que ocorriam nos títulos dessas notícias ou que fossem mais frequentes nos textos.

A partir da coletânea de notícias, partiu-se para a criação de um classificador automático de *fake news*. Foram executados alguns testes utilizando aprendizado de maquina sobre a coletânea, utilizando algumas técnicas de diferentes paradigmas e truncando ou não o texto. Rafael A. Monteiro *et al.* (2018) descreve os resultados como acima das expectativas e cita que um fator que pode explicar os resultados ótimos é a filtragem das notícias com “meias verdades”, que elas tornariam o processo mais complexo.

3.2 DETECÇÃO DE NOTÍCIAS FALSAS UTILIZANDO TÉCNICAS DE *DEEP LEARNING*

Com o objetivo de gerar um classificador de notícias em português brasileiro que seja capaz de identificar *fake news* utilizando técnicas de *deep learning*, o trabalho de Guarise e Rezende (2019) surge da falta de existência de conteúdo acadêmico sobre detecção de notícias falsas por software para a língua portuguesa, principalmente pela falta de uma base de dados com notícias previamente classificadas.

O trabalho utiliza a coleção “Fake.br”, desenvolvida por Rafael A. Monteiro *et al.* (2018), como base de dados para a implementação, contudo, houve a necessidade de realizar um pré-processamento das notícias da base de dados, que são constituídos apenas dos textos puros das notícias, contendo um arquivo “.txt” para cada uma das notícias. Dessa forma, foi necessário remover alguns caracteres não necessários para a compreensão do texto, como ™, por exemplo. Também foram feitas a separação das aspas simples e duplas de palavras ligadas a elas (a mudança de “atraso” para “ atraso ”) e a correção de algumas sentenças (o termo “num” passa a ser “em um”) (GUARISE; REZENDE, 2019).

O avanço de algoritmos para a criação de palavras vetorizadas e de técnicas de processamento da linguagem natural agregam em um melhor desempenho de redes neurais sobre tarefas relacionadas a texto e linguagem. A estrutura hierárquica de textos permitiu a escolha de uma entre as diversas arquiteturas de redes neurais existentes, a HAN, ou *Hierarchical Attention Networks*, que utiliza esta característica do texto para analisar os dados de entrada (GUARISE; REZENDE, 2019).

Diferentemente de outros modelos de redes neurais, o modelo HAN permite que os resultados sejam visualizados de forma que as palavras e sentenças mais determinantes fiquem destacadas, possibilitando a classificação através de um mapa de calor, destacando as sentenças mais importantes na classificação (GUARISE; REZENDE, 2019). É a partir do HAN que o classificador foi desenvolvido.

O trabalho atingiu, após implementação e treinamento do modelo, resultados similares a outros trabalhos considerados como o estado da arte da classificação de *fake news* para a língua inglesa e obteve um bom desempenho dentro da base de dados escolhida para o experimento do projeto (GUARISE; REZENDE, 2019).

3.3 *CLASSIFICAÇÃO DE FAKE NEWS COM TEXTOS DE NOTÍCIAS EM LÍNGUA PORTUGUESA INTEGRANDO DATA WAREHOUSING E MACHINE LEARNING*

O trabalho desenvolvido por (MONTEIRO, R. *et al.*, 2019) também trata sobre a temática da classificação de *fake news*. A proposta foi utilizar técnicas de aprendizado de máquina para classificar textos de notícias falsas para uma posterior aplicação ao processo de ETL de um *Data Warehouse*, além da geração de um ambiente de

consulta que, disponibilizado, contribuísse para trabalhos futuros. Assim, um *dataset* foi criado e foram avaliados os métodos de classificação Regressão Logística, AdaBoost, Naive Bayes e SVM. O melhor método foi utilizado em um sistema de avaliação online.

Utilizando Python e a biblioteca Beautiful Soup no desenvolvimento de um *web crawler* para a coleta de notícias verdadeiras e fraudulentas para a construção de um *dataset*. Este *dataset* é constituído de 1744 títulos e corpo de notícias falsas coletadas dos sites *boatos.org* e *g1.globo.com/fato-ou-fake*. Além disso, foram também coletados 3185 títulos de notícias verdadeiras coletadas do site *brasil.elpais.com* (MONTEIRO, R. *et al.*, 2019).

Os algoritmos avaliados eram acoplados à ETL e, a partir da fonte de dados, as notícias eram classificadas automaticamente, incrementando a precisão do classificador. Uma interface web também foi construída, desta forma, os usuários poderiam submeter links de notícias para validar se a mesma é verdadeira ou não.

Os teste inicialmente ocorreram apenas com os títulos das notícias, depois com o corpo e, por fim, com o corpo junto ao título. Os resultados obtidos por (MONTEIRO, R. *et al.*, 2019) mostraram que o método de Naive Bayes obteve o melhor desempenho, pelo fato de apresentar a maior precisão entre os algoritmos avaliados, além de ser um método de aprendizado incremental, ou seja, o algoritmo é treinado enquanto opera.

3.4 ANÁLISE COMPARATIVA

A partir dos trabalhos apresentados acima, é possível construir uma tabela com técnicas e as fontes de dados utilizadas. Observa-se que todos os trabalhos utilizaram técnicas já existentes de aprendizado de máquina para classificar as notícias extraídas. Tanto (MONTEIRO, R. A. *et al.*, 2018) quanto (GUARISE; REZENDE, 2019) utilizam a mesma base de dados, o “Fake.br” (a Tabela 1 apresenta apenas as fontes utilizadas na criação do “Fake.br”). Da mesma forma, todos os trabalhos utilizam apenas sites de notícias como fontes de dados para seus trabalhos.

O TNSA estuda uma abordagem diferente das demais com o uso de técnicas de avaliação semântica dos dados como forma de realizar o *Data Matching*. Além disso, ao invés de um site de notícias como fonte dos dados a serem classificados, o TNSA utiliza dados de uma rede social e de artigos científicos, com o objetivo de avaliar se o que está sendo publicado pelos usuários acorda com as afirmações publicadas nos artigos.

Tabela 1 – Comparação de técnicas e fonte de dados dos trabalhos

Trabalho	**Técnica utilizada**	**Fonte de dados para avaliação**	**Fonte de dados confiáveis**
(MONTEIRO, R. A. *et al.*, 2018)	Machine Learning + SVM	A Folha do Brasil, The Jornal Brasil, Diário do Brasil, Top Five TV	G1, Folha de São Paulo, Estadão
(GUARISE; REZENDE, 2019)	Deep Learning + HAN	A Folha do Brasil, The Jornal Brasil, Diário do Brasil, Top Five TV	G1, Folha de São Paulo, Estadão
(MONTEIRO, R. *et al.*, 2019)	Machine Learning + Regressão Logística + AdaBoost + Naive Bayes + SVM	boatos.org, Fato ou Fake (G1)	El País Brasil
Tweet-Nature Similarity Analyzer	**TF-IDF / modelos contextualizados de linguagem baseados em tokens**	**Twitter**	**Revista Nature**

4 *TWEET-NATURE SIMILARITY ANALYZER*

Nesta seção é introduzida a ideia do projeto a ser desenvolvido, o TNSA. Primeiramente, aborda-se o conceito do projeto, apresentando uma visão geral dos módulos e a arquitetura elaborada para o mesmo. Após isso, a implementação é apresentada, tratando de informações técnicas como linguagens, bibliotecas e estratégias que são utilizadas no desenvolvimento.

4.1 VISÃO GERAL

A proposta de desenvolvimento é que, através do *match* dos dados analisados, seja possível avaliar se os *tweets* estão propagando informações de acordo com o que os artigos científicos publicados por organizações reconhecidas afirmam. A Figura 2 apresenta a arquitetura da implementação definida para este trabalho.

O primeiro módulo do projeto é responsável por popular uma base de dados com dados reais obtidos de duas fontes, a rede social e os artigos científicos. As postagens da rede social são obtidas através de um *Scraper*, ferramenta responsável por extrair os dados necessários para o trabalho. Após a extração, as postagem passarão por um pré-processamento, que transforma a estrutura da postagem, deixando-a mais adequada para o modelo definido para um banco de dados relacional. O fluxo é o mesmo para a obtenção dos artigos científicos.

Com os dados já disponíveis no banco de dados, é possível buscá-los e encaminhá-los para o próximo módulo, responsável pela análise de similaridade e por disponibilizar os resultados obtidos. Nessa segunda etapa, o dado ainda passa por mais um pré-processamento antes de ser direcionado para a análise. Ao final do processo, a similaridade da postagem é classificada ao ser comparada com as frases presentes nos artigos científicos.

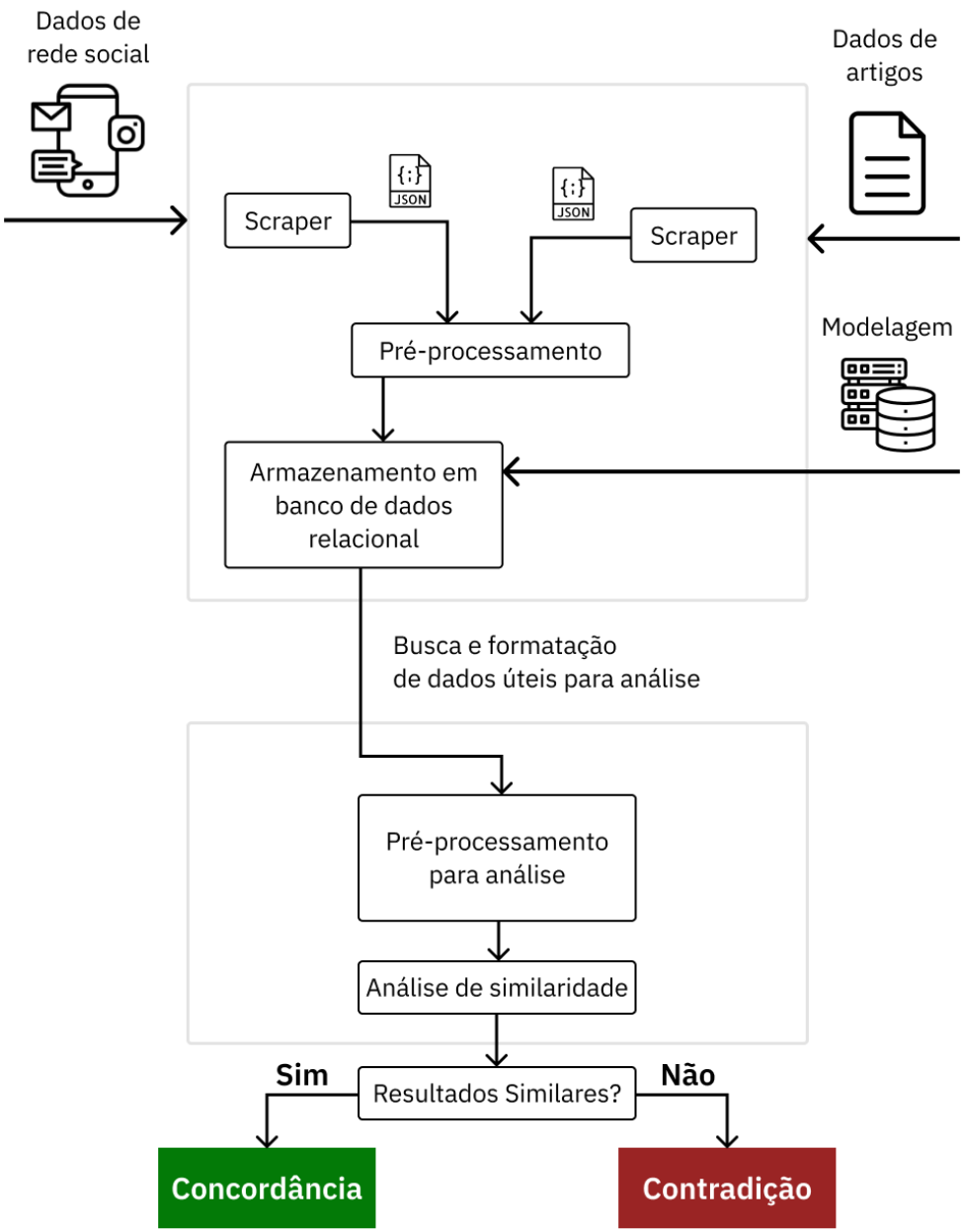
4.2 IMPLEMENTAÇÃO

O desenvolvimento do TNSA utiliza a linguagem de programação Python, juntamente com a ferramenta Poetry, responsável pelo gerenciamento de dependências, como bibliotecas e scripts, no trabalho.

As bibliotecas Tweepy e Scrapy são utilizadas no o desenvolvimento dos *scrapers*, responsáveis por auxiliar na extração dos *tweets* e das seções dos artigos científicos da revista Nature, respectivamente. Além disso, como limitante para o escopo do trabalho, todos os dados coletados são do idioma inglês.

O PostgreSQL é utilizado como SGBD para banco de dados relacional, sendo o mesmo disponibilizado através de um *container* Docker, que é uma plataforma aberta para desenvolvimento, envio e execução de aplicações. Como ele fornece a capaci-

Figura 2 – Modelo arquitetural do *Tweet-Nature Similarity Analyzer*

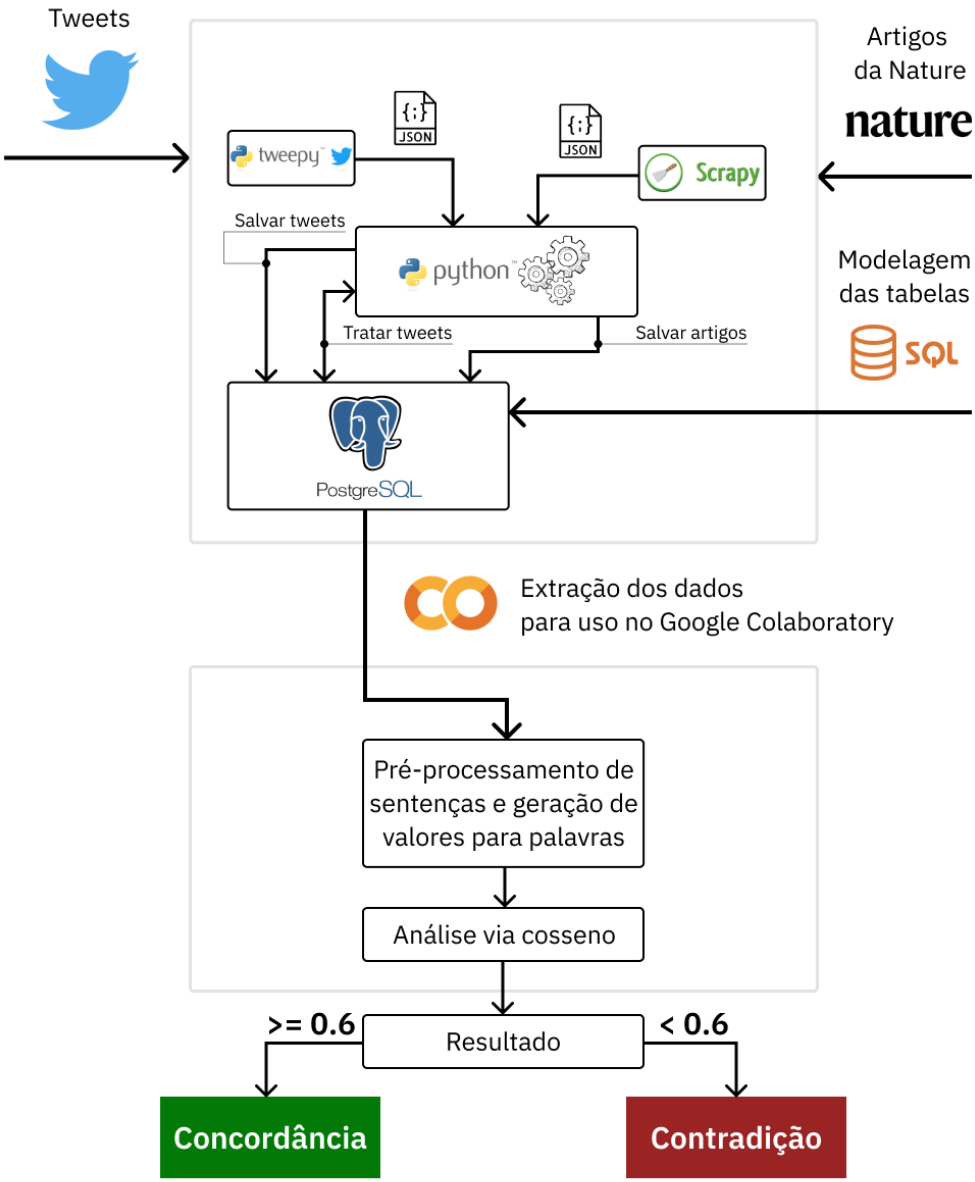


dade de executar uma aplicação em um ambiente semi-isolado chamado *container*, é possível desacoplar aplicações da infraestrutura local.

Além do PostgresSQL, a biblioteca *psycopg2*, responsável por interfacear as transações com o banco, também é utilizada no trabalho. A biblioteca NLTK (Natural Language Toolkit) é utilizada em processamento de sentenças, responsável por separação de sentenças, tokenização e remoção de *stop words*. O Sentence Transformers é a biblioteca responsável por gerar vetores que simbolizam as palavras de uma sentença, bem como oferecer funções de análise semântica.

Informações sensíveis como tokens e informações sobre a conexão com o banco de dados são armazenado em um arquivo denominado "*.env*", responsável pelas variáveis de ambiente. A Figura 3 apresenta o modelo de implementação tecnológica da ferramenta, desenvolvida a partir do modelo arquitetural.

Figura 3 – Modelo de implementação do *Tweet-Nature Similarity Analyzer*



4.2.1 Extração, tratamento e armazenamento de dados

A seção a seguir aborda detalhes da implementação do primeiro módulo do trabalho, responsável pela extração dos dados com o uso dos *Scrapers*, bem como

seu tratamento e armazenamento no banco de dados PostgreSQL, até a geração de arquivos intermediários necessários para o segundo módulo.

4.2.1.1 Extração de *tweets*

Antes de iniciar o desenvolvimento do extrator responsável por coletar os *tweets*, é necessário realizar um cadastro na plataforma de desenvolvedores do Twitter, pois é a partir desta conta que se obtém o token de acesso (*Bearer*), necessário para fazer chamadas aos *endpoints* do Twitter.

O Tweepy é uma interface aos *endpoints* disponibilizados na API do Twitter, contendo métodos que requisitam os dados desses *endpoints*. Após a geração do token de acesso, o Tweepy pode ser inicializado e utilizado, sendo definido para esse trabalho na classe *TwitterScraper*. A classe *Client* do Tweepy é a responsável por gerenciar o acesso aos métodos relacionados com *endpoints* por meio do token de acesso. O trecho de Código 1 ilustra a configuração e um exemplo de consulta aos 100 *tweets* mais recentes da rede social.

O exemplo é de chamada a um dos métodos de busca disponibilizados pela biblioteca, o *search_recent_tweets*. É passado como parâmetro uma *query* que representa um filtro realizado para esses *tweets*, e o *max_results* que é a quantidade máxima de resultados que se deseja obter da consulta. A *query* é constituída de palavras-chave, frases ou filtros específicos da biblioteca, como *"-filter:retweets"*, que exclui *retweets* dos resultados. Os parâmetros citados acima estão disponíveis para este método específico, mas cada método possui parâmetros relacionados ao que o *endpoint* da API do Twitter está apto a receber. Os métodos estão disponíveis em Tweepy (2022).

Código 1 – Exemplo de consulta utilizando o Tweepy

```
1 import tweepy
2 client = tweepy.Client(bearer_token="BEARER_TOKEN_GERADO")
3 result = client.search_recent_tweets(query=query,max_results=100)
```

O resultado da busca é uma estrutura de dicionário, trazendo uma lista de objetos no qual cada objeto representa a estrutura de um *tweet*, contendo o texto da publicação, o autor, quantas curtidas ou *retweets* tiveram, data de publicação, localização geográfica, e outros atributos. Para este trabalho apenas o texto e seu identificador são utilizados. O identificador é necessário para armazenamento, sendo utilizado como chave primária da tabela *tweet*, garantindo unicidade dos dados coletados. A modelagem do banco de dados pode ser vista na Seção 4.2.1.4. Após a busca, os dados encontrados são persistidos no banco de dados.

No presente trabalho, em um período de 1 mês, são coletados os *tweets*. Alguns *endpoints* da API do Twitter não são disponibilizados gratuitamente, apenas na API Premium, assim, os métodos do Tweepy também necessitam do acesso à API Premium.

Dos métodos de busca por vários *tweets*, apenas a busca pelos mais recentes pode ser feita sem custos (citada no exemplo acima). Dessa forma, utilizando deste método de busca com a *query* “covid masks -is:retweet”, o processo é executado repetidamente até obter-se uma quantidade de dados julgada suficiente, totalizando 2130 *tweets*.

Definido no arquivo de configuração do Poetry para o projeto, há um *script* para executar o processo de busca por *tweets* e seu armazenamento no banco de dados, simplificando a chamada da função responsável. Por linha de comando, no diretório raiz do projeto, é executado da seguinte forma:

Código 2 – Executando o Scraper dos Tweets

```
1 $ poetry run save_tweets
```

4.2.1.2 Processamento dos *tweets*

Com as publicações do Twitter armazenadas no banco de dados, é possível analisar a estrutura dos textos. A partir dessa análise, pode-se observar alguns elementos presentes nas publicações coletadas que podem ser tratados, de forma a simplificar o texto e auxiliar no cálculo de um resultado mais preciso.

Da avaliação de textos como “🚨 Open gym cancelled 🚨 Tuesday & Wednesday this week. Here’s remainder of open gym schedule. Bring multiple Covid-19 masks so you can play with dry masks- pulling them down below your👃& 🗣️ will not be tolerated. @mwwarriors @MWActivities @MaineWestWAPA #COVID https://t.co/aYn0NWmt5y”, três elementos a serem tratados são encontrados:

- Emojis;
- Menções;
- *Hyperlinks*;

Como tratativas para emojis, a biblioteca *emoji_translate* é utilizada para converter cada emoji em seu significado. O objetivo é que esses elementos contribuam para a etapa de análise, trazendo o significado do que eles representam para o texto como um fator relevante.

Para menções e *hyperlinks*, a estratégia escolhida é apenas removê-los, já que não são úteis na comparação com os artigos científicos e não trazem valor à análise. Ambos são encontrados por meio de expressões regulares e removidos do texto da publicação.

O resultado do tratamento dos *tweets* é persistido em uma nova tabela com estrutura idêntica a tabela de origem dos dados, a *tweet_clean*, apresentada na Seção 4.2.1.4. Ela utiliza também o identificador como chave primária, mas também o utiliza

como chave estrangeira para a tabela *tweet*. Publicações coletadas que, após o tratamento, ficam com o texto vazio, ou seja, sem nenhuma informação, são removidas desta tabela no mesmo processo para evitar valores vazios populados na tabela.

Há também um *script* definido para executar o tratamento dos *tweets* coletados. A execução por linha de comando, no diretório raiz do projeto, ocorre da forma abaixo.

Código 3 – Executando o tratador de Tweets

```
1 $ poetry run clean_tweets
```

4.2.1.3 Extração de artigos científicos

O processo de extração dos artigos inicia-se com a seleção dos artigos que fazem parte da amostra para o desenvolvimento do presente trabalho.

Os cinco artigos escolhidos são da revista científica Nature, sendo buscados pelas palavras-chave “masks” e “covid”. O motivo da escolha pela Nature se dá pelo fato de que todos os artigos científicos da revista seguem um Guia de Formatação (NATURE, 2022) para serem publicados, e por seguirem este guia, auxiliam no desenvolvimento de um *Scraper* mais preciso e eficiente, orientado pela estrutura do artigo.

Para utilizar o Scrapy e extrair as informações necessárias dos artigos, é necessário criar uma *Spider*. *Spiders* são classes que o Scrapy utiliza para coletar as informações de um ou mais *websites*. A *Spider* implementada deve herdar a classe *Spider* do Scrapy e deve ser desenvolvida definindo as solicitações iniciais a serem feitas e como analisar o conteúdo da página baixada para extrair dados. Opcionalmente, pode ser definido a forma como a *Spider* deve seguir os links nas páginas (SCRAPY, 2021).

A *Spider* do trabalho é definida na classe *ArticleScraper*. As URLs dos artigos são armazenados em um arquivo externo, que é lido pela *Spider* quando a mesma é executada. A lista das URLs lidas é atribuída à variável *start_urls*, necessária para o funcionamento da *Spider*. A partir dessa lista, as requisições para obter o conteúdo da página iniciam e o método *parse* é ser invocado. O *parse* é o método responsável por tratar a resposta baixada para a requisições feita a uma URL, e a resposta é o conteúdo da página, ou seja, o código HTML. Também é necessário para o funcionamento da *Spider*.

Com a página baixada pela *Spider*, a buscar dados nas estruturas da página a partir de elementos HTML e suas propriedades, como a *class* é realizada utilizando o XPath. Da página dos artigos são extraídos o título, os autores, as sessões presentes e o conteúdo das sessões. Além disso, as sessões são filtradas para que apenas texto seja encontrado, desconsiderando imagens e outros elementos. Cada extração

de artigo resulta em um Dicionário, que é construído com o objetivo de ser convertido e exportado como um arquivo JSON.

Após a exportação o arquivo JSON é utilizado, sendo lido e tendo suas informações persistidas no banco de dados. Para extrair os artigos por linha de comando, basta executar o comando abaixo.

Código 4 – Executando o Scraper de Artigos

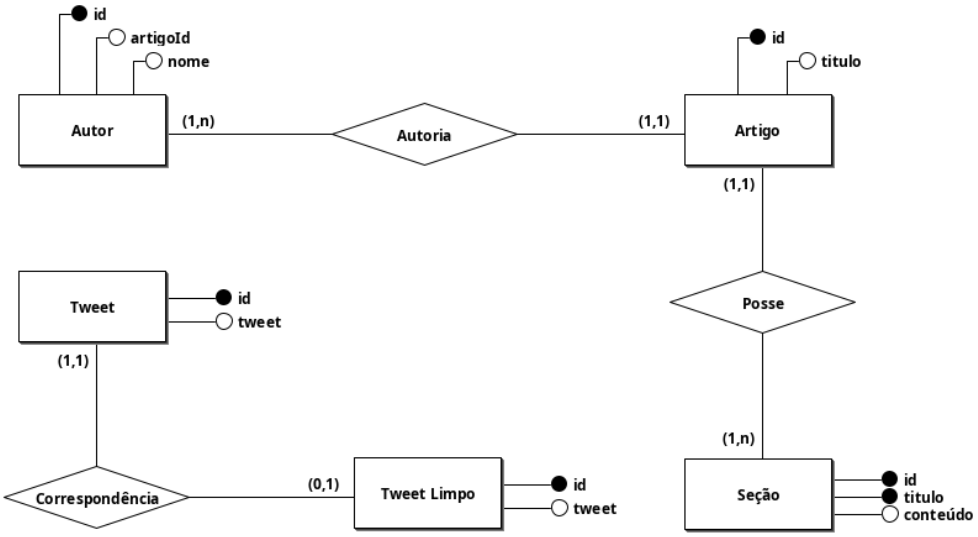
```
1 $ poetry run save_articles
```

4.2.1.4 Modelagem e armazenamento dos dados

Durante o desenvolvimento dos extratores de dados, surge a necessidade de armazenar os dados coletados. Avaliando as propriedades que seriam armazenadas do artigo e do *tweet*, percebe-se que os dados obtidos são estruturados e com algumas relações, podendo ser armazenado de forma organizada em uma tabela. Portanto, um banco de dados do tipo relacional é escolhido para ser utilizado. Como SGBD, o PostgreSQL, um dos mais avançados, desenvolvido como projeto de código aberto.

Assim, o diagrama ER (Entidade Relacionamento) 4 é desenvolvido com o propósito de facilitar a visualização das entidades e atributos que estão presentes no trabalho antes do desenvolvimento da comunicação dos serviços de extração dos dados. A partir dele, o esquema do banco de dados é gerado.

Figura 4 – Modelo ER do banco de dados do *Tweet-Nature Similarity Analyzer*



Com o esquema já definido no PostgreSQL, a estrutura utilizada para persistir e consultar os dados obtidos dos processos presentes nas Seções 4.2.1.1 e 4.2.1.3 pode ser desenvolvida.

A classe *PostgresManager* tem o propósito de gerenciar inicio, fim de conexão e transações, utilizando o *psycopg2*. Ela possui alguns métodos como *insert*, *update*

e *delete*, que recebem o SQL a ser executado como uma *string* e o executam, tendo comportamentos e tratativas específicas para cada caso.

4.2.1.5 Geração de arquivos intermediários

Durante o desenvolvimento do trabalho há a necessidade de discutir a transferência de um dos módulos para outro ambiente de desenvolvimento com maior capacidade de processamento, dessa forma, uma análise mais rápida pode ser realizada utilizando uma infraestrutura mais robusta.

O Google Colab, ou Google Collaboratory, é um serviço de armazenamento em nuvem de *notebooks*, hospedado pelo Jupyter Notebook, voltados à criação e execução de códigos em Python, diretamente em um navegador. Com ele é possível ler, desenvolver e rodar códigos e *rich texts* em documentos interativos que agrupam células de códigos, os notebooks, e mantê-los online (ROVEDA, 2020).

O poder computacional utilizado para a execução do software é fornecido pela nuvem de computadores da Google, assim, é possível processar uma grande quantidade de dados. Além disso, é possível acessar pastas e arquivos do Google Drive para serem utilizados dentro da plataforma.

Portanto, para migrar os dados coletados para arquivos que podem ser importados no Google Colab, há uma funcionalidade desenvolvida neste trabalho que exporta versões distintas do conteúdo dos dados armazenados, destinados ao *data matching*, como possíveis formatos necessários para sua execução. Os artigos e *tweets* são exportados de duas formas:

1. **Conteúdo completo** (*tratado*): Os dados são exportados por completo para um JSON. Os *tweets* presentes na tabela *tweet_clean* são exportados com identificador e texto para um arquivo só com *tweets*. Já os artigos têm o conteúdo de todas as suas seções concatenado, sendo gerado um arquivo com identificador e o conteúdo completo dos artigos.

2. **Raízes de palavras**: Utilizando a técnica de *stemming* junto da biblioteca NLTK (Natural Language Toolkit), os artigos e *tweets* são manipulados para que o retorno seja uma lista de raízes de palavras gerada a partir do conteúdo completo, processo denominado *stemming* (MANNING; RAGHAVAN; SCHÜTZE, 2008). O algoritmo de *stemming* escolhido foi o *Porter Stemmer*, que é um processo para remover as terminações morfológicas mais comuns da língua inglesa. Além disso, *stop words* existentes no texto são removidas do resultado. Também exporta os dados para um JSON.

Ou seja, como resultado do processo de geração dos arquivos com os dados coletados, são gerados quatro arquivos, dois para *tweets* e dois para artigos. Para

gerar os arquivos do projeto, basta utilizar o script, definido no Poetry e apresentado abaixo.

Código 5 – Executando o gerador de JSON

```
1 $ poetry run export_jsons
```

Após a geração dos arquivos, eles estão disponíveis para importação na plataforma do Google Colab. Para o presente trabalho, os arquivos são importados no Google Drive para serem utilizados na plataforma.

4.2.2 Implementação da análise dos dados extraídos

O *data matching* é o principal módulo da aplicação. É através dela que os dados são comparados, isto é, mensurados por um algoritmo que retorna a avaliação de concordância semântica entre duas sentenças. Ou seja, é o responsável por avaliar se os *tweets* propagados seguem ou não os estudos publicados.

Existem algumas técnicas para realizar o casamento entre dados. No presente trabalho duas técnicas são abordadas, a primeira é utilizando o TF-IDF, que é uma medida estatística, para encontrar a relevância em um documento, já a outra é utilizando o *framework* Sentence Transformers junto da função Cosseno para análise de similaridade. Ambas as técnicas e aplicações são melhores descritas abaixo.

4.2.2.1 *Data matching* com TF-IDF

Como método escolhido na primeira abordagem, o TF-IDF é utilizado buscando gerar o peso, também chamado de grau de importância, das palavras do artigo e dos *tweets* (MANNING; RAGHAVAN; SCHÜTZE, 2008), que serão calculados com o objetivo de gerar o *matching* entre os dados. O TF-IDF é calculado para os artigos considerando cada artigo é um documento. O mesmo ocorre para os *tweets*, mas neste caso, cada publicação é considerada como um documento.

A biblioteca *sklearn*, utilizada neste trabalho, fornece a classe *TfidfVectorizer*, que contém a função que realiza o cálculo do TF-IDF sobre uma lista de documentos, retornando uma matriz com os pesos de cada uma das palavras para cada um dos documentos.

Inicialmente, após a extração dos dados, percebe-se que a forma que um *tweet* e um artigo são escritos é sintaticamente muito distinta. Enquanto o artigo traz uma formalidade na formação das suas sentenças, o *tweet* é mais informal, contendo gírias, abreviações, entre outros elementos. Assim, é necessária uma forma de aproximar os *tweets* do formato de escrita do artigo. Neste caso, os arquivos gerados na Seção 4.2.1.5 que retornam as raízes das palavras são uma possível solução, pois apenas as raízes estão presentes e com poucas variações de escrita. Assim, é sobre elas que

Tabela 2 – Exemplo de raízes de palavras de um *tweet* e seus respectivos pesos

Raiz da palavra	**TF-IDF**
dish	0.426358440518873
dispos	0.024829323185419597
dissemin	0.0074778979399909625
distanc	0.012414661592709798
distribut	0.10552462353803328
emit	0.04486738763994577
extrem	0.014955795879981925
hole	0.14829858800656454
knowledg	0.018537323500820567
mask	0.030915968901549056

o TF-IDF é calculado.

Contudo, observando a Tabela 2 com os valores extraídos de um dos vetores de TF-IDF para um *tweet* da base, é possível perceber que a raiz da palavra *mask*, considerada uma palavra importante para os dados deste trabalho, tem um peso atribuído bem mais baixo que raízes de palavras que não estão diretamente relacionadas ao contexto da "efetividade de máscaras contra a doenças respiratórias (como a COVID-19)", como *dish* e *hole*, indicando que essa palavra é menos importante.

Portanto, percebe-se um problema ao calcular a estatística TF-IDF sobre as amostras deste trabalho, já que o esperado é que *mask* seria uma palavra mais importante no contexto dos documentos. Como todos os artigos e *tweets* presentes na amostra abordam o mesmo tópico, palavras que deveriam ser importantes são classificadas como não tão relevantes para o algoritmo, pois aparecem em todos os documentos. Assim, com a limitação do tema dos documentos da amostra impactando no TF-IDF, a métrica não se adaptou ao cenário de estudo.

4.2.2.2 Métrica de similaridade por cosseno

Grazziotim (2022) cita em seu trabalho que uma forma intuitiva de se definir uma similaridade entre dois vetores *u* e *v* é medindo a distância entre eles.

Uma das formas mais comuns de se obter medidas de similaridade é através da similaridade por cosseno (do inglês Cosine-Similarity). Calculada em 1, apenas a orientação dos vetores é considerada, descartando assim a sua magnitude. Como os valores estão limitados ao intervalo do cosseno, o $[-1, 1]$, a interpretação do resultado da operação é direta. Similaridades com valor tendendo a 1 representam vetores

apontando para a mesma direção, ou seja, sentenças similares, já similaridades com valor tendendo a -1 representam valores apontando para direções opostas, ou seja, sentenças diferentes. Esse cálculo é baseado na operação de produto escalar, da álgebra linear, que age como métrica de similaridade. Logo, o valor tende a ser alto apenas quando vetores tem grandes valores nas mesmas direções, e baixo quando vetores possuem zeros em diferentes direções, ficando com produto escalar 0, indicando dissimilaridade. Contudo, sendo os valores de frequências não negativos, o valor de cosseno para esses vetores varia apenas de 0 a 1 (JURAFSKY; MARTIN, 2021).

$$sim\_cos(u, v) = \frac{u * v}{||u|| * ||v||} \tag{1}$$

Independente da redução de dimensionalidade dos vetores das sentenças, a similaridade por cosseno se mantém dentro do mesmo intervalo fechado, devolvendo um resultado mais consistente e intuitivo (GRAZZIOTIM, 2022) e, portanto, é a forma de medir a similaridade entre sentenças utilizada no presente trabalho.

4.2.2.3 *Data matching* com Sentence Transformers

Os arquivos citados na Seção 4.2.1.5 são importados no ambiente do Google Colab para pré-processamento e análise. Cada arquivo, contendo todo o conteúdo de um artigo concatenado em uma única *string*, é lido e tratado, sendo primeiramente separado em sentenças utilizando o NLTK. Há a aplicação de uma expressão regular com o objetivo de remover algumas estruturas de cada sentença, como pontuação, alguns caracteres especiais e caracteres Unicode. Dois dicionários são resultantes deste processo, um diretamente após o processo acima. O segundo dicionário é gerado a partir do primeiro, no entanto, cada sentença passa por um processo de *stemming* (semelhante ao realizado na Seção 4.2.1.5), formando as novas sentenças apenas com as raízes das palavras apenas. O resultado do pré-processamento é apresentado na Tabela 3.

O *framework* Sentence Transformers fornece um método rápido para computar representações vetoriais para sentenças, parágrafos e imagens, provendo uma série de modelos pré-treinados voltados para a computação de sentenças e *word embeddings* para mais de 100 linguagens, aperfeiçoados para diversos casos de uso (UKPLAB, 2022). Tais *embeddings* podem ser comparados com similaridade por cosseno para encontrar significados similares entre sentenças. A ferramente ainda permite o treinamento e *fine-tunning* de redes próprias. O trabalho de (REIMERS; GUREVYCH, 2019) é parte integrante desta ferramenta. Os modelos *open source* pré-treinados estão disponíveis na plataforma Hugging Face, repositório existente para criação, treino e disponibilização de modelos (HUGGING FACE, 2022).

O modelo escolhido para incorporar as sentenças presentes na base de dados

Tabela 3 – Pré-processamento de sentenças

Sentença original	Method 2 corresponds to direct colony counting using an HD counter and converting the count into viable particles (7659\u2009±\u20091177 counts)
Sentença pré-processada	Method 2 corresponds to direct colony counting using an HD counter and converting the count into viable particles
Raízes da sentença	correspond direct coloni count use counter convert count viabl particl

deste trabalho é o *all-mpnet-base-v2*, modelo com a arquitetura padrão do BERT-base (de 768 dimensões), que provém a melhor qualidade dentre os disponibilizados, com o custo de não ser o mais rápido. Os dados de treino deste modelo resultam da concatenação de diversos *datasets* para aprimorar o modelo. O total de pares de sentenças é superior a 1 bilhão de sentenças (SENTENCE TRANSFORMERS, 2022).

Código 6 – Configurando o *all-mpnet-base-v2*

```
1 from sentence_transformers import SentenceTransformer
2 model = SentenceTransformer('sentence-transformers/all-mpnet-base-v2')
```

Após configurar o Sentence Transformers com o modelo escolhido, apresentado no Código 6, as *word embeddings* podem ser calculadas para todas as sentenças. Aproveitando do poder da biblioteca, todas as sentenças são concatenadas em uma única lista para que este processamento seja realizado em lote e apenas uma vez para todos os dados. Os *embeddings* gerados são convertidos para um objeto do tipo Tensor, através do parâmetro “convert_to_tensor”. Este objeto é esperado na função de cálculo de similaridade por cosseno disponibilizada pelo próprio Sentence Transformers (REIMERS; GUREVYCH, 2019). A função cosseno é executada utilizando os *embeddings* gerados para os artigos e para os *tweets*, resultando em uma matriz $i \times j$ de similaridade entre sentenças, sendo “i” a quantidade de sentenças de todos os artigos e “j” a quantidade de *tweets*.

Código 7 – Gerando as *word embeddings* e calculando a similaridade por cosseno

```
1 articleEmbeddings = model.encode(articleSentences, convert_to_tensor=
      True)
2 tweetEmbeddings = model.encode(tweets, convert_to_tensor=True)
3 cosine = util.cos_sim(articleEmbeddings, tweetEmbeddings)
```

A matriz gerada precisa ser interpretada para que se consiga relacionar os resul-

tados da função cosseno com o *tweet* e sentença do artigo. Para isso, é necessário percorrer todas as sentenças dos artigos e, para cada uma, percorrer todos os *tweets*.

O Algoritmo 1 exemplifica a ligação dos resultados às suas fontes. Na prática, a partir dos índices dos laços de repetição, é possível extrair as sentenças de entrada para o qual o cosseno foi calculado. O resultado da extração é adicionado em um dicionário, mapeado pelo identificador do *tweet*, ordenado pelo valor obtido do cosseno (denominado *score*), e que contém uma lista com todos os dados da análise feita com as sentenças dos artigos, como a própria sentença, artigo de origem e o *score*. Para o presente trabalho, é definido um ponto de corte nos valores de similaridade obtidos, buscando sentenças com valores superiores a 0.6, ou seja, com a tendência de serem mais similares, assim, é possível avaliar uma amostra mais específica dentre todas as sentenças avaliadas.

Algoritmo 1: Retorno da extração dos resultados

Data: $listaDeArtigos$, $listaDeTweets$, $sentencas$, $cosseno$
Result: Dicionário de *tweets* com uma lista de artigos

```
1  paresDeDados ← {}
2  foreach sentenca ∈ sentencas do
3      foreach tweet ∈ listaDeTweets do
           // extrai o resultado do cosseno de sentenca e tweet
4          resultado ← cosseno[i][j]
           // busca dados do artigo a partir da sentença
5          dadosArtigo ← listaDeArtigos.find(sentenca)
           // adiciona os dados do artigo à lista de artigos do tweet
6          paresDeDados[tweet].push(dadosArtigo)
7      end
8  end
9  foreach (tweet, artigos) ∈ paresDeDados do
       // filtra artigos pelo score de similaridade usando ponto de corte
10     artigos ← artigos.filter(artigo.score ≤ −0.6 ∨ artigo.score ≥ 0.6)
       // ordena pelo score de similaridade
11     artigos ← artigos.sort(artigo.score)
12 end
13 return paresDeDados
```

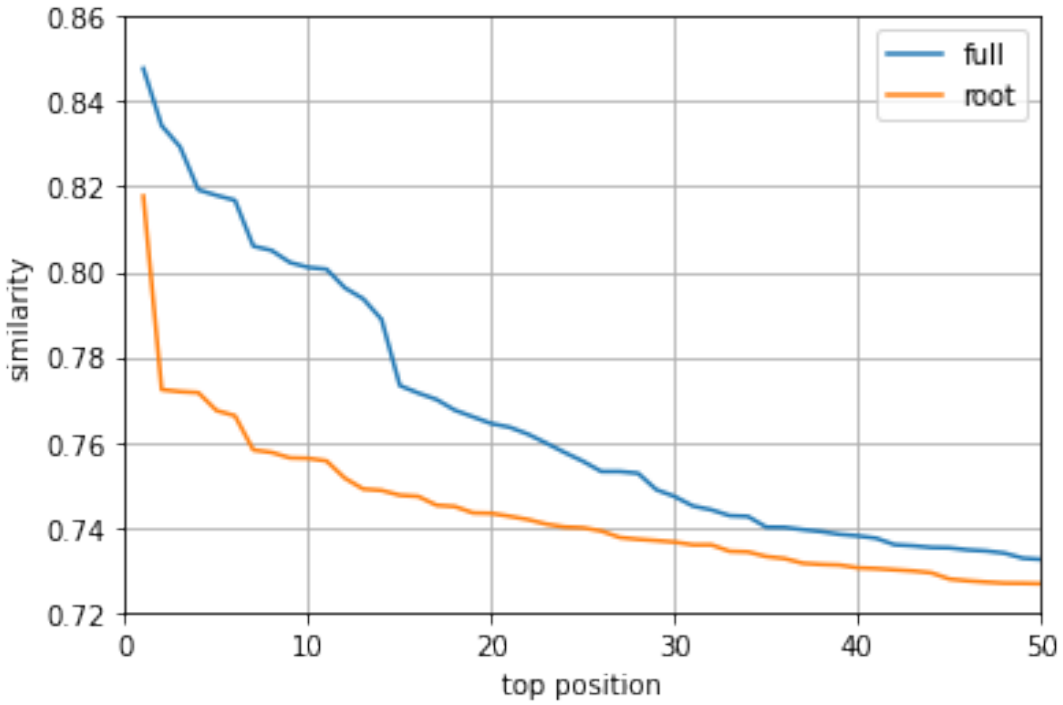
Ao final, os dados são exportados para arquivos “csv” no mesmo diretório onde estão presentes os arquivos gerados na Seção 4.2.1.5. O objetivo da geração do “csv” é promover uma visão melhor sobre os dados obtidos, exibindo as sentenças juntamente do seu *score*.

5 ANÁLISE DE RESULTADOS

Tendo o TF-IDF não se adequado ao cenário, pode-se observar os resultados obtidos pela outra abordagem. Os dados obtidos a partir da análise utilizando o Sentence Transformers, tanto das sentenças originais quando das sentenças pré-processadas, trazem diversas sentenças com alto grau de similaridade. No total, 506 sentenças presentes nos artigos e 500 *tweets* foram utilizados como amostra na Seção 4.2.2.3. Das sentenças originais, ou seja, sem pré-processamento, são 1345 pares de sentenças gerados com similaridade considerando o ponto de corte. Já as pré-processadas são 1923 pares de sentenças (Tabela 4).

A figura abaixo apresenta um gráfico com os 50 maiores valores de similaridade para as sentenças avaliadas. Em azul, com a label "*full*" estão as sentenças originais, sem pré-processamento. Em laranja, com a label "*root*" estão as sentenças pré-processadas.

Figura 5 – 50 similaridades mais altas obtidas pelo *Tweet-Nature Similarity Analyzer*



No entanto, a quantidade de pares e o grau de similaridade gerado não reflete a similaridade ao comparar o par de sentenças que originaram o valor (Tabela 5). Um exemplo onde o cálculo do grau de similaridade resultou em um número muito alto e que não reflete na prática são o *tweet* e a frase de um dos artigos apresentada a seguir.

***Tweet*:** "*The Flu and COVID are both respiratory viruses with the same modes of transmission If the masks and social distancing worked for the neareradication of*

Tabela 4 – Pares gerados no cálculo da similaridade

Tipo da sentença	**Pares gerados**
Sentenças originais	1345
Sentenças pré-processadas	1923

the flu what do you think it should have done for COVID Wake Up"

Artigo: "*Nasal swab samples were first tested by a diagnosticuse viral panel xTAG Respiratory Viral Panel to qualitatively detect 12 common respiratory viruses and subtypes including coronaviruses influenza A and B viruses respiratory syncytial virus parainfluenza virus adenovirus human metapneumovirus and enterovirus/rhinovirus*"

O cálculo do cosseno sobre suas representações vetoriais resulta no valor "0.817779004573822", o mais alto obtido na análise e muito próximo de 1, o que significa que as duas sentenças são muito similares, mas que não é verdadeiramente o caso. Esta situação ocorre em todas as comparações do trabalho, o cálculo de similaridade entre *tweets* e artigos não é efetivo e não é possível afirmar o *matching* de informações. Buscando avaliar locais de falha para a análise feita no presente trabalho, algumas hipóteses são levantadas.

A primeira trata da parte sintática das sentenças. Uma das causas possíveis para este resultado pode estar relacionada ao conflito na forma de escrita das duas fontes, tornando difícil para o modelo a compreensão do contexto da sentença para uma melhor avaliação e atribuição de valor às *word embeddings*. A dificuldade do modelo com a compreensão de formas sintáticas distintas também trás a tona a segunda hipótese, que trata de uma inadequação do modelo pré-treinado escolhido para o presente trabalho, que pode não estar treinado e otimizado com um *dataset* que se enquadre nas avaliações realizadas.

Tabela 5 – Resultados da função cosseno sobre *embeddings* e suas fontes

Tweet	**Trecho do artigo**	**Similaridade**
If you have any doubts over what's worse COVID or vaccinesmasks I highly recommend you check out this segment Freedom isn't free It requires sacrifice from the collective to preserve those freedoms Watch this Wear a mask Get the shot Save lives	Saving 560 Euro per 100 protected healthcare workers in the first wave and 15 Euro in the second Corona wave when using reprocessed masks	≈ 0.60900831
As Ladapo's refusal sparks furor expert says it's good to mask up around those at high COVID risk	Our results highlight the importance of regular changing of disposable masks and washing of homemade masks and suggests that special care must be taken when removing and cleaning the masks	≈ 0.61257416
Masks are a staple of any good Halloween costume — but most costume masks aren't up to snuff when it comes to COVID19 prevention	In characterizing the particle size distribution of the bioaerosol generated current recommendations of standards to calculate the MPS seem outdated and the MAD seems more relevant than does the MPS for characterizing aerosol size using a lognormal distribution	≈ 0.76433241
...		

6 CONCLUSÃO E TRABALHOS FUTUROS

O presente trabalho busca o *matching* de dados a partir de uma análise de similaridade entre os dados extraídos de *tweets* com dados de artigos científicos através da ferramenta desenvolvida, o *Tweet-Nature Similarity Analyzer*, com o objetivo de avaliar se *tweets* publicados propagam informações em conformidade com o conteúdo dos artigos ou não.

Para isso, a construção de Scrapers para dados do Twitter e de artigos da Nature foram necessárias para popular uma base de dados utilizada como fonte para o estudo. A biblioteca Tweepy foi utilizada como interface para a busca por *tweets* da API do Twitter. Apesar da única forma de consulta disponível gratuitamente ser um "endpoint" de publicações mais recentes, conseguiu-se obter as publicações mais recentes sobre o tema abordado: "máscaras e coronavírus". Com uma estrutura de fácil manipulação, o Scrapy com suas Spiders é o responsável pela extração dos artigos da revista Nature e, com o auxílio do XPath, foi possível extrair as informações dos documentos utilizados como amostra de forma simplificada. Além disso, com o auxílio do Docker, pôde-se configurar o ambiente do PostgreSQL para modelá-lo, sendo essa a parte central e estrutural do trabalho, para que o mesmo esteja apto a armazenar os dados extraídos pelos Scrapers.

Contudo, os métodos envolvidos no *matching* não corroboraram um resultado positivo do trabalho. O grau de importância entregue pelo TF-IDF, calculado em uma amostra coletada toda sobre o mesmo tópico fez com que o método não fosse o mais adequado para resultado esperado, diminuindo o grau de termos importantes que, pelo fato de estarem presentes em todos os documentos utilizados, se tornavam comuns e, portanto, eram classificados como pouco relevantes.

O Sentence Transformers se mostrou muito versátil em sua utilização junto de um modelo pré-treinado, necessitando de poucos ajustes adicionais para obter a funcionalidade configurada. No entanto, a função cosseno do Sentence Transformers teve seus resultados também distintos do esperado. Apresentados na Tabela 5, sentenças de significados distintos foram classificadas em um alto grau de similaridade, que não corresponde com o resultado esperado para este trabalho. Dessa forma, outros modelos podem ser testados afim de obter melhores resultados sobre esta classificação. Assim, o presente trabalho conclui parte dos seus objetivos, que dizem respeito à coleta e armazenamento de dados, restando somente o aprimoramento do algorítmo de *matching* e a exibição dos resultados de maneira visual, objetivo dependente do anterior.

Como trabalhos futuros para o *Tweet-Nature Similarity Analyzer*, pode-se sugerir os seguintes itens:

- Analisar e propor outras estratégias ou métodos de similaridade semântica, bem

como outros modelos pré-treinados capazes de realizarem uma melhor classificação de sentenças como forma de sanar a imprecisão dos modelos estudados no presente trabalho;

- Projetar *embeddings* gerados em um plano vetorial para avaliar a disposição dos mesmos;
- Gerar uma núvem de palavras desconhecidas para avaliar como o modelo do SBERT escolhido está percebendo os dados recebidos;
- Gerar uma visualização dos resultados obtidos do *matching*, com o objetivo de facilitar a visualização da classificação dos dados;

Por fim, o código da aplicação desenvolvida está disponível na íntegra em um repositório público do GitHub[1].

[1] `https://github.com/gustavomoser/ftd-tfidf-we`

REFERÊNCIAS

ALLCOTT, Hunt; GENTZKOW, Matthew. Social Media and Fake News in the 2016 Election. **Journal of Economic Perspectives**, v. 31, n. 2, p. 211–36, mai. 2017. DOI: 10.1257/jep.31.2.211. Disponível em: <https://www.aeaweb.org/articles?id=10.1257/jep.31.2.211>. Citado nas pp. 14, 18.

ALMEIDA, Felipe; XEXÉO, Geraldo. **Word Embeddings: A Survey**. [*S.l.*]: arXiv, 2019. DOI: 10.48550/ARXIV.1901.09069. Disponível em: <https://arxiv.org/abs/1901.09069>. Citado na p. 22.

BARCELOS, Thainá do Nascimento de *et al.* Análise de fake news veiculadas durante a pandemia de COVID-19 no Brasil. **Revista Panamericana de Salud Pública**, v. 45, n. 65, mai. 2021. DOI: 10.26633/RPSP.2021.65. Disponível em: <https://doi.org/10.26633/RPSP.2021.65>. Citado nas pp. 14, 18.

CHANDRASEKARAN, Dhivya; MAGO, Vijay. Evolution of semantic similarity—a survey. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 54, n. 2, p. 1–37, 2021. Citado na p. 20.

DEMING, W. Edwards. **Out of the Crisis**. [*S.l.*]: The MIT Press, 1986. v. 1. (MIT Press Books, 0262541157). ISBN ARRAY(0x4ba1f580). Disponível em: <https://ideas.repec.org/b/mtp/titles/0262541157.html>. Citado na p. 15.

FERRARA, Emilio *et al.* Web data extraction, applications and techniques: A survey. **Knowledge-Based Systems**, v. 70, p. 301–323, 2014. ISSN 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2014.07.007. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950705114002640>. Citado na p. 19.

GARCIA, Miguel. **How to make a Twitter Bot in Python with Tweepy**. Disponível em: <https://realpython.com/twitter-bot-python-tweepy/#what-is-tweepy>. Citado na p. 20.

GRAZZIOTIM, Lucas Bortolanza. **Agrupamento de sentenças semanticamente similares aplicado à descoberta de novas intenções para chatbots cognitivos**. [*S.l.*: *s.n.*], mai. 2022. Disponível em: <http://hdl.handle.net/10183/240211>. Citado nas pp. 22, 38, 39.

GUARISE, Lucas; REZENDE, Solange Oliveira. **Detecção de notícias falsas usando técnicas de deep learning**. [*S.l.*: *s.n.*], 2019. Disponível em: <%5Curl%7Bhttps://repositorio.usp.br/directbitstream/494173d1-2375-4063-8b9c-8146811ffe6f/lucas%20guarise.pdf%7D>. Citado nas pp. 26–28.

HUGGING FACE. **Hugging Face**. Disponível em: <https://huggingface.co/>. Acesso em: 15 nov. 2022. Citado na p. 39.

JURAFSKY, Dan; MARTIN, James H. **Speech and language processing (3rd Edition Draft): an introduction to natural language processing, computational linguistics, and speech recognition**. [*S.l.*: *s.n.*], 2021. Disponível em: <https://web.stanford.edu/~jurafsky/slp3>. Citado nas pp. 21, 22, 39.

LAGE, N. **Ideologia e técnica da notícia**. [*S.l.*]: Digitaliza Conteudo, 2021. ISBN 9786588401989. Disponível em: <https://books.google.com.br/books?id=UHo-EAAAQBAJ>. Citado na p. 17.

LIU, Bing. **Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data**. Berlin, Heidelberg: Springer-Verlag, 2009. ISBN 3642072372. Citado na p. 20.

MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHÜTZE, Hinrich. **Introduction to Information Retrieval**. Cambridge, UK: Cambridge University Press, 2008. ISBN 978-0-521-86571-5. Disponível em: <http://nlp.stanford.edu/IR-book/information-retrieval-book.html>. Acesso em: 21 set. 2022. Citado nas pp. 21, 36, 37.

MITCHELL, Ryan. **Web Scraping with Python: Collecting Data from the Modern Web**. 1st. [*S.l.*]: O'Reilly Media, Inc., 2015. ISBN 1491910291. Citado nas pp. 18, 19.

MONTEIRO, Rafael A. *et al.* Contributions to the Study of Fake News in Portuguese: New Corpus and Automatic Detection Results. *In:* PROPOR. [*S.l.*: *s.n.*], 2018. Disponível em: <https://sites.icmc.usp.br/taspardo/PROPOR2018-MonteiroEtAl.pdf>. Citado nas pp. 14, 25–28.

MONTEIRO, Roger *et al.* Classificação de Fake News com Textos de Notícias em Língua Portuguesa Integrando Data Warehousing e Machine Learning. *In:* Disponível em: <https://www.researchgate.net/publication/340633367_Classificacao_de_Fake_News_com_Textos_de_Noticias_em_Lingua_Portuguesa_Integrando_Data_Warehousing_e_Machine_Learning>. Citado nas pp. 26–28.

NATURE. **Formatting guide**. Disponível em: <https://www.nature.com/nature/for-authors/formatting-guide>. Acesso em: 13 ago. 2022. Citado na p. 34.

REIMERS, Nils; GUREVYCH, Iryna. **Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks**. [*S.l.*]: arXiv, 2019. DOI: 10.48550/ARXIV.1908.10084. Disponível em: <https://arxiv.org/abs/1908.10084>. Citado nas pp. 23, 24, 39, 40.

ROVEDA, Ugo. **Google Colab: o que é, como usar e quais são as vantagens?** Nov. 2020. Disponível em: <https://kenzie.com.br/blog/google-colab>. Acesso em: 15 nov. 2022. Citado na p. 36.

SCRAPY. **Scrapy at glance**. Abr. 2021. Disponível em: <https://doc.scrapy.org/en/latest/intro/overview.html>. Acesso em: 10 set. 2021. Citado nas pp. 19, 34.

SENTENCE TRANSFORMERS. **all-mpnet-base-v2**. Disponível em: <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>. Acesso em: 19 nov. 2022. Citado na p. 40.

SMITH, Kit. **60 Incredible and Interesting Twitter Stats and Statistics**. Jan. 2020. Disponível em: <https://www.brandwatch.com/blog/twitter-stats-and-statistics/>. Citado na p. 14.

TANDOC JR., Edson C.; LIM, Zheng Wei; LING, Richard. Defining “Fake News”. **Digital Journalism**, Routledge, v. 6, n. 2, p. 137–153, 2018. DOI: 10.1080/21670811.2017.1360143. eprint: https://doi.org/10.1080/21670811.2017.1360143. Disponível em: <https://doi.org/10.1080/21670811.2017.1360143>. Citado na p. 17.

TWEEPY. **Tweepy Documentation**. Disponível em: <https://docs.tweepy.org/en/stable/client.html>. Acesso em: 18 set. 2022. Citado na p. 32.

UKPLAB, Ubiquitous Knowledge Processing Lab. **Sentence Transformers: Multilingual Sentence, Paragraph, and Image Embeddings using BERT & Co.** Disponível em: <https://github.com/UKPLab/sentence-transformers>. Acesso em: 18 nov. 2022. Citado na p. 39.

VOLPATO, Bruno. **Ranking: as redes sociais mais usadas no Brasil e no mundo em 2021, com insights e materiais gratuitos**. Ago. 2021. Disponível em: <https://resultadosdigitais.com.br/blog/redes-sociais-mais-usadas-no-brasil/>. Acesso em: 12 set. 2021. Citado na p. 18.

Apêndices

APÊNDICE A – CÓDIGO-FONTE

Arquivo: ftd-tfidf-we/.env

```
HOST=localhost
DATABASE=tcc
UNAME=admin
PASSWORD=admin
```

Arquivo: ftd-tfidf-we/assets/articles.txt

```
https://www.nature.com/articles/s41591-020-0843-2
https://www.nature.com/articles/s41598-021-85327-x
https://www.nature.com/articles/s41598-020-72798-7
https://www.nature.com/articles/s41598-021-01005-y
https://www.nature.com/articles/s41598-021-97188-5
```

Arquivo: ftd-tfidf-we/assets/database-schema.sql

```
CREATE TABLE TWEET (
        id BIGINT,
        tweet TEXT NOT NULL,
        CONSTRAINT PK_TWEET_ID PRIMARY KEY (id)
);

CREATE TABLE TWEET_CLEAN (
        id BIGINT,
        tweet TEXT NOT NULL,
        CONSTRAINT PK_TWEET_CLEAN_ID PRIMARY KEY (id),
        FOREIGN KEY (id) REFERENCES TWEET (id)
);

CREATE TABLE ARTICLE (
        id SERIAL,
        title VARCHAR NOT NULL,
        CONSTRAINT PK_ARTICLE_ID PRIMARY KEY (id)
);

CREATE TABLE AUTHOR (
```

```
        id SERIAL,
        article_id INT,
        name VARCHAR NOT NULL,
        CONSTRAINT PK_AUTHOR_ID PRIMARY KEY (id),
        FOREIGN KEY (article_id) REFERENCES ARTICLE (id)
);

CREATE TABLE ARTICLE_SECTION (
        id SERIAL,
        title VARCHAR NOT NULL,
        content TEXT NOT NULL,
        CONSTRAINT PK_SECTION_ID PRIMARY KEY (id, title)
);
```

Arquivo: ftd-tfidf-we/src/db/PostgresManager.py

```
from os import environ, path

import psycopg2
from dotenv import load_dotenv
from psycopg2.extras import execute_batch


class PostgresManager:
    __conn = None

    def config(self):
        load_dotenv(dotenv_path=path.join(path.dirname(__file__), "..",
        ↪  "..", ".env"))
        db = [
            environ.get("HOST"),
            environ.get("DATABASE"),
            environ.get("UNAME"),
            environ.get("PASSWORD"),
        ]
        return db

    def __connect(self):
        params = self.config()
```

```
        self.__conn = psycopg2.connect(
            host=params[0], database=params[1], user=params[2],
            ↪  password=params[3]
        )

    def __cursor(self):
        self.__connect()
        return self.__conn.cursor()

    def __closeCursor(self):
        self.__conn.cursor().close()

    def __closeConn(self):
        self.__conn.close()

    def __commit(self):
        self.__conn.commit()

    def cursor(self):
        return self.__cursor()

    def insert(self, sql):
        try:
            self.__connect()
            cursor = self.__cursor()
            cursor.execute(sql)
            self.__commit()
        except (Exception, psycopg2.DatabaseError) as error:
            print(error)
        finally:
            if self.__conn is not None:
                self.__closeCursor()
                self.__closeConn()

    def insertAndReturnId(self, sql, tuple):
        id = None
        try:
            self.__connect()
            cursor = self.__cursor()
```

```
            cursor.execute(sql, tuple)
            id = cursor.fetchone()[0]
            self.__commit()
        except (Exception, psycopg2.DatabaseError) as error:
            print(error)
        finally:
            if self.__conn is not None:
                self.__closeCursor()
                self.__closeConn()
        return id

    def findOne(self, sql):
        rs = None
        try:
            self.__connect()
            cursor = self.__cursor()
            cursor.execute(sql)
            rs = cursor.fetchone()
        except (Exception, psycopg2.DatabaseError) as error:
            print(error)
        finally:
            if self.__conn is not None:
                self.__closeCursor()
                self.__closeConn()
        return rs

    def fetchAll(self, sql):
        rs = None
        try:
            self.__connect()
            cursor = self.__cursor()
            cursor.execute(sql)
            rs = cursor.fetchall()
        except (Exception, psycopg2.DatabaseError) as error:
            print(error)
        finally:
            if self.__conn is not None:
                self.__closeCursor()
                self.__closeConn()
```

```
        return rs

    def update(self, update, pair, tweets):
        try:
            self.__connect()
            cursor = self.__cursor()
            cursor.execute("PREPARE updateStmt AS " + update)
            execute_batch(cursor, "EXECUTE updateStmt " + pair, tweets, 100)
            cursor.execute("DEALLOCATE updateStmt")
            self.__commit()
        except (Exception, psycopg2.DatabaseError) as error:
            print(error)
        finally:
            if self.__conn is not None:
                self.__closeCursor()
                self.__closeConn()

    def delete(self, table, condition):
        try:
            self.__connect()
            cursor = self.__cursor()
            cursor.execute("DELETE FROM " + table + " WHERE " + condition)
            self.__commit()
        except (Exception, psycopg2.DatabaseError) as error:
            print(error)
        finally:
            if self.__conn is not None:
                self.__closeCursor()
                self.__closeConn()


Arquivo: ftd-tfidf-we/src/common.py

from service.Service import Service

service = Service()


def export():
```

```
    service.export()
```

Arquivo: ftd-tfidf-we/src/twitter/TwitterScraper.py

```
from os import environ, path

import tweepy
from dotenv import load_dotenv


class TwitterScraper:
    def extract_tweets(self, query):
        load_dotenv(dotenv_path=path.join(path.dirname(__file__), "..",
        ↪  "..", ".env"))

        bearer = environ.get("BEARER_TOKEN")

        client = tweepy.Client(bearer_token=bearer, wait_on_rate_limit=True)

        query = f"query -is:retweet"

        result = client.search_recent_tweets(query=query, max_results=100)

        tweets = ["id": tweet.id, "text": tweet.text for tweet in
        ↪  result.data]

        return tweets
```

Arquivo: ftd-tfidf-we/src/articles.py

```
from json import load
from os import listdir, path

from scrapy.crawler import CrawlerProcess

from article.ArticleScraper import ArticleScraper
from service.Service import Service
```

```
service = Service()
path_to_json = path.join(path.dirname(__file__), "..", "assets", "articles")


def __crawlArticles():
    process = CrawlerProcess()
    process.crawl(ArticleScraper)
    process.start()


def __loadArticles(files):
    jsons = []
    for file in files:
        with open(path.join(path_to_json, file)) as json:
            json_text = load(json)
            jsons.append(json_text)
    return jsons


def saveArticles():
    __crawlArticles()
    files = [pos_json for pos_json in listdir(path_to_json)]
    loaded_files = __loadArticles(files)
    service.saveArticles(loaded_files)
```

Arquivo: ftd-tfidf-we/src/util.py

```
# from emoji import UNICODE_EMOJI_PORTUGUESE
import re

from emoji_translate.emoji_translate import Translator


class TweetStr(str):
    def decode_emoji(self):
        emo = Translator(exact_match_only=False, randomize=True)
        self = TweetStr(emo.demojify(self))
```

```
        return self

    def remove_mention(self):
        self = TweetStr(
            re.compile(pattern="
s*@
S+
s+").sub(r"", self).replace("
n", " ")
        )
        return self

    def remove_url(self):
        self = TweetStr(
            re.compile(pattern="
w+:
/2[
d
w-]+(
.[
d
w-]+)*(?:(?:
/[^
s/]*))*")
            .sub(r"", self)
            .strip()
        )
        return self

    def to_string(self):
        return str(self)


def clean(text: str) -> str:
    return Tweet-
    ↪ Str(text).decode_emoji().remove_mention().remove_url().to_string()
```

Arquivo: ftd-tfidf-we/src/service/Service.py

```
import re
from json import dumps
from os import makedirs, path

import nltk
from db.PostgresManager import PostgresManager
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from util import clean

#  NLTK built-in support for dozens of corpora and trained models
nltk.download("punkt", quiet=True)
nltk.download("stopwords", quiet=True)


class Service:
    ps = PorterStemmer()

    def saveTweets(self, tweets):
        pm = PostgresManager()
        insert = ",".join(
            pm.cursor().mogrify("(%s,%s)", (tweet["id"],
            ↪  tweet["text"])).decode("utf-8")
            for tweet in tweets
        )
        pm.insert(
            "INSERT INTO tweet (id, tweet) VALUES "
            + insert
            + " ON CONFLICT DO NOTHING;"
        )

    def cleanTweets(self):
        pm = PostgresManager()
        base_tweets = pm.fetchAll("SELECT * FROM tweet")
        tweets = ["id": tt[0], "tweet": clean(tt[1]) for tt in base_tweets]
        insert = ",".join(
            pm.cursor()
            .mogrify("(%s,%s)", (tweet["id"], tweet["tweet"]))
```

```
            .decode("utf-8")
            for tweet in tweets
        )
        pm.insert(
            "INSERT INTO tweet_clean (id, tweet) VALUES "
            + insert
            + " ON CONFLICT DO NOTHING;"
        )
        pm.delete("tweet_clean", "tweet IS NULL OR tweet = ''")

    def saveArticles(self, articles):
        pm = PostgresManager()
        for article in articles:
            title = article["title"]
            exists = pm.findOne(
                "SELECT 1 FROM article WHERE title = '';".format(title)
            )
            if exists is None:
                id = pm.insertAndReturnId(
                    "INSERT INTO article (title) VALUES (%s) RETURNING id;",
                    ↪ (title,)
                )

                authors_list = article["authors"]
                authors = ",".join(
                    pm.cursor().mogrify("(%s,%s)", (id,
                    ↪ author)).decode("utf-8")
                    for author in authors_list
                )
                pm.insert(
                   "INSERT INTO author (article_id, name) VALUES " + authors
                    ↪ + ";"
                )

                section_list = article["sections"]
                sections = ",".join(
                    pm.cursor()
                    .mogrify(
                        "(%s,%s,%s)",
```

```
                    (id, section["section"],
                    ↪  section["content"].replace("'", "''")),
                )
                .decode("utf-8")
                for section in section_list
            )
            pm.insert(
              "INSERT INTO article_section (id, title, content) VALUES "
               + sections
               + ";"
            )

    def getArticleSections(self):
        pm = PostgresManager()
        rs = pm.fetchAll("SELECT id, content FROM article_section")

        res =
        for result in rs:
            id = result[0]
            content = result[1]
            if id not in res:
                res[id] = content
            else:
                res.update(id: res[id] + " " + content)

        return res

    def getTweets(self):
        pm = PostgresManager()
        rs = pm.fetchAll("SELECT * FROM tweet_clean")

        res =
        for result in rs:
            id = result[0]
            content = result[1]
            res.update(id: content)

        return res
```

```
    def __textToStemTokens(self, text: str) -> list[str]:
        tokens = nltk.word_tokenize(text)
        stem = []
        for word in tokens:
            if (
                re.fullmatch(r"[a-z]+", word)
                and word not in stopwords.words("english")
                and len(word) > 1
            ):
                stem.append(self.ps.stem(word))
        return stem

    def export(self):
        self.exportFullArticleContentToJson()
        self.exportCleanWordsToJson()
        self.exportFullTweetsToJson()
        self.exportCleanTweetsToJson()

    def exportFullTweetsToJson(self):
        self.writeJsonFile(self.getTweets(), "tweet_full.json")

    def exportCleanTweetsToJson(self):
        export =

        tweets = self.getTweets()
        for id, tweet in tweets.items():
            root = self.__textToStemTokens(tweet)
            export.update(id: root)

        self.writeJsonFile(export, "tweet_root_words.json")

    def exportFullArticleContentToJson(self):
        self.writeJsonFile(self.getArticleSections(),
        ↪  "article_full_section.json")

    def exportCleanWordsToJson(self):
        export =

        sections = self.getArticleSections()
```

```
        for id, section in sections.items():
            tokens = self.__textToStemTokens(section)
            export.update(id: tokens)

        self.writeJsonFile(export, "article_root_words.json")

    def writeJsonFile(self, obj: object, filename: str):
        if not obj or not filename:
            return

        json = dumps(obj)

        directory = path.join(path.dirname(__file__), "..", "..",
        ↪  "assets/json")
        filepath = path.join(directory, filename)

        if not path.isdir(directory):
            makedirs(directory, exist_ok=True)

        f = open(filepath, "w")
        f.write(json)
        f.close()
```

Arquivo: ftd-tfidf-we/src/article/ArticleScraper.py

```
from json import dumps
from os import makedirs, path

import scrapy


class ArticleScraper(scrapy.Spider):
    name = "naturespider"

    articles_url = []

    with open(
        path.join(path.dirname(__file__), "..", "..", "assets",
        ↪  "articles.txt"), "r"
```

```
) as file:
    lines = file.readlines()
    for line in lines:
        articles_url.append(line)

start_urls = articles_url
used_sections = [
    "Abstract",
    "Main",
    "Results",
    "Discussion",
    "Methods",
    "Conclusions",
]

def parse(self, response):
    # extract title and create filename
    title = response.css(".c-article-title::text").get()
    filename = f'article-title.lower().replace(" ", "-").json'

    # extract authors
    authors =
    ↪  response.xpath('//a[@data-test="author-name"]/text()').getall()

    # extract body sections
    sections = self.extract_sections(
        response.xpath('//div[@class="c-article-section"]').getall()
    )

    # write file
    self.write_file(
        filename=filename,
        json=self.create_json(
            "title": title, "authors": authors, "sections": sections
        ),
    )

    self.log(f"filename saved!")
```

```
    def extract_sections(self, sections):
        extracted = []
        for s in sections:
            sec = scrapy.Selector(text=s)
            title = sec.xpath(
                '//h2[contains(@class, "c-article-section__title")]/text()'
            ).get()
            if title in self.used_sections:
                content = self.clean_section_content(
                    sec.xpath('//div[@class="c-article-
                    ↪  section__content"]').get()
                )
                extracted.append("section": title, "content": content)
        return extracted

    def clean_section_content(self, content):
        sec = scrapy.Selector(text=content)
        contents = ",".join(sec.xpath("//p/text()").getall())
        return contents

    def create_json(self, object):
        # create object
        return dumps(object)

    def write_file(self, filename, json):
        # create path to articles folder
        dir = path.join(path.dirname(__file__), "..", "..",
        ↪  "assets/articles")
        filepath = path.join(dir, filename)

        if not path.isdir(dir):
            makedirs(dir, exist_ok=True)

        f = open(filepath, "w")
        f.write(json)
        f.close()
```

Arquivo: ftd-tfidf-we/src/google-colab/similarity_analysis.py

```
# -*- coding: utf-8 -*-
"""Análise de Similaridade

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1hlErGbNt0xK0hB-
    ↪ 8mXgS24aOQEw8E7Wz

# Análise de Similaridade entre Tweets e conteúdos de Artigos Científicos

Montando a pasta pessoal do Google Drive para acesso pelo Google Colab
"""

from google.colab import drive
drive.mount('/content/drive')

DATA_DIR = './drive/MyDrive/fake-tweet-detector/assets/'

"""## Leitura dos arquivos e geração de sentenças

Lendo o arquivo completo de artigos e interpretando o JSON como um
↪ dicionário no Python
"""

import json

with open(DATA_DIR + 'article_full_section.json', 'r') as f:
  articles = json.load(f)

"""Utilizando a biblioteca *nltk* como auxílio para a geração de sentenças a
↪ partir do conteúdo de cada um dos artigos.

"""

!pip install nltk

from nltk import tokenize
```

```
import nltk
import re
nltk.download('punkt')

for id, content in articles.items():
  sentences = tokenize.sent_tokenize(content)
  sentences = [re.sub(r'
      ([^()]*
      )|[
      )
      (]|([.,!?()
      ]|[^
      x00-
      x7F]+)', '', sentence).strip() for sentence in sentences]
  articles.update( id:  list(filter(None, sentences)) )

articles['1']

import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
nltk.download("stopwords")

ps = PorterStemmer()

def textToStemTokens(text: str):
  tokens = nltk.word_tokenize(text)
  stem = []
  for word in tokens:
    if (
      re.fullmatch(r"[a-z]+", word)
      and word not in stopwords.words("english")
      and len(word) > 1
    ):
      stem.append(ps.stem(word))
  return stem

tokenizedArticles =
```

```
for id, content in articles.items():
  val = [" ".join(textToStemTokens(sentence)) for sentence in content]
  tokenizedArticles.update( id: val )

"""Lendo os arquivos de tweets e interpretando o JSON como um dicionário no
↪  Python"""

with open(DATA_DIR + 'tweet_full.json', 'r') as f:
  tweets = json.load(f)

with open(DATA_DIR + 'tweet_root_words.json', 'r') as f:
  tweets_root = json.load(f)

"""### Criando uma amostra dos tweets para análise"""

tweet_keys = list(tweets.keys())[:500]

for k in tweet_keys:
  tweets.update( k: re.sub('[.,!?
-]', '', tweets[k]) )

"""## Análise de Similaridade

## TF-IDF
"""

with open(DATA_DIR + 'article_root_words.json', 'r') as f:
  ar = json.load(f)

articles_to_tfidf = []

for id, article_words in ar.items():
  articles_to_tfidf.append(' '.join(article_words))

from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

tfidf = vectorizer.fit_transform(articles_to_tfidf).toarray()
```

```
vectorizer.get_feature_names_out()

article_tfidf_object =

for id in ar.keys():
  article_values = []
  index = int(id) - 1

  word_list = vectorizer.get_feature_names_out()

  for j, word in enumerate(word_list):
    word_value = tfidf[index][j]

    if word_value > 0.0001:
      article_values.append(word: word_value)

  article_tfidf_object.update(id: article_values)

article_tfidf_object

"""gerar tfidf para os tweets e gerar matriz de comparação dos valores
↪  obtidos."""

tweet_root_f = [t for t in tweets_root.items() if len(t[1]) > 0]

tweet_to_tfidf = []

for tweet in tweet_root_f:
  tweet_to_tfidf.append(' '.join(tweet[1]))

tweet_to_tfidf

tweet_tfidf = vectorizer.fit_transform(tweet_to_tfidf).toarray()

tweet_tfidf

tweet_tfidf_object =
```

```
for key, tweet in enumerate(tweet_root_f):
  tweet_values = []

  word_list = vectorizer.get_feature_names_out()

  for j, word in enumerate(word_list):
    word_value = tweet_tfidf[key][j]

    if word_value > 0.0001:
      tweet_values.append(word: word_value)

  tweet_tfidf_object.update(tweet[0]: tweet_values)

tweet_tfidf_object

"""## Sentence Transformers (SBERT)

### Definição do modelo treinado

Aqui se utiliza da biblioteca **sentence_transformers** como forma de
↪  importar um modelo treinado para a análise.
"""

!pip install sentence_transformers

"""O modelo selecionado para o trabalho é o `all-mpnet-base-v2`


"""

from sentence_transformers import SentenceTransformer

model = SentenceTransformer('sentence-transformers/all-mpnet-base-v2')

"""### Gerando *Sentence Embeddings*

* Conjunto de técnicas de PLN onde sentenças são mapeadas para vetores de
↪  números reais,
"""
```

```
from sentence_transformers import util
from torch import Tensor
from typing import List, Dict

articleSentences = []
articleSupLim =

accLim = 0
for articleId, sentences in articles.items():
  for sentence in sentences:
    articleSentences.append(sentence)
  accLim = accLim + len(sentences)
  articleSupLim.update( articleId: accLim - 1 )

articleSupLim

articleEmbeddings = model.encode(articleSentences, convert_to_tensor=True)

tweetEmbeddings = model.encode(list(tweets.values()),
↪  convert_to_tensor=True)

"""### Realizando a análise de similaridade semântica para sentenças
↪  completas"""

cosine = util.cos_sim(articleEmbeddings, tweetEmbeddings)

def getArticleIdByIndex(limits, index):
  for id, limit in limits.items():
    if (index <= limit):
      return id, limit - index

pairs =
cosineValues = []

for i in range(len(articleSentences)):
  for j in range(len(tweet_keys)):
    score = cosine[i][j]
```

```
    tweetKey = tweet_keys[j]

    tweet = tweets.get(tweetKey)

    if tweet:
      articleId, index = getArticleIdByIndex(articleSupLim, i)
      articleSentence = articles.get(articleId)[index]

      sentences = pairs.get(tweetKey)

      if not sentences:
        sentences = []

      sentences.append(
                  'tweet': tweet,
                  'articleId': articleId,
                  'articleSentence': articleSentence,
                  'index': [i, j],
                  'score': float(score.item())
                )

      cosineValues.append(float(score.item()))

      pairs.update( tweetKey: sentences )

cosineValues.sort(reverse=True)
flatCosine = cosineValues[:50]

import operator

for key, sentences in pairs.items():
  sentences.sort(key=operator.itemgetter('score'))
  pairs.update( key: sentences )

"""### Realizando a análise de similaridade semântica a partir de raízes de
↪  palavras

Gerando amostra, embeddings e obtendo os resultados.
"""
```

```
tweetRootSample =

for k in tweet_keys:
  tweetRootSample.update( k: " ".join(tweets_root[k]) )

handledArticleSentences = []
handledArticleSupLim =

accLim = 0
for articleId, sentences in tokenizedArticles.items():
  for sentence in sentences:
    handledArticleSentences.append(sentence)
  accLim = accLim + len(sentences)
  handledArticleSupLim.update( articleId: accLim - 1 )

handledArticleEmbeddings = model.encode(handledArticleSentences,
↪  convert_to_tensor=True)

rootTweetEmbeddings = model.encode(list(tweetRootSample.values()),
↪  convert_to_tensor=True)

root_cosine = util.cos_sim(handledArticleEmbeddings, rootTweetEmbeddings)

rootPairs =
rootCosineValues = []

for i in range(len(handledArticleSentences)):
  for j in range(len(tweet_keys)):
    score = root_cosine[i][j]

    tweetKey = tweet_keys[j]

    tweet = tweetRootSample.get(tweetKey)

    if tweet:
      articleId, index = getArticleIdByIndex(handledArticleSupLim, i)
      articleSentence = tokenizedArticles.get(articleId)[index]
      fullArticleSentence = articles.get(articleId)[index]
```

```
      fullTweet = tweets.get(tweetKey)
      sentences = rootPairs.get(tweetKey)

      if not sentences:
        sentences = []

      scr = float(score.item())
      sentences.append(
                  'tweet': tweet,
                  'fullTweet': fullTweet,
                  'tweetKey': tweetKey,
                  'articleId': articleId,
                  'rootArticleSentence': articleSentence,
                  'fullSentence': fullArticleSentence,
                  'index': [i, j],
                  'score': scr
                )
      rootCosineValues.append(scr)

      rootPairs.update( tweetKey: sentences )

rootCosineValues.sort(reverse=True)
flatRootCosine = rootCosineValues[:50]

for key, sentences in rootPairs.items():
  sentences.sort(key=operator.itemgetter('score'))
  rootPairs.update( key: sentences )

"""### Preparando csv de resultados"""

pointcut = 0.6

nlen = 0
rlen = 0

result =
rootResult =

for key, value in pairs.items():
```

```
  filtered = list(filter(lambda r: r['score'] >= pointcut, value))
  if len(filtered) > 0:
    nlen += len(filtered)
    result.update( key: filtered )

for key, value in rootPairs.items():
  filtered = list(filter(lambda r: r['score'] >= pointcut, value))
  if len(filtered) > 0:
    rlen += len(filtered)
    rootResult.update( key: filtered )

"""### Exportando valores obtidos como CSV"""

import csv

with open(DATA_DIR + 'result.csv', 'w', newline='') as w:
  writer = csv.writer(w,delimiter=':')

  for k, v in result.items():
    writer.writerow(['id do tweet', k])
    for i in v:
      writer.writerow(['tweet', i['tweet']])
      writer.writerow(['sentença', i['articleSentence']])
      writer.writerow(['id do artigo', i['articleId']])
      writer.writerow(['similaridade', i['score']])
      writer.writerow(['índice global', str(i['index'])])
      writer.writerow(['-'])
    writer.writerow(['
n'])

with open(DATA_DIR + 'result_root.csv', 'w', newline='') as w:
  writer = csv.writer(w,delimiter=':')

  for k, v in rootResult.items():
    writer.writerow(['id do tweet', k])
    for i in v:
      writer.writerow(['tweet', i['tweet']])
      writer.writerow(['sentença', i['rootArticleSentence'] ])
      writer.writerow(['tweet completo', i['fullTweet'] ])
```

```
      writer.writerow(['sentença completa', i['fullSentence'] ])
      writer.writerow(['id do artigo', i['articleId'] ])
      writer.writerow(['similaridade', i['score'] ])
      writer.writerow(['índice global', str(i['index']) ])
      writer.writerow(['-'])
    writer.writerow(['
n'])

"""### Gerando gráfico com os maiores graus de similaridade obtidos"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

cosineArray = np.asarray([None] + flatCosine)
rootCosArray = np.asarray([None] + flatRootCosine)

df = pd.DataFrame.from_dict("full": cosineArray, "root": rootCosArray)

df.plot(xlabel='top position', ylabel='similarity', ylim=[0.72,0.86],
↪  xlim=[0,50], grid=True)

Arquivo: ftd-tfidf-we/src/tweets.py

from service.Service import Service
from twitter.TwitterScraper import TwitterScraper

service = Service()


def saveTweets():
    tweets = TwitterScraper().extract_tweets("covid masks")
    service.saveTweets(tweets)


def cleanTweets():
    service.cleanTweets()
```

Arquivo: ftd-tfidf-we/docker-compose.yml

```
version: "1"

services:
  postgres:
    image: postgres:14.5-alpine
    restart: always
    ports:
      - "5432:5432"
    environment:
      POSTGRES_DB: tcc
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: admin

  pgadmin:
    image: dpage/pgadmin4
    environment:
      PGADMIN_DEFAULT_EMAIL: "pgadmin@pgadmin.com"
      PGADMIN_DEFAULT_PASSWORD: "pgadmin"
    ports:
      - "15432:80"
    depends_on:
      - postgres
    networks:
      - postgres-network

networks:
  postgres-network:
    driver: bridge
```

Arquivo: ftd-tfidf-we/pyproject.toml

```
[tool.poetry]
name = "tcc"
version = "0.1.0"
description = ""
authors = ["Gustavo Moser <gustavovbmoser@gmail.com>"]
```

```
packages = [
     include = "article", from = "src" ,
     include = "db", from = "src" ,
     include = "service", from = "src" ,
     include = "twitter", from = "src" ,
]

[tool.poetry.dependencies]
python = "^3.10"
tweepy = "^4.10.1"
Scrapy = "^2.6.2"
psycopg2-binary = "^2.9.3"
python-dotenv = "^0.20.0"
emoji-translate = "^0.1.1"
regex = "^2022.9.13"
spacy = "^3.4.1"
nltk = "^3.7"

[tool.poetry.scripts]
save_articles = "src.articles:saveArticles"
export_jsons = "src.common:export"
save_tweets = "src.tweets:saveTweets"
clean_tweets = "src.tweets:cleanTweets"

[tool.poetry.group.dev.dependencies]
black = "^22.8.0"

[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"
```

APÊNDICE B – ARTIGO

Análise de similaridade entre TF-IDF e modelos contextualizados de linguagem baseados em tokens

Gustavo Vicente Barroso Moser[1]

[1]Departamento de Informática e Estatística – Centro Tecnológico
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

gustavovbmoser@gmail.com

Abstract. *As the access to the media grows, and the popularity of social networks increases, the term "fake news" gets more and more form and space. Twitter is a social network focused on real-time communication, used by millions of users. It is where a lot of news spread and can take large proportions. A current example concerns the false news that have been circulating on health-related topics. While the outbreak of COVID-19 generates a crisis across the planet, many opportunities to create false information regarding the use of masks, vaccines, and others. Therefore, in this context, the objective of the present work is to develop a solution to locate tweets whose veracity can be checked with experiments published in scientific articles, using data matching and semantic similarity, and later classifying them as fraudulent or not.*

Resumo. *Com o crescimento do acesso aos meios de comunicação e a popularização das redes sociais, o termo fake news ganha cada vez mais forma e espaço. O Twitter é uma rede social voltada para comunicação em tempo real utilizada por milhões de usuários, é nela que muitas fake news são divulgadas e conseguem tomar grandes proporções. Um exemplo atual diz respeito às fake news que têm circulado sobre temas voltados à saúde. Com o surto da COVID-19 pelo planeta, muitos viram oportunidades de gerar informações falsas a respeito do uso de máscaras, vacinas, entre outros. Portanto, nesse contexto, o objetivo do presente trabalho é desenvolver uma solução para localizar tweets com informações cuja veracidade possa ser checada com experimentos publicados em artigos científicos, utilizando casamento de dados e similaridade semântica, posteriormente, classificando os tweets como fraudulentos ou não.*

1. Introdução

O termo *fake news* têm exercido o papel de vilão quando se trata do combate à desinformação. Com seu uso popularizado a partir de 2016, durante as eleições americanas [Allcott and Gentzkow 2017], o termo, que simboliza uma falácia, uma mentira, é inimigo dos meios de comunicação, sendo influência direta no crescimento de crises de informação. As redes sociais têm sido vetores essenciais para a disseminação de notícias falsas. Um exemplo é o Twitter, fundado em 2006, que oferece aos seus milhões de usuários um espaço para compartilhamento de conteúdo de maneira prática e simples. Dados mostram que aproximadamente 500 milhões de *tweets* são postados por dia [Smith 2020].

A grave situação sanitária gerada pela COVID-19 se complicou ainda mais devido à rápida disseminação de informações. Por um lado, possibilita ao público o acesso a

mídias confiáveis como fonte de informação, sites e artigos de entidades e órgãos de saúde nacionais e internacionais. No entanto, abre espaço para uma intensa propagação de *fake news* [Barcelos et al. 2021]. Nesta situação, a disseminação de notícias falsas afeta diretamente a eficácia de programas e iniciativas como, por exemplo, campanhas pró-vacinas e pró-máscaras.

Há um desempenho ruim por parte dos humanos em separar informações verdadeiras de *fake news*, um fator que pode ser influente na alta disseminação [Monteiro et al. 2018]. A partir dos problemas citados acima, a tarefa de classificar *tweets* como informações falsas e identificá-las se torna ainda mais difícil. Com base nesta problemática, a proposta deste trabalho é o desenvolvimento de uma ferramenta, o *Tweet-Nature Similarity Analyzer* (TNSA), responsável por extrair dados de artigos científicos comprovados e de *tweets*, e após a extração, utilizando técnicas de *data matching* e similaridade semântica a partir dos *tweets* e artigos, realiza o casamento dos dados com técnicas que não usem de *machine learning* para a análise dos dados, como TF-IDF e modelos contextualizados de linguagem baseados em tokens. Dessa forma, com o resultado da análise é possível avaliar se o que está sendo afirmado na conclusão do artigo é o que vem sendo propagado, ou seja, se são ou não *fake news*.

1.1. Objetivos

O presente trabalho resulta na ferramenta *Tweet-Nature Similarity Analyzer*, que realiza uma *match* entre *tweets* e artigos científicos buscando, desta forma, avaliar se os *tweets* estão propagando informações de acordo com o que é apresentado no artigo. Para isso, dados do Twitter e de artigos de revistas científicas são extraídos e tratados através de estruturas de coleta, para então serem analisados e o *match* seja feito.

Os objetivos específicos são os seguintes:

- Desenvolver um Scraper para coleta e extração de *tweets* utilizando as palavras-chave ”covid mask”;
- Desenvolver um Scraper para coleta e extração de dados de artigos científicos utilizando as palavras-chave ”covid mask”;
- Armazenar dados de *tweets* e artigos em um banco de dados;
- Implementar o *match* dos dados entre os dados de artigos e *tweets* utilizando TF-IDF / modelos contextualizados de linguagem baseados em tokens;
- Apresentar a avaliação do resultado da análise de dados de forma visual (por meio de gráficos);

2. Trabalhos relacionados

Esta seção apresenta alguns trabalhos relacionados a *fake news* e sobre a criação de ferramentas para análise e detecção das mesmas no cenário da língua portuguesa. Todos os estudos relatam a falta de bases de dados para o português brasileiro.

2.1. *Contributions to the Study of Fake News in Portuguese: New Corpus and Automatic Detection Results*

O estudo investiga o problema da detecção de *fake news* na língua portuguesa e, inspirado por iniciativas em outras linguagens, introduz uma coleção de notícias de referência para o português, composta por notícias verdadeiras e falsas, as quais foram analisadas com o

objetivo de descobrir características linguísticas de cada uma. Após a avaliação em cima da coleção de dados, utilizando técnicas de aprendizado de máquina e aplicando ideias de estudos anteriores para outras linguagens, foram realizados testes sobre a detecção automática de *fake news* [Monteiro et al. 2018].

Além disso, buscando criar uma coleção de confiança, uma coleção de amostras reais escritas em português foram coletadas e rotuladas. A coleção, chamada de "Fake.br Corpus" é composta de notícias reais e fraudulentas com foco apenas no português brasileiro. Ao todo, foram coletadas 7200 notícias, com exatas 3600 verdadeiras e 3600 notícias falsas [Monteiro et al. 2018].

Todas as *fake news* disponíveis durante o período de coleta foram coletadas e analisadas manualmente. Apenas as notícias completamente falsas foram filtradas. A partir disso, utilizando de um *web crawler*, foram coletadas noticias das maiores agências de notícias do Brasil (G1, Folha de São Paulo e Estadão). O *crawler* procurou nas páginas da web dessas agências por palavras-chave das notícias falsas, que ocorriam nos títulos dessas notícias ou que fossem mais frequentes nos textos.

A partir da coletânea de notícias, partiu-se para a criação de um classificador automático de *fake news*. Foram executados alguns testes utilizando aprendizado de maquina sobre a coletânea, utilizando algumas técnicas de diferentes paradigmas e truncando ou não o texto. Os resultados obtidos ficaram acima das expectativas e cita que um fator que pode explicar os resultados ótimos é a filtragem das notícias com "meias verdades", que elas tornariam o processo mais complexo [Monteiro et al. 2018].

2.2. Detecção de notícias falsas utilizando técnicas de *Deep Learning*

Com o objetivo de gerar um classificador de notícias em português brasileiro que seja capaz de identificar *fake news* utilizando técnicas de *deep learning*, o trabalho de [Guarise and Rezende 2019] surge da falta de existência de conteúdo acadêmico sobre detecção de notícias falsas por software para a língua portuguesa, principalmente pela falta de uma base de dados com notícias previamente classificadas.

O trabalho utiliza a coleção "Fake.br", desenvolvida por [Monteiro et al. 2018], como base de dados para a implementação, contudo, houve a necessidade de realizar um pré-processamento das notícias da base de dados, que são constituídos apenas dos textos puros das notícias, contendo um arquivo ".txt" para cada uma das notícias [Guarise and Rezende 2019]. A estrutura hierárquica de textos permitiu a escolha de uma entre as diversas arquiteturas de redes neurais existentes, a HAN, ou *Hierarchical Attention Networks*, que utiliza esta característica do texto para analisar os dados de entrada [Guarise and Rezende 2019].

O modelo HAN permite que os resultados sejam visualizados de forma que as palavras e sentenças mais determinantes fiquem destacadas, possibilitando a classificação através de um mapa de calor, destacando as sentenças mais importantes na classificação [Guarise and Rezende 2019]. É a partir do HAN que o classificador foi desenvolvido.

O trabalho atingiu, após implementação e treinamento do modelo, resultados similares a outros trabalhos considerados como o estado da arte da classificação de *fake news* para a língua inglesa e obteve um bom desempenho dentro da base de dados escolhida para o experimento do projeto [Guarise and Rezende 2019].

2.3. Classificação de *Fake News* com Textos de Notícias em Língua Portuguesa Integrando *Data Warehousing* e *Machine Learning*

O trabalho desenvolvido por [Monteiro et al. 2019] trata sobre a temática da classificação de *fake news*. A proposta foi utilizar técnicas de aprendizado de máquina para classificar textos de notícias falsas para uma posterior aplicação ao processo de ETL de um *Data Warehouse*, além da geração de um ambiente de consulta que, disponibilizado, contribuísse para trabalhos futuros. Assim, um *dataset* foi criado e foram avaliados os métodos de classificação Regressão Logística, AdaBoost, Naive Bayes e SVM. O melhor método foi utilizado em um sistema de avaliação online.

Utilizando Python e a biblioteca Beautiful Soup no desenvolvimento de um *web crawler* para a coleta de notícias verdadeiras e fraudulentas para a construção de um *dataset*, constituído de 1744 títulos e corpo de notícias falsas coletadas dos sites *boatos.org* e *g1.globo.com/fato-ou-fake*. Além disso, foram também coletados 3185 títulos de notícias verdadeiras coletadas do site *brasil.elpais.com* [Monteiro et al. 2019].

Os algoritmos avaliados eram acoplados à ETL e, a partir da fonte de dados, as notícias eram classificadas automaticamente, incrementando a precisão do classificador. Uma interface web também foi construída, desta forma, os usuários poderiam submeter links de notícias para validar se a mesma é verdadeira ou não.

Os teste inicialmente ocorreram apenas com os títulos das notícias, depois com o corpo e, por fim, com o corpo junto ao título. Os resultados obtidos por [Monteiro et al. 2019] mostraram que o método de Naive Bayes obteve o melhor desempenho, pelo fato de apresentar a maior precisão entre os algoritmos avaliados, além de ser um método de aprendizado incremental, ou seja, o algoritmo é treinado enquanto opera.

3. *Tweet-Nature Similarity Analyzer*

Nesta seção é introduzida a ideia do projeto a ser desenvolvido, o TNSA. Primeiramente, aborda-se o conceito do projeto, apresentando uma visão geral dos módulos e a arquitetura elaborada para o mesmo. Após isso, a implementação é apresentada, tratando de informações técnicas como linguagens, bibliotecas e estratégias que são utilizadas no desenvolvimento.

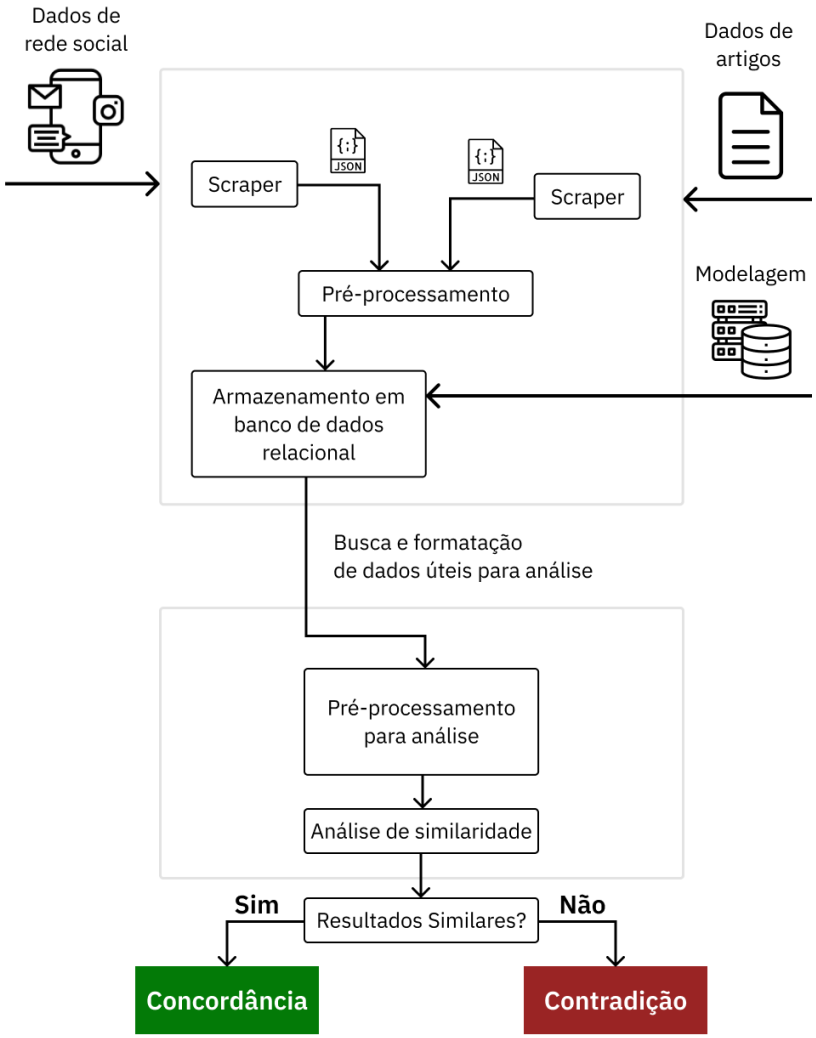
3.1. Visão geral

A proposta de desenvolvimento é que, através do *match* dos dados analisados, seja possível avaliar se os *tweets* estão propagando informações de acordo com o que os artigos científicos publicados por organizações reconhecidas afirmam. A Figura 1 apresenta a arquitetura da implementação definida para este trabalho., que consiste de dois módulos. O primeiro, módulo de coleta e tratamento, é responsável por coletar os dados de suas respectivas fontes, realizar um pré-processamento dos dados e armazená-los. Já o segundo, módulo de análise, é responsável por realizar a análise de similaridade a partir dos dados armazenados e disponibilizar os resultados obtidos.

3.2. Implementação

O desenvolvimento do TNSA utiliza a linguagem de programação Python, juntamente com a ferramenta Poetry, responsável pelo gerenciamento de dependências, como bibliotecas e scripts, no trabalho.

Figure 1. Modelo arquitetural do *Tweet-Nature Similarity Analyzer*



As bibliotecas Tweepy e Scrapy são utilizadas no o desenvolvimento dos *scrapers*, responsáveis por auxiliar na extração dos *tweets* e das seções dos artigos científicos da revista Nature, respectivamente. Além disso, como limitante para o escopo do trabalho, todos os dados coletados são do idioma inglês.

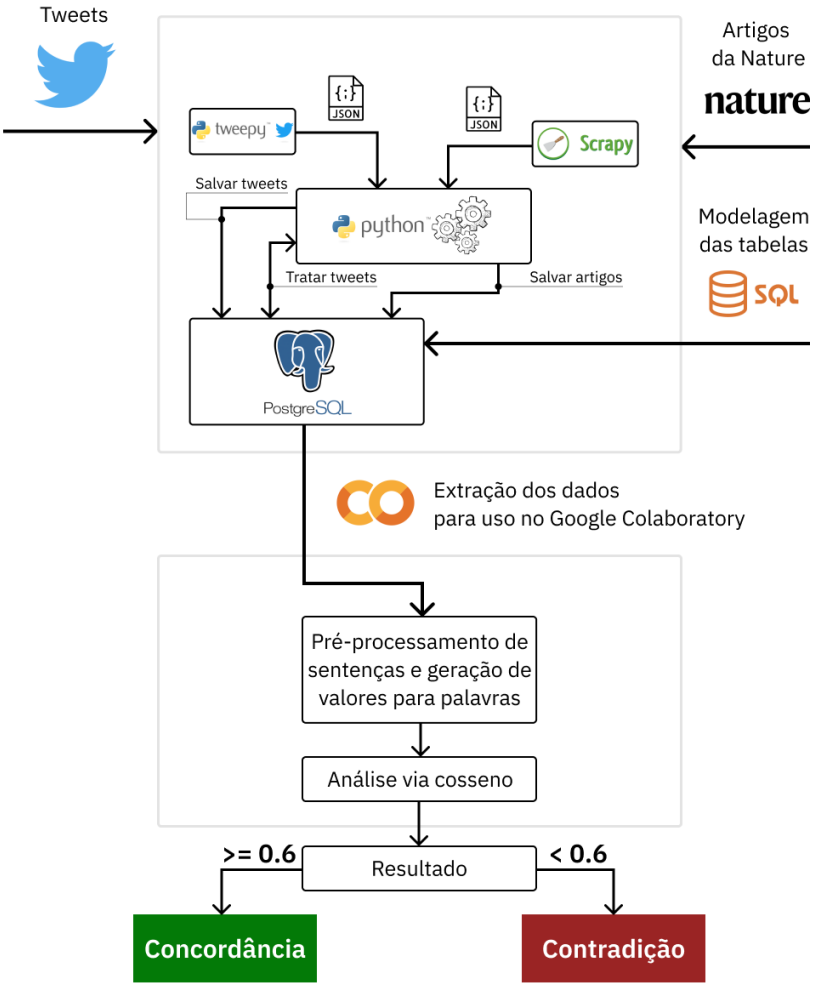
O PostgreSQL é utilizado como SGBD para banco de dados relacional, sendo o mesmo disponibilizado através de um *container* Docker, que é uma plataforma aberta para desenvolvimento, envio e execução de aplicações.

Além do PostgresSQL, a biblioteca *psycopg2*, responsável por interfacear as transações com o banco, também é utilizada no trabalho. A biblioteca NLTK (Natural Language Toolkit) é utilizada em processamento de sentenças, responsável por separação de sentenças, tokenização e remoção de *stop words*.

O Sentence Transformers é a biblioteca responsável por gerar vetores que sim-

bolizam as palavras de uma sentença, bem como oferecer funções de análise semântica. A Figura 2 apresenta o modelo de implementação tecnológica da ferramenta, desenvolvida a partir do modelo arquitetural.

Figure 2. Modelo de implementação do *Tweet-Nature Similarity Analyzer*



4. Extração e tratamento de tweets

Antes de iniciar o desenvolvimento do extrator responsável por coletar os *tweets*, é necessário realizar um cadastro na plataforma de desenvolvedores do Twitter, pois é a partir desta conta que se obtém o token de acesso (*Bearer*), necessário para fazer chamadas aos *endpoints* do Twitter.

O Tweepy é uma interface aos *endpoints* disponibilizados na API do Twitter, contendo métodos que requisitam os dados desses *endpoints*. Após a geração do token de acesso, o Tweepy pode ser inicializado e utilizado, sendo definido para esse trabalho na

classe *TwitterScraper*. A classe *Client* do Tweepy é a responsável por gerenciar o acesso aos métodos relacionados com *endpoints* por meio do token de acesso.

O resultado da busca é uma estrutura de dicionário, trazendo uma lista de objetos no qual cada objeto representa a estrutura de um *tweet*, contendo o texto da publicação, o autor, quantas curtidas ou *retweets* tiveram, data de publicação, localização geográfica, e outros atributos. Para este trabalho apenas o texto e seu identificador são utilizados. O identificador é necessário para armazenamento, sendo utilizado como chave primária da tabela *tweet*, garantindo unicidade dos dados coletados. Após a busca, os dados encontrados são persistidos no banco de dados.

No presente trabalho, em um período de 1 mês, são coletados os *tweets*. Uutilizando da busca pelos *tweets* mais recentes da plataforma com a *query* "covid masks -is:retweet", o processo é executado repetidamente até obter-se uma quantidade de dados julgada suficiente, totalizando 2130 *tweets*.

Com as publicações do Twitter armazenadas no banco de dados, é possível analisar a estrutura dos textos. A partir dessa análise, pode-se observar alguns elementos presentes nas publicações coletadas que podem ser tratados, de forma a simplificar o texto e auxiliar no cálculo de um resultado mais preciso. Os elementos encontrados que necessitam ser avaliados são emojis, menções e *hyperlinks*. Apenas os emojis são realmente tratados, sendo convertidos para o seu significado. Os outros elementos não apresentaram utilidade na comparação de dados.

5. Extração e tratamento de artigos científicos

O processo de extração dos artigos inicia-se com a seleção dos artigos que fazem parte da amostra para o desenvolvimento do presente trabalho. Os cinco artigos escolhidos são da revista científica Nature, sendo buscados pelas palavras-chave "masks" e "covid". O motivo da escolha pela Nature se dá pelo fato de que todos os artigos científicos da revista seguem um Guia de Formatação [Nature] para serem publicados, e por seguirem este guia, auxiliam no desenvolvimento de um *Scraper* mais preciso e eficiente, orientado pela estrutura do artigo.

Para utilizar o Scrapy e extrair as informações necessárias dos artigos, é necessário criar uma *Spider*, que são classes que o Scrapy utiliza para coletar as informações de um ou mais *websites*. A *Spider* implementada deve herdar a classe *Spider* do Scrapy e deve ser desenvolvida definindo as solicitações iniciais a serem feitas e como analisar o conteúdo da página baixada para extrair dados. Opcionalmente, pode ser definido a forma como a *Spider* deve seguir os links nas páginas [Scrapy 2021].

A partir de uma lista de URLs, as URLs dos artigos, as requisições para obter o conteúdo da página iniciam e o método *parse* é ser invocado. O *parse* é o método responsável por tratar a resposta baixada para a requisições feita a uma URL, e a resposta é o conteúdo da página, ou seja, o código HTML.

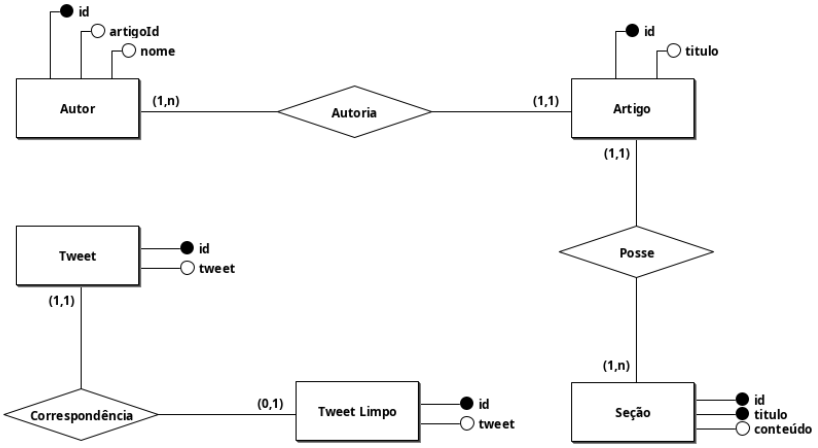
Com a página baixada pela *Spider*, a buscar dados nas estruturas da página a partir de elementos HTML e suas propriedades, como a *class* é realizada utilizando o XPath. Da página dos artigos são extraídos o título, os autores, as sessões presentes e o conteúdo das sessões. Além disso, as sessões são filtradas para que apenas texto seja encontrado, desconsiderando imagens e outros elementos. Cada extração de artigo resulta

em um Dicionário, que é construído com o objetivo de ser convertido e exportado como um arquivo JSON. Após a exportação o arquivo JSON é utilizado, sendo lido e tendo suas informações persistidas no banco de dados.

6. Modelagem e armazenamento dos dados

Durante o desenvolvimento dos extratores de dados, surge a necessidade de armazenar os dados coletados. Avaliando as propriedades que seriam armazenadas do artigo e do *tweet*, percebe-se que os dados obtidos são estruturados e com algumas relações, podendo ser armazenado de forma organizada em uma tabela. Portanto, um banco de dados do tipo relacional é escolhido para ser utilizado. Como SGBD, o PostgreSQL, um dos mais avançados, desenvolvido como projeto de código aberto. Assim, o diagrama ER (Entidade Relacionamento) 3 é desenvolvido com o propósito de facilitar a visualização das entidades e atributos que estão presentes no trabalho antes do desenvolvimento da comunicação dos serviços de extração dos dados. A partir dele, o esquema do banco de dados é gerado.

Figure 3. Modelo ER do banco de dados do *Tweet-Nature Similarity Analyzer*



Com o esquema já definido no PostgreSQL, a estrutura utilizada para persistir e consultar os dados obtidos dos processos presentes nas Seções 4 e 5 pode ser desenvolvida.

7. Geração de arquivos intermediários

Durante o desenvolvimento do trabalho há a necessidade de discutir a transferência de um dos módulos para outro ambiente de desenvolvimento com maior capacidade de processamento, dessa forma, uma análise mais rápida pode ser realizada utilizando uma infraestrutura mais robusta.

O Google Collaboratory, é um serviço de armazenamento em nuvem de *notebooks*, hospedado pelo Jupyter Notebook, voltados à criação e execução de códigos em Python, diretamente em um navegador. Com ele é possível ler, desenvolver e rodar códigos e *rich texts* em documentos interativos que agrupam células de códigos, os notebooks, e mantê-los online [Roveda 2020]. O poder computacional utilizado para a execução do software é fornecido pela nuvem de computadores da Google, assim, é possível processar uma grande quantidade de dados.

Para migrar os dados coletados para arquivos que podem ser importados no Google Colab, há uma funcionalidade desenvolvida neste trabalho que exporta versões distintas do conteúdo dos dados armazenados, destinados ao *data matching*, como possíveis formatos necessários para sua execução. Os artigos e *tweets* são exportados de duas formas:

1. **Conteúdo completo** (*tratado*): Os dados são exportados por completo para um JSON. Os *tweets* presentes na tabela *tweet_clean* são exportados com identificador e texto para um arquivo só com *tweets*. Já os artigos têm o conteúdo de todas as suas seções concatenado, sendo gerado um arquivo com identificador e o conteúdo completo dos artigos.
2. **Raízes de palavras**: Utilizando a técnica de *stemming* junto da biblioteca NLTK (Natural Language Toolkit), os artigos e *tweets* são manipulados para que o retorno seja uma lista de raízes de palavras gerada a partir do conteúdo completo, processo denominado *stemming* [Manning et al. 2008]. O algoritmo de *stemming* escolhido foi o *Porter Stemmer*, que é um processo para remover as terminações morfológicas mais comuns da língua inglesa. Além disso, *stop words* existentes no texto são removidas do resultado. Também exporta os dados para um JSON.

Como resultado do processo de geração dos arquivos com os dados coletados, são gerados quatro arquivos, dois para *tweets* e dois para artigos.

8. Implementação da análise dos dados extraídos

O *data matching* é o principal módulo da aplicação. É através dela que os dados são comparados, isto é, mensurados por um algoritmo que retorna a avaliação de concordância semântica entre duas sentenças. Ou seja, é o responsável por avaliar se os *tweets* propagados seguem ou não os estudos publicados.

Existem algumas técnicas para realizar o casamento entre dados. No presente trabalho duas técnicas são abordadas, a primeira é utilizando o TF-IDF, que é uma medida estatística, para encontrar a relevância em um documento, já a outra é utilizando o *framework* Sentence Transformers junto da função Cosseno para análise de similaridade. Ambas as técnicas e aplicações são melhores descritas abaixo.

9. *Data matching* com TF-IDF

Como método escolhido na primeira abordagem, o TF-IDF é utilizado buscando gerar o peso, também chamado de grau de importância, das palavras do artigo e dos *tweets* [Manning et al. 2008], que serão calculados com o objetivo de gerar o *matching* entre os dados. O TF-IDF é calculado para os artigos considerando cada artigo é um documento. O mesmo ocorre para os *tweets*, mas neste caso, cada publicação é considerada como um documento.

A biblioteca *sklearn*, utilizada neste trabalho, fornece a classe *TfidfVectorizer*, que contém a função que realiza o cálculo do TF-IDF sobre uma lista de documentos, retornando uma matriz com os pesos de cada uma das palavras para cada um dos documentos.

Inicialmente, após a extração dos dados, percebe-se que a forma que um *tweet* e um artigo são escritos é sintaticamente muito distinta. Enquanto o artigo traz uma formalidade na formação das suas sentenças, o *tweet* é mais informal, contendo gírias,

Table 1. Exemplo de raízes de palavras de um *tweet* e seus respectivos pesos

Raiz da palavra	**TF-IDF**
dish	0.426358440518873
dispos	0.024829323185419597
dissemin	0.0074778979399909625
distanc	0.012414661592709798
distribut	0.10552462353803328
emit	0.04486738763994577
extrem	0.014955795879981925
hole	0.14829858800656454
knowledg	0.018537323500820567
mask	0.030915968901549056

abreviações, entre outros elementos. Assim, é necessária uma forma de aproximar os *tweets* do formato de escrita do artigo. Neste caso, os arquivos gerados na Seção 7 que retornam as raízes das palavras são uma possível solução, pois apenas as raízes estão presentes e com poucas variações de escrita. Assim, é sobre elas que o TF-IDF é calculado.

Contudo, observando a Tabela 1 com os valores extraídos de um dos vetores de TF-IDF para um *tweet* da base, é possível perceber que a raiz da palavra *mask*, considerada uma palavra importante para os dados deste trabalho, tem um peso atribuído bem mais baixo que raízes de palavras que não estão diretamente relacionadas ao contexto da "efetividade de máscaras contra a doenças respiratórias (como a COVID-19)", como *dish* e *hole*, indicando que essa palavra é menos importante.

Portanto, percebe-se um problema ao calcular a estatística TF-IDF sobre as amostras deste trabalho, já que o esperado é que *mask* seria uma palavra mais importante no contexto dos documentos. Como todos os artigos e *tweets* presentes na amostra abordam o mesmo tópico, palavras que deveriam ser importantes são classificadas como não tão relevantes para o algoritmo, pois aparecem em todos os documentos. Assim, com a limitação do tema dos documentos da amostra impactando no TF-IDF, a métrica não se adaptou ao cenário de estudo.

9.1. Métrica de similaridade por cosseno

[Grazziotim 2022] cita em seu trabalho que uma forma intuitiva de se definir uma similaridade entre dois vetores u e v é medindo a distância entre eles.

Uma das formas mais comuns de se obter medidas de similaridade é através da similaridade por cosseno (do inglês Cosine-Similarity). Calculada em 1, apenas a orientação dos vetores é considerada, descartando assim a sua magnitude. Como os valores estão limitados ao intervalo do cosseno, o $[-1, 1]$, a interpretação do resultado da operação é direta. Similaridades com valor tendendo a 1 representam vetores apon-

tando para a mesma direção, ou seja, sentenças similares, já similaridades com valor tendendo a -1 representam valores apontando para direções opostas, ou seja, sentenças diferentes. Esse cálculo é baseado na operação de produto escalar, da álgebra linear, que age como métrica de similaridade. Logo, o valor tende a ser alto apenas quando vetores tem grandes valores nas mesmas direções, e baixo quando vetores possuem zeros em diferentes direções, ficando com produto escalar 0, indicando dissimilaridade. Contudo, sendo os valores de frequências não negativos, o valor de cosseno para esses vetores varia apenas de 0 a 1 [Jurafsky and Martin 2021].

$$sim\_cos(u, v) = \frac{u * v}{||u|| * ||v||} \tag{1}$$

Independente da redução de dimensionalidade dos vetores das sentenças, a similaridade por cosseno se mantém dentro do mesmo intervalo fechado, devolvendo um resultado mais consistente e intuitivo [Grazziotim 2022] e, portanto, é a forma de medir a similaridade entre sentenças utilizada no presente trabalho.

9.2. *Data matching* com Sentence Transformers

Os arquivos citados na Seção 7 são importados no ambiente do Google Colaboratory para pré-processamento e análise. Cada arquivo, contendo todo o conteúdo de um artigo concatenado em uma única *string*, é lido e tratado, sendo primeiramente separado em sentenças utilizando o NLTK. Há a aplicação de uma expressão regular com o objetivo de remover algumas estruturas de cada sentença, como pontuação, alguns caracteres especiais e caracteres Unicode. Dois dicionários são resultantes deste processo, um diretamente após o processo acima. O segundo dicionário é gerado a partir do primeiro, no entanto, cada sentença passa por um processo de *stemming* (semelhante ao realizado na Seção 7), formando as novas sentenças apenas com as raízes das palavras apenas. O resultado do pré-processamento é apresentado na Tabela 2.

O *framework* Sentence Transformers fornece um método rápido para computar representações vetoriais para sentenças, parágrafos e imagens, provendo uma série de modelos pré-treinados voltados para a computação de sentenças e *word embeddings* para mais de 100 linguagens, aperfeiçoados para diversos casos de uso [UKPLab 2022]. Tais *embeddings* podem ser comparados com similaridade por cosseno para encontrar significados similares entre sentenças. A ferramente ainda permite o treinamento e *fine-tunning* de redes próprias. O trabalho de [Reimers and Gurevych 2019] é parte integrante desta ferramenta. Os modelos *open source* pré-treinados estão disponíveis na plataforma Hugging Face, repositório existente para criação, treino e disponibilização de modelos [Hugging Face 2022].

O modelo escolhido para incorporar as sentenças presentes na base de dados deste trabalho é o *all-mpnet-base-v2*, modelo com a arquitetura padrão do BERT-base (de 768 dimensões), que provém a melhor qualidade dentre os disponibilizados. Os dados de treino deste modelo resultam da concatenação de diversos *datasets* para aprimorar o modelo [Sentence Transformers 2022].

Após configurar o Sentence Transformers com o modelo escolhido, as *word embeddings* podem ser calculadas para todas as sentenças. Todas as sentenças são concatenadas em uma única lista para que este processamento seja realizado em lote e apenas uma

Table 2. Pré-processamento de sentenças

Sentença original	Method 2 corresponds to direct colony counting using an HD counter and converting the count into viable particles (7659\u2009±\u20091177 counts)
Sentença pré-processada	Method 2 corresponds to direct colony counting using an HD counter and converting the count into viable particles
Raízes da sentença	correspond direct coloni count use counter convert count viabl particl

vez para todos os dados. Os *embeddings* gerados são convertidos para um objeto do tipo Tensor, através do parâmetro "convert_to_tensor". Este objeto é esperado na função de cálculo de similaridade por cosseno disponibilizada pelo próprio Sentence Transformers [Reimers and Gurevych 2019].

A função cosseno é executada utilizando os *embeddings* gerados para os artigos e para os *tweets*, resultando em uma matriz $i \times j$ de similaridade entre sentenças, sendo "i" a quantidade de sentenças de todos os artigos e "j" a quantidade de *tweets*. A matriz gerada precisa ser interpretada para que se consiga relacionar os resultados da função cosseno com o *tweet* e sentença do artigo. Para isso, é necessário percorrer todas as sentenças dos artigos e, para cada uma, percorrer todos os *tweets*.

O Algoritmo 1 exemplifica a ligação dos resultados às suas fontes. Na prática, a partir dos índices dos laços de repetição, é possível extrair as sentenças de entrada para o qual o cosseno foi calculado. O resultado da extração é adicionado em um dicionário, mapeado pelo identificador do *tweet*, ordenado pelo valor obtido do cosseno (denominado *score*), e que contém uma lista com todos os dados da análise feita com as sentenças dos artigos, como a própria sentença, artigo de origem e o *score*. Para o presente trabalho, é definido um ponto de corte nos valores de similaridade obtidos, buscando sentenças com valores superiores a 0.6, ou seja, com a tendência de serem mais similares, assim, é possível avaliar uma amostra mais específica dentre todas as sentenças avaliadas.

Ao final, os dados são exportados para arquivos "csv" no mesmo diretório onde estão presentes os arquivos gerados na Seção 7. O objetivo da geração do "csv" é promover uma visão melhor sobre os dados obtidos, exibindo as sentenças juntamente do seu *score*.

10. Análise dos resultados

Tendo o TF-IDF não se adequado ao cenário, pode-se observar os resultados obtidos pela outra abordagem. Os dados obtidos a partir da análise utilizando o Sentence Transformers, tanto das sentenças originais quando das sentenças pré-processadas, trazem diversas sentenças com alto grau de similaridade. No total, 506 sentenças presentes nos artigos e 500 *tweets* foram utilizados como amostra em 9.2. Das sentenças originais, ou seja, sem pré-processamento, são 1345 pares de sentenças gerados com similaridade considerando o ponto de corte. Já as pré-processadas são 1923 pares de sentenças (Tabela 3).

A figura abaixo apresenta um gráfico com os 50 maiores valores de similaridade para as sentenças avaliadas. Em azul, com a label "*full*" estão as sentenças originais, sem

Algoritmo 1: Retorno da extração dos resultados

Data: $listaDeArtigos$, $listaDeTweets$, $sentencas$, $cosseno$
Result: Dicionário de *tweets* com uma lista de artigos

$paresDeDados \leftarrow \{\}$
foreach $sentenca \in sentencas$ **do**
  foreach $tweet \in listaDeTweets$ **do**
    // extrai o resultado do cosseno de *sentenca* e *tweet*
    $resultado \leftarrow cosseno[i][j]$
    // busca dados do artigo a partir da sentença
    $dadosArtigo \leftarrow listaDeArtigos.find(sentenca)$
    // adiciona os dados do artigo à lista de artigos do *tweet*
    $paresDeDados[tweet].push(dadosArtigo)$
  end
end
foreach $(tweet, artigos) \in paresDeDados$ **do**
  // filtra artigos pelo *score* de similaridade usando ponto de corte
  $artigos \leftarrow artigos.filter(artigo.score \leq -0.6 \vee artigo.score \geq 0.6)$
  // ordena pelo *score* de similaridade
  $artigos \leftarrow artigos.sort(artigo.score)$
end
return $paresDeDados$

pré-processamento. Em laranja, com a label "*root*" estão as sentenças pré-processadas.

Figure 4. 50 similaridades mais altas obtidas pelo *Tweet-Nature Similarity Analyzer*

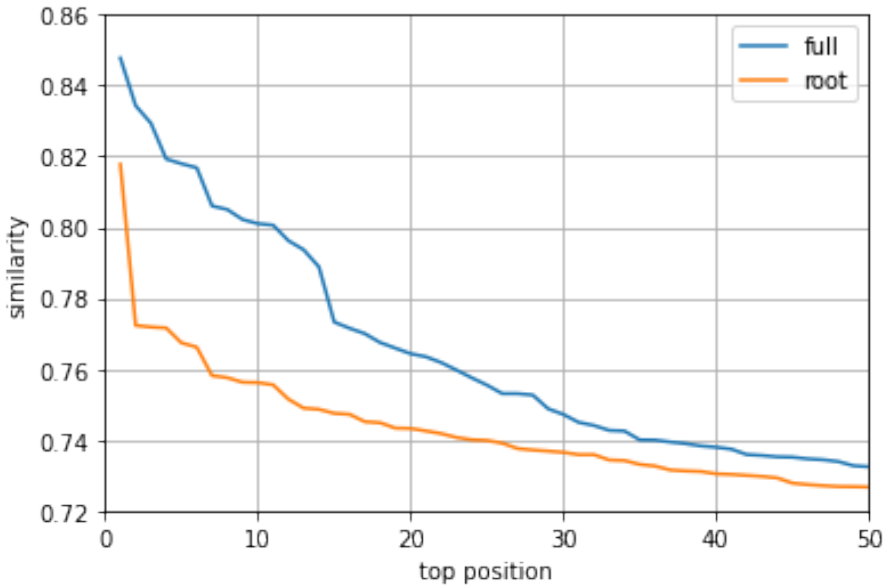


Table 3. Pares gerados no cálculo da similaridade

Tipo da sentença	Pares gerados
Sentenças originais	1345
Sentenças pré-processadas	1923

No entanto, a quantidade de pares e o grau de similaridade gerado não reflete a similaridade ao comparar o par de sentenças que originaram o valor. Um exemplo onde o cálculo do grau de similaridade resultou em um número muito alto e que não reflete na prática são o *tweet* e a frase de um dos artigos apresentada a seguir.

***Tweet*:** "*The Flu and COVID are both respiratory viruses with the same modes of transmission If the masks and social distancing worked for the neareradication of the flu what do you think it should have done for COVID Wake Up*"

Artigo: "*Nasal swab samples were first tested by a diagnosticuse viral panel xTAG Respiratory Viral Panel to qualitatively detect 12 common respiratory viruses and subtypes including coronaviruses influenza A and B viruses respiratory syncytial virus parainfluenza virus adenovirus human metapneumovirus and enterovirus/rhinovirus*"

O cálculo do cosseno sobre suas representações vetoriais resulta no valor "0.817779004573822", o mais alto obtido na análise e muito próximo de 1, o que significa que as duas sentenças são muito similares, mas que não é verdadeiramente o caso. Esta situação ocorre em todas as comparações do trabalho, o cálculo de similaridade entre *tweets* e artigos não é efetivo e não é possível afirmar o *matching* de informações. Buscando avaliar locais de falha para a análise feita no presente trabalho, algumas hipóteses são levantadas.

A primeira trata da parte sintática das sentenças. Uma das causas possíveis para este resultado pode estar relacionada ao conflito na forma de escrita das duas fontes, tornando difícil para o modelo a compreensão do contexto da sentença para uma melhor avaliação e atribuição de valor às *word embeddings*. A dificuldade do modelo com a compreensão de formas sintáticas distintas também trás a tona a segunda hipótese, que trata de uma inadequação do modelo pré-treinado escolhido para o presente trabalho, que pode não estar treinado e otimizado com um *dataset* que se enquadre nas avaliações realizadas.

11. Conclusão e trabalhos futuros

O presente trabalho busca o *matching* de dados a partir de uma análise de similaridade entre os dados extraídos de *tweets* com dados de artigos científicos através da ferramenta desenvolvida, o *Tweet-Nature Similarity Analyzer*, com o objetivo de avaliar se *tweets* publicados propagam informações em conformidade com o conteúdo dos artigos ou não.

Para isso, a construção de Scrapers para dados do Twitter e de artigos da Nature foram necessárias para popular uma base de dados utilizada como fonte para o estudo. A biblioteca Tweepy foi utilizada como interface para a busca por *tweets* da API do Twitter para obter as publicações mais recentes sobre o tema abordado: "máscaras e coronavírus".

Com uma estrutura de fácil manipulação, o Scrapy com suas Spiders é o responsável pela extração dos artigos da revista Nature e, com o auxílio do XPath, foi possível extrair as informações dos documentos utilizados como amostra de forma simplificada. Além disso, com o auxílio do Docker, pôde-se configurar o ambiente do PostgreSQL para modelá-lo, sendo essa a parte central e estrutural do trabalho, para que o mesmo esteja apto a armazenar os dados extraídos pelos Scrapers.

Contudo, os métodos envolvidos no *matching* não corroboraram um resultado positivo do trabalho. O grau de importância entregue pelo TF-IDF, calculado em uma amostra coletada toda sobre o mesmo tópico fez com que o método não fosse o mais adequado para resultado esperado, diminuindo o grau de termos importantes que, pelo fato de estarem presentes em todos os documentos utilizados, se tornavam comuns e, portanto, eram classificados como pouco relevantes.

O Sentence Transformers se mostrou muito versátil em sua utilização junto de um modelo pré-treinado, necessitando de poucos ajustes adicionais para obter a funcionalidade configurada. No entanto, a função cosseno do Sentence Transformers teve seus resultados também distintos do esperado. Sentenças de significados distintos foram classificadas em um alto grau de similaridade, que não corresponde com o resultado esperado para este trabalho. Dessa forma, outros modelos podem ser testados afim de obter melhores resultados sobre esta classificação. Assim, o presente trabalho conclui parte dos seus objetivos, que dizem respeito à coleta e armazenamento de dados, restando somente o aprimoramento do algorítmo de *matching* e a exibição dos resultados de maneira visual, objetivo dependente do anterior.

Como trabalhos futuros para o *Tweet-Nature Similarity Analyzer*, pode-se sugerir os seguintes itens:

- Analisar e propor outras estratégias ou métodos de similaridade semântica, bem como outros modelos pré-treinados capazes de realizarem uma melhor classificação de sentenças como forma de sanar a imprecisão dos modelos estudados no presente trabalho;
- Projetar *embeddings* gerados em um plano vetorial para avaliar a disposição dos mesmos;
- Gerar uma núvem de palavras desconhecidas para avaliar como o modelo do SBERT escolhido está percebendo os dados recebidos;
- Gerar uma visualização dos resultados obtidos do *matching*, com o objetivo de facilitar a visualização da classificação dos dados;

Por fim, o código da aplicação desenvolvida está disponível na íntegra em um repositório público do GitHub[1].

12. References

References

Allcott, H. and Gentzkow, M. (2017). Social media and fake news in the 2016 election. *Journal of Economic Perspectives*, 31(2):211–36.

[1] `https://github.com/gustavomoser/ftd-tfidf-we`

Barcelos, T. d. N. d., Muniz, L. N., Dantas, D. M., Cotrim Junior, D. F., Cavalcante, J. R., and Faerstein, E. (2021). Análise de fake news veiculadas durante a pandemia de covid-19 no brasil. *Revista Panamericana de Salud Pública*, 45(65).

Grazziotim, L. B. (2022). Agrupamento de sentenças semanticamente similares aplicado à descoberta de novas intenções para chatbots cognitivos.

Guarise, L. and Rezende, S. O. (2019). Detecção de notícias falsas usando técnicas de deep learning.

Hugging Face (2022). Hugging face.

Jurafsky, D. and Martin, J. H. (2021). *Speech and language processing (3rd Edition Draft): an introduction to natural language processing, computational linguistics, and speech recognition*.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

Monteiro, R., Nogueira, R., Soares, S., and Anderle, D. (2019). Classificação de fake news com textos de notícias em língua portuguesa integrando data warehousing e machine learning.

Monteiro, R. A., Santos, R. L. S., Pardo, T., Almeida, T. A., Ruiz, E., and Vale, O. A. (2018). Contributions to the study of fake news in portuguese: New corpus and automatic detection results. In *PROPOR*.

Nature. Formatting guide.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks.

Roveda, U. (2020). Google colab: o que é, como usar e quais são as vantagens?

Scrapy (2021). Scrapy at glance.

Sentence Transformers (2022). all-mpnet-base-v2.

Smith, K. (2020). 60 incredible and interesting twitter stats and statistics.

UKPLab, U. K. P. L. (2022). Sentence transformers: Multilingual sentence, paragraph, and image embeddings using bert & co.